PyPlanet Documentation

Release 0.3.3

PyPlanet (Tom Valk)

May 30, 2017

User Documentation

PyPlanet is a Maniaplanet Dedicated Server Controller that works on Python 3.5 and later. Because Maniaplanet is using an system that can be event based we use AsyncIO to provide a event loop and have simultaneously processing of received events from the dedicated server.

Features:

- Core: Super fast and 'event' driven based on Python 3.5 `asyncio` eventloop.
- Core: Stable and well designed core and apps system. (Inspired by Django).
- Core: All apps will handle the game experience.
- Core: Adjustable settings for all your apps.
- Core: Supports **Trackmania** and **Shootmania**, **Scripted only!**
- App: Local Records, including widget + list.
- App: Dedimania Records, including widget + list.
- App: Admin Commands, Providing with basic commands and control for maintaining your server.
- App: Karma, Let your players vote on your maps!
- App: Jukebox, Let your players 'juke' the next map.
- App: ManiaExchange, Simply add your maps directly from Mania-Exchange.
- App: Players, This app shows messages when players join and leave.
- App: Transactions, Donate planets to the server, show number of planets on server and pay out players.
- App: Live Rankings, Show the live rankings of the game mode. (Trackmania).

Do you want to install PyPlanet, head towards our *Getting Started Manual*

The code is open source, and available on GitHub.

The main documentation for the site is organized into a couple sections:

- *User Documentation*
- *Apps Documentation*
- *About PyPlanet*

Information about development of apps and the core is also available under:

- *Developer Documentation*

CHAPTER 1

Getting Started

Installation

Requirements

PyPlanet runs on Python 3.5 and later. Most linux distributions contain default packages or will come with Python preinstalled. To be 100% sure you have to check if you have Python 3 and your version is above 3.5.

Summary of requirements:

- Python 3.5+ and pip 9+ (see bellow for upgrading pip).
- Virtualenv (see: http://pythoncentral.io/how-to-install-virtualenv-python/)
- MySQL Server or PostgreSQL Server.
- Maniaplanet Dedicated MP4+, local or remote.

Installing operating system requirements

For some libraries, like crypto are some native libraries and build tools required.

- Ubuntu: `sudo apt-get install build-essential libssl-dev libffi-dev python3-dev zlib1g-dev`
- Fedora/RHEL: `sudo yum install gcc libffi-devel python3-devel openssl-devel zlib`.
- Windows: Run as Admin: `pip install cryptography`

Tip: If you still get errors with installing with pip, please take a look at: https://cryptography.io/en/latest/installation/#building-cryptography-on-linux

If you are on Ubuntu 16.04 or later you can also use our wrapper bash script that automatically installs required os packages.

```
bash <(curl -s https://raw.githubusercontent.com/PyPlanet/PyPlanet/master/
docs/scripts/setup.sh)
```

1. Check your Python and PIP version

First of all you have to check if your operating system has Python 3.5 or higher installed. To find out, type the following commands in the command shell.

```
python3 --version
# OR
python --version
```

The output should show this `Python 3.5.2` and the version number should be **3.5 or higher**! If this is not the case you could check if your operating system has Python 3.5 support from it's package manager. Ubuntu 16.04 and higher has Python 3.5, Debian 8 has no 3.5.

Windows, download Python 3.6 from the site: https://www.python.org/downloads/

PyEnv

If your operating system doesn't provide you 3.5 or higher, you have to use PyEnv. To install PyEnv execute the following commands:

```
# Execute this as the user you want to install PyPlanet for
curl -L https://raw.githubusercontent.com/pyenv/pyenv-installer/master/bin/pyenv-
↪installer | bash
```

After installing you would have to edit your ~/.bashrc file and add the following lines:

```
export PATH="~/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

Restart your SSH session right now to activate the PyEnv installation.

Next up is the installation of Python 3.6 with PyEnv, you can do this by executing the following shell commands. This can take some time

```
pyenv install 3.6.1
```

2. Virtual Environment for your installation

We recommend using a *virtualenv* to install PyPlanet, and keep the version separate for multiple projects/dedicated servers. With this method you won't have to upgrade all servers at the same time and don't have any issues with system managed python packages.

In order to create a virtualenv you need to have the virtualenv tools installed. To install virtualenv, execute the following command:

Using 'virtualenv':

```
# Ubuntu + Debian
sudo apt-get install virtualenv

# Generic Other Linux
sudo -H pip install virtualenv
```

```
# Windows (run cmd as administrator)
pip install virtualenv
```

From now on you can create virtualenv environments.

```
# Linux
virtualenv -p python3 env

# Windows:
virtualenv -p [full path to python3 executable] env
```

From now you have to activate the virtualenv, every time you want to execute operations with PyPlanet (such as starting, installing, updating, etc). To activate, use the following commands:

```
# Linux
source env/bin/activate

# PyEnv
pyenv activate pyplanet

# Windows (cmd)
env\Scripts\Activate.bat
```

Using PyEnv

With PyEnv it's slightly different, you have to create a virtualenv, but this virtualenv is not located in the same folder as you are in now.

Create virtualenv with the following command:

```
pyenv virtualenv 3.6.1 pyplanet
# 3.6.1 = your installed python version
# pyplanet = name you will give your virtualenv. Can be anything. remember it of␣
↪course!
```

Activating the virtualenv is pretty easy with PyEnv:

```
pyenv activate pyplanet
# Where pyplanet is your virtualenv name.
```

3. PyPlanet Installation

PyPlanet is published through the Python Package Index (PyPi) and is easy to install with `pip`. **Make sure you activated your virtualenv first!**

```
# Install as root:
pip install pyplanet -U
```

After installing it on your system you can use the pyplanet cli commands. To get help about commands, use `pyplanet help`.

4. Setup Project

After installing PyPlanet on your system, you can't yet start any instances because starting requires you to give up an settings module. You could either provide this with the start command or create a project directory with skeleton files.

We recommend using the `init_project` command to create a local project installation where you can install apps, keep PyPlanet and it's apps up-to-date, provide settings through a useful settings module and provide `manage.py` as a wrapper so you never have to manually provide your settings module.

In the example bellow we will setup a project with the name *canyon_server*. The folder *canyon_server* will be created and skeleton files will be copied.

```
pyplanet init_project canyon_server
```

After setup your project, you have to install or update your dependencies from your local `requirements.txt`. You should also use this command to **upgrade your installation**.

```
pip install -r requirements.txt --upgrade
```

After setting up your project environment your ready to go the the next section bellow.

Warning: If you use *virtualenv* or *pyenv*, make sure you activate it **before you install or update dependencies**!

Configuration

Settings module is where the PyPlanet settings are stored. You provide the settings module by providing the environment variable `PYPLANET_SETTINGS_MODULE`. Most of the times this is set in the `manage.py`.

In most cases this settings module is `settings` and is located at the project root subfolder `settings`.

Split files is the default, based on the CLI project generation. This will create two files inside of the settings module, the one is for apps (`apps.py`) and the other for all base configuration (`base.py`).

Pools are the different instances that will be running from PyPlanet. PyPlanet supports multiple controllers from a single setup and project, and even a start command. We are just spawning subprocesses when you start PyPlanet. More information about this setup and architecture on the Architecture overview.

Warning: In the examples in this document you often find an dictionary with the key being `default`. This is a **Pool aware setting** and is different for every pool.

If you are going to add another pool, you should add the pool name to the keys of the dictionary, and fill the value like it is in the examples given here.

Debug Mode (base.py)

In most cases you don't have to use this setting. This setting is only here for developers. While you are in debug mode, there will be **More verbose output, no reporting of exceptions, and debugging of SQL queries**.

When generating a project with the CLI, you will find this setting to be looking at your environment variable `PYPLANET_DEBUG`. Therefor, enable debug by starting PyPlanet with `PYPLANET_DEBUG=1`. Or changing the setting to `DEBUG = True`.

Note: Please enable `DEBUG` when developing, as it won't send reports to the PyPlanet developers, which needs time to investigate and cleanup.

Pool defining (base.py)

You need to define the pools you want to start and have activated with the `POOLS` list.

```
POOLS = ['default'] # Add more identifiers to start more controller instances.
```

Owners (base.py)

Because you want to have admin access at the first boot, you have to define a few master admin logins here. This is optional but will help you to get started directly after starting. This setting is pool aware.

```
OWNERS = {
  'default': [ 'your-maniaplanet-login', 'second-login' ]
}
```

Database configuration (base.py)

The database configuration is mostly the first setting you will adjust to your needs. Currently PyPlanet has support for these *database drivers*:

- `peewee_async.MySQLDatabase`: Using PyMySQL, a full Python based driver. (Supports MariaDB and PerconaDB).
- `peewee_async.PostgresqlDatabase`: Using a full native Python driver.

Creating database:

You will have to create the database scheme yourself. Make sure you create it with a database collate that is based on UTF-8. We recommend for MySQL: `utf8mb4_unicode_ci` to work with the new symbols in Maniaplanet.

Create MySQL Database by running this command:

```
CREATE DATABASE pyplanet
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;
```

Configuration

Configuration can follow the following examples:

```
DATABASES = { # Using PostgreSQL.
'default': {
    'ENGINE': 'peewee_async.PostgresqlDatabase',
    'NAME': 'pyplanet',
    'OPTIONS': {
      'host': 'localhost',
      'user': 'pyplanet',
```

```
      'password': 'pyplanet',
      'autocommit': True,
    }
  }
}

DATABASES = { # Using MySQL (or MariaDB, PerconaDB, etc).
  'default': {
    'ENGINE': 'peewee_async.MySQLDatabase',
    'NAME': 'pyplanet',
    'OPTIONS': {
      'host': 'localhost',
      'user': 'pyplanet',
      'password': 'pyplanet',
      'charset': 'utf8',
    }
  }
}
```

Dedicated Server (base.py)

This one is pretty important, and pretty simple too. Look at the examples bellow, and you know how to set this up!

```
DEDICATED = {
  'default': {
    'HOST': '127.0.0.1',
    'PORT': '5000',
    'USER': 'SuperAdmin',
    'PASSWORD': 'SuperAdmin',
  }
}
```

Map settings (base.py)

Some of these settings are required to be able to save match settings for example.

```
# Map configuration is a set of configuration options related to match settings etc.
# Matchsettings filename.
MAP_MATCHSETTINGS = {
  'default': 'autosave.txt',
}

# You can set this to a automatically generated name:
MAP_MATCHSETTINGS = {
  'default': '{server_login}.txt',
}
```

Storage (base.py)

This may need some explanation, why is this here? We wanted to be able to run PyPlanet on a separate machine as the dedicated is. But also access files from the dedicated for investigating maps, loading and writing maps and settings.

To be able to make this simple, and robust, we will implement several so called *storage drivers* that will work local or remote. For example: *SFTP*, *FTP*, etc.

Local Dedicated

If you run your dedicated server locally, you should use the following setting:

```
STORAGE = {
  'default': {
    'DRIVER': 'pyplanet.core.storage.drivers.local.LocalDriver',
    'OPTIONS': {},
  }
}
```

Using SFTP/SCP/SSH

If your dedicated server is remote, and you want to give access, you can use the SFTP driver (that works over SSH).

```
STORAGE = {
  'default': {
    'DRIVER': 'pyplanet.core.storage.drivers.asyncssh.SFTPDriver',
    'OPTIONS': {
      'HOST': 'remote-hostname.com',
      'PORT': 22,
      'USERNAME': 'maniaplanet',

      # Using password:
      'PASSWORD': 'only-when-using-password',

      # Using private/public keys:
      'CLIENT_KEYS': [
        '/home/mp/.ssh/id_rsa'
      ],
      'PASSPHRASE': 'optional',

      # Optional:
      'KNOWN_HOSTS': '~/.ssh/known_hosts',
      'KWARGS': {
        'CUSTOM_OPTIONS': 'http://asyncssh.readthedocs.io/en/latest/#sftp-client',
      }
    },
  }
}
```

Note: The SFTP driver has not yet been fully tested. Documentation is available on: http://asyncssh.readthedocs.io/en/latest/#sftp-client

Cache (base.py)

Note: This functionality is not yet implemented. Please don't define `CACHE` setting.

Enabling apps (apps.py)

You can enable apps in the `APPS` setting. This is pretty simple and straight forward. The order doesn't make a difference when starting/loading PyPlanet.

```
APPS = {
  'default': [
    'pyplanet.apps.contrib.admin',
    'pyplanet.apps.contrib.jukebox',
    'pyplanet.apps.contrib.karma',
    'pyplanet.apps.contrib.local_records',
    'pyplanet.apps.contrib.dedimania',
    'pyplanet.apps.contrib.players',
    'pyplanet.apps.contrib.mapinfo',
    'pyplanet.apps.contrib.mx',
    'pyplanet.apps.contrib.transactions',
  ],
}
```

Note: When new contributed apps will come available, you have to manually enable it in your settings. Please take a look at our *Change Log* for details on changes.

Starting PyPlanet

After following the instructions on how to install and configure PyPlanet you are ready to start up the controller itself.

Activate virtualenv

You always have to be in the projects virtualenv!

virtualenv:

```
# Linux / Mac OS
source env/bin/activate

# Windows
env\Scripts\Activate.bat
```

PyEnv:

```
pyenv activate pyplanet
# pyplanet is your virtualenv name here!
```

Start PyPlanet

Head to your projects folder where the file `manage.py` is located in your terminal and execute the following command:

```
./manage.py start
```

This will start your PyPlanet setup.

CHAPTER 2

Upgrading PyPlanet

Upgrading an existing installation isn't difficult at all. The only thing you really need to be careful about is the breaking changes.

Before upgrading, please check your existing version, and check the *Change Log Document*.

Note: We assume you installed PyPlanet with PyPi and initiated your project folder with `init_project`. If you installed directly from Git, this document may not be suited for you.

1. Check requirements.txt

In your project root you will find a file called `requirements.txt`. This file is the input of the `pip` manager in the next commands. So it needs to be well maintained.

By default you will see something like this:

```
pyplanet>=0.0.1,<1.0.0
```

This will tell `pip` to install a PyPlanet version above 0.0.1, but under 1.0.0. This way you will prevent sudden breaking changes that may occur in big new releases, or breaking changes that were introduced to a major Maniaplanet update.

If you want to upgrade to a newer major version, for example 1.2.0 to 2.0.0. you have to change these numbers here. If not, continue to the next step

2. Activate env

If you use `virtualenv` or `pyenv` it's now time to activate your virtual environment. Do so with the commands.

```
# Linux
source env/bin/activate

# PyEnv
pyenv activate pyplanet

# Windows
env\Scripts\Activate.bat
```

3. Upgrade PyPlanet core

Now you can run the `pip` command that will upgrade your installation.

```
pip install -r requirements.txt --upgrade
```

Warning: You may find errors during installation, make sure you have `openssl`, `gcc`, `python development` installed on your os! See the installation manual on how to install this.

4. Upgrade settings

See the changelog for new or updated settings and apply the changes now.

5. Upgrade apps setting

It can be possible that we introduced new apps in the update. You will find this in the changelog, and all newest apps will always be provided in the documentation.

On the configuration page you will always find the latest apps settings entries.

6. Start PyPlanet

At the next start it will apply any database migrations automatically.

CHAPTER 3

Migrating from old controller

Migrating from Xaseco2

We provide a basic convert procedure to convert your database from XAseco2 to PyPlanet. You will keep these data:

- Player basic information.
- Driven times by players.
- Map basic information.
- Local records. (`records` table).
- Karma.

As we don't have anything yet that can hold statistics except the times table (`rs_times`), we cannot convert these unfortunately. We will soon have a store for player stats, like donations, total played time, etc.

Command to convert, Change the parameters:

```
./manage.py db_convert --pool default --source-format xaseco2 --source-db-username␣
↪root --source-db-name xaseco2
```

Note: For additional arguments, see ./manage.py db_convert –help

Warning: This has not yet been fully tested with several installations. **Make sure your source is utf8**.

CHAPTER 4

Admin

Information

Name: `pyplanet.apps.contrib.admin`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

Features

This app includes the main admin features PyPlanet has to offer. It's features can be seperated in to these three areas:

- Maps: skip, restart
- Players: mute, kick, ban
- Server: set server/spectator password

Commands

PyPlanet

Reboot PyPlanet Pool Process

Command: `//reboot`

Parameters: None.

Functionality: Reboot pyplanet pool process.

Required permission: `admin:reboot`, requires admin level 3.

Maps

Skip map

Command: `//next` / `//skip`

Parameters: None.

Functionality: Skips to the next map.

Required permission: `admin:next`, requires admin level 1.

Restart map

Command: `//restart` / `//res` / `//rs`

Parameters: None.

Functionality: Restarts the current map.

Required permission: `admin:restart`, requires admin level 1.

Replay map

Command: `//replay`

Parameters: None.

Functionality: Queue the current map to be replayed

Required permission: `admin:replay`, requires admin level 1.

Add Local map

Command: `//add local`

Parameters:

- Local file name or path.

Functionality: Add map from local server disk.

Required permission: `admin:add_local`, requires admin level 2.

Write Map list

Command: `//writemaplist` / `//wml`

Parameters:

- Optional match settings file. Will use the file from your settings if not provided!

Functionality: Write maplist to match settings file.

Required permission: `admin:write_map_list`, requires admin level 2.

Remove Map

Command: `//remove`

Parameters:

- Map number given, the ID column from database. If not given, the current map will be removed!

Functionality: Remove map from loadedd map list. (Doesn't write the maplist to disk!). This command doesn't remove the actual map file!

Required permission: `admin:remove_map`, requires admin level 2.

Erase Map

Command: `//erase`

Parameters:

- Map number given, the ID column from database. If not given, the current map will be removed!

Functionality: Remove map from loadedd map list. (Doesn't write the maplist to disk!). Also removes the map file from the disk!

Required permission: `admin:remove_map`, requires admin level 2.

Players

Force player to spec

Command: `//forcespec`

Parameters:

- Player login.

Functionality: Force player into spectator.

Required permission: `admin:force_spec`, requires admin level 1.

Mute player

Command: `//mute` / `//ignore`

Parameters:

- Player login.

Functionality: Mutes the player, messages won't appear in server chat.

Required permission: `admin:ignore`, requires admin level 1.

Unmute player

Command: `//unmute` / `//unignore`

Parameters:

- Player login.

Functionality: Unmutes the player, messages will appear in server chat again.

Required permission: `admin:unignore`, requires admin level 1.

Kick player

Command: `//kick`

Parameters:

- Player login.

Functionality: Kicks the player from the server.

Required permission: `admin:kick`, requires admin level 1.

Ban player

Command: `//ban`

Parameters:

- Player login.

Functionality: Bans the player from the server.

Required permission: `admin:ban`, requires admin level 2.

Unban player

Command: `//unban`

Parameters:

- Player login.

Functionality: Unbans the player from the server.

Required permission: `admin:unban`, requires admin level 2.

Change user admin level

Command: `//level`

Parameters:

- Player login.
- (Optional) Level: 0 = player, 1 = operator, 2 = admin, 3 = master admin. Leave empty to remove level (0).

Functionality: Changes the admin permission level of the player.

Required permission: `admin:manage_admins`, requires admin level 2.

Game Flow

Force round to end

Command: `//endround`

Parameters: None

Functionality: Force the trackmania round to an end.

Required permission: `admin:end_round`, requires admin level 2.

Force WarmUp round to end

Command: `//endwuround`

Parameters: None

Functionality: Force the trackmania WarmUp round to an end.

Required permission: `admin:end_round`, requires admin level 2.

Force WarmUp to an end

Command: `//endwu`

Parameters: None

Functionality: Force the whole WarmUp to an end.

Required permission: `admin:end_round`, requires admin level 2.

Set rounds points (Points repartition)

Command: `//pointsrepartition` / `//pointsrep`

Parameters:

- Points per place, top to bottom, separated with either spaces or commas.

Functionality: Set the rounds points (points per player and place it ends in an round).

Required permission: `admin:points_repartition`, requires admin level 2.

Server

Set server name

Command: `//servername`

Parameters:

- Server name.

Functionality: Changes the server name.

Required permission: `admin:servername`, requires admin level 2.

Set game mode

Command: `//mode`

Parameters:

- Game mode 'ta', 'laps', 'rounds', 'cup' or any script name (e.g. 'Rounds.Script.txt')

Functionality: Changes the server game mode script.

Required permission: `admin:mode`, requires admin level 2.

Get/set game mode settings

Command: `//modesettings`

Parameters: None, or: * Setting name * New setting value

Functionality: Displays a list of current mode settings (no parameters) or changes a setting according with the given parameters.

Required permission: `admin:mode`, requires admin level 2.

Set server password

Command: `//setpassword` / `//srvpass`

Parameters:

- Server password (none or empty for no password).

Functionality: Changes the server password.

Required permission: `admin:password`, requires admin level 2.

Set server password

Command: `//setspecpassword` / `//spectpass`

Parameters:

- Spectator password (none or empty for no password).

Functionality: Changes the spectator password.

Required permission: `admin:password`, requires admin level 2.

Signal handlers

None.

CHAPTER 5

Jukebox

Information

Name: `pyplanet.apps.contrib.jukebox`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

Features

This app enables players to schedule maps from the maplist to be played next.

Commands

Display maplist

Command: `/list`

Parameters: None or search string.

Functionality: Displays a list of maps currently on the server. First parameter added to command will search the list accordingly.

Required permission: None.

Display jukebox list

Command: `/jukebox list` / `/jukebox display`

Parameters: None.

Functionality: Displays a list of maps currently in the jukebox.

Required permission: None.

Drop jukeboxed map

Command: `/jukebox drop`

Parameters: None.

Functionality: Drops the last (if any) map juked by the player from the jukebox.

Required permission: None.

Clear jukebox

Command: `/admin clearjukebox` / `/admin cjb` / `/jukebox clear`

Parameters: None.

Functionality: Clears the current jukebox list.

Required permission: `jukebox:clear`, requires admin level 1.

Signal handlers

Podium start

Signal: `pyplanet.apps.core.maniaplanet.callbacks.flow.podium_start`

Functionality: Sets the next map to be the first one in the jukebox.

CHAPTER 6

Karma

Information

Name: `pyplanet.apps.contrib.karma`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

Features

This app enables players to vote on maps and provides a karma widget.

Commands

Display votes

Command: `/whokarma`

Parameters: None.

Functionality: Displays a list of votes cast on the current map.

Required permission: None.

Signal handlers

Map begin

Signal: `pyplanet.apps.core.maniaplanet.callbacks.map.map_begin`

Functionality: Retrieves votes for the new map and updates the karma widget.

Player chat

Signal: `pyplanet.apps.core.maniaplanet.callbacks.player.player_chat`

Functionality: Handles chat-based voting (++ or −−).

Player connect

Signal: `pyplanet.apps.core.maniaplanet.callbacks.player.player_connect`

Functionality: Displays the karma widget for the connecting player.

CHAPTER 7

Local Records

Information

Name: `pyplanet.apps.contrib.local_records`

Depends on: `core.maniaplanet`

Game: TrackMania

Features

This app enables players to have their map records stored and displays the records in a widget.

Commands

Display local records

Command: `/records`

Parameters: None.

Functionality: Displays a list of local records on the current map.

Required permission: None.

Signal handlers

Map begin

Signal: `pyplanet.apps.core.maniaplanet.callbacks.map.map_begin`

Functionality: Retrieves records for the new map and updates the widget.

Player connect

Signal: `pyplanet.apps.core.maniaplanet.callbacks.player.player_connect`

Functionality: Displays the records widget for the connecting player.

Player finish

Signal: `pyplanet.apps.core.trackmania.finish`

Functionality: Registers new records.

CHAPTER 8

Dedimania Records

Information

Name: `pyplanet.apps.contrib.dedimania`

Depends on: `core.maniaplanet`

Game: TrackMania

Mode: TimeAttack + Rounds

Features

This app enables players to have their map records stored at Dedimania.net. Displays widget + list for records.

Setup:

1. Make sure you generate a Dedimania Code for your server.
2. Start PyPlanet with this app enabled.
3. Type //settings and edit the two settings for dedimania, paste the code in the code entry.
4. Save and restart PyPlanet.

> **Warning:** The Dedimania App has not yet been fully tested with Rounds and Laps modes.

Commands

Signal handlers

Map begin

Signal: `pyplanet.apps.core.maniaplanet.callbacks.map.map_begin`

Functionality: Retrieves records for the new map and updates the widget.

Map start

Signal: `pyplanet.apps.core.maniaplanet.callbacks.map.map_start`

Functionality: Used to handle map restarts with saving of dedimania records.

Map end

Signal: `pyplanet.apps.core.maniaplanet.callbacks.map.map_end`

Functionality: Used to save dedimania records.

Player connect

Signal: `pyplanet.apps.core.maniaplanet.callbacks.player.player_connect`

Functionality: Displaying widget + sending dedimania request.

Player disconnect

Signal: `pyplanet.apps.core.maniaplanet.callbacks.player.player_connect`

Functionality: Sending dedimania request.

Player finish

Signal: `pyplanet.apps.core.trackmania.finish`

Functionality: Registers new records.

CHAPTER 9

Map Info

Information

Name: `pyplanet.apps.contrib.mapinfo`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

Features

Displays basic map information in widget.

Commands

None.

Signal handlers

Map begin

Signal: `pyplanet.apps.core.maniaplanet.callbacks.map.map_begin`

Functionality: Updates widget with new map information.

Player connect

Signal: `pyplanet.apps.core.maniaplanet.callbacks.player.player_connect`

Functionality: Displays the map info widget for the connecting player.

CHAPTER 10

Players

Information

Name: `pyplanet.apps.contrib.players`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

Features

This app provides the playerlist UI.

Commands

Display playerlist

Command: `/players`

Parameters: None.

Functionality: Displays a list of players currently on the server.

Required permission: None.

Signal handlers

None.

CHAPTER 11

ManiaExchange

Information

Name: `pyplanet.apps.contrib.mx`

Depends on: `core.maniaplanet`

Game: Any

Features

Adding maps from Mania-Exchange.

Commands

Add map(s) from MX

Set server password

Command: `//add mx`

Parameters:

- ManiaExchange ID(s). One or more with space between it.

Functionality: Adding maps from ManiaExchange to the server.

Required permission: `mx:add_remote`, requires admin level 3.

CHAPTER 12

Transactions

Activate with adding `'pyplanet.apps.contrib.transactions.app.Transactions'` to your apps.py

Information

Name: `pyplanet.apps.contrib.transactions`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

Features

Donate, show planets on server and payout players.

Commands

Donate

Command: `/donate`

Parameters:

- Amount of planets.

Functionality: Donate planets to the server.

Required permission:

-

Get amount of planets on server

Command: `//planets`

Parameters: None

Functionality: Get planet

Required permission: `admin:planets`, requires admin level 3.

Pay planets to player

Command: `//pay`

Parameters:

- Player login
- Amount of planets

Functionality: Pay planets to player.

Required permission: `admin:pay`, requires admin level 3.

Signal handlers

Map begin

Signal: `pyplanet.apps.core.maniaplanet.callbacks.other.bill_updated`

Functionality: Update bill signal

CHAPTER 13

Live Rankings

Information

Name: `pyplanet.apps.contrib.live_rankings`

Depends on: `core.maniaplanet`

Game: TrackMania

Features

This app enables the live rankings widget for the game modes:

- Laps (Live cp statistics).
- Rounds (Match sum of points).
- TimeAttack (Top times of players).
- Cup & Team (Points gathered).

Installation

Just add this line to your `apps.py` file:

```
APPS = {
  'default': [
    '...',
    'pyplanet.apps.contrib.live_rankings',  # Add this line.
    '...',
  ]
}
```

Commands

-

Signal handlers

Map begin

Signal: `pyplanet.apps.core.maniaplanet.callbacks.map.map_start`

Functionality: Clears rankings and widget

Player finish

Signal: `pyplanet.apps.core.trackmania.callbacks.finish`

Functionality: Process and update widget.

Player waypoint

Signal: `pyplanet.apps.core.trackmania.callbacks.waypoint`

Functionality: Process and update widget.

Player give up

Signal: `pyplanet.apps.core.trackmania.callbacks.give_up`

Functionality: Set the time to DNF in specific modes.

Player connect

Signal: `pyplanet.apps.core.maniaplanet.callbacks.player.player_connect`

Functionality: Display widget.

Scores

Signal: `pyplanet.apps.core.trackmania.callbacks.scores`

Functionality: Update the widget with the driven scores.

CHAPTER 14

Architecture & Design

Contents

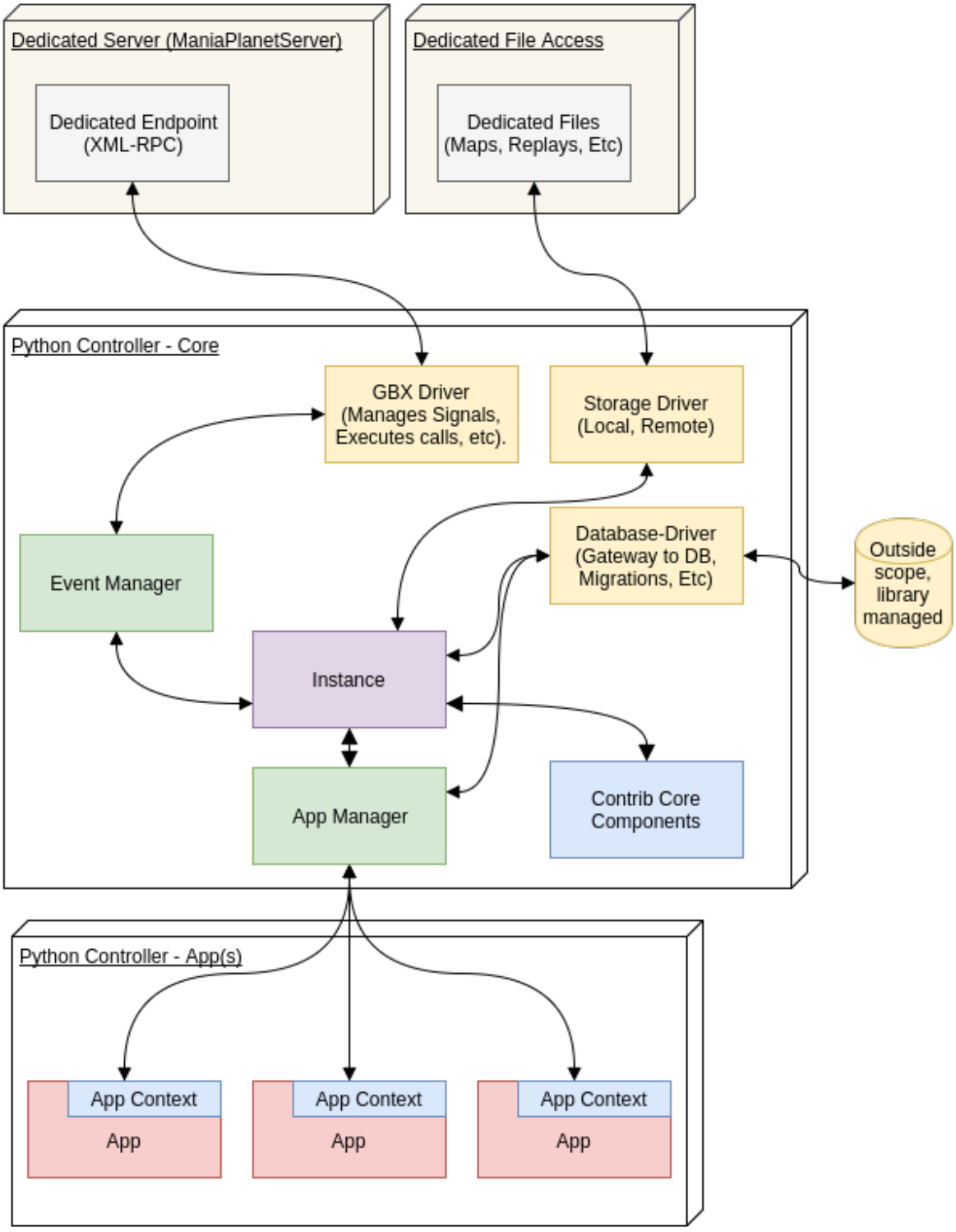
Core Architecture

The architecture of the core and plugins is described in the sections bellow.

Inspiration.

While developing the Core we did look at how Django is managing their so called Apps. Because these apps are self contained applications on it's own, we also call it Apps.

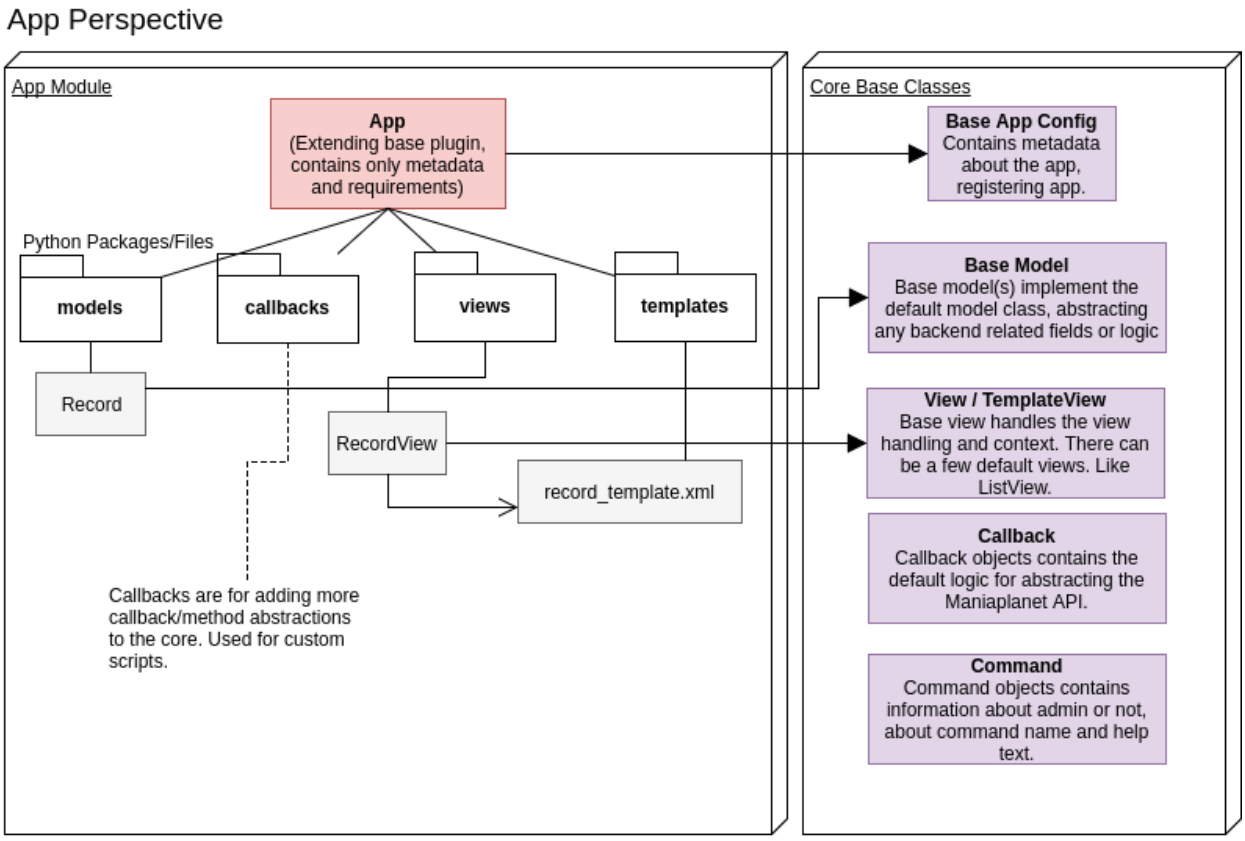
Global Overview

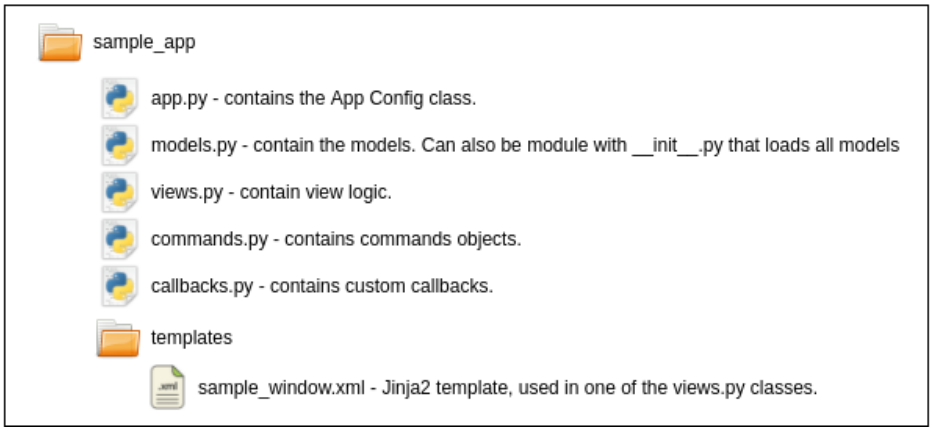


Note: This image is only describing the most important core components, some components are not shown here.

Apps Architecture

More information about the apps itself, please go to *Apps Dev Documentation*





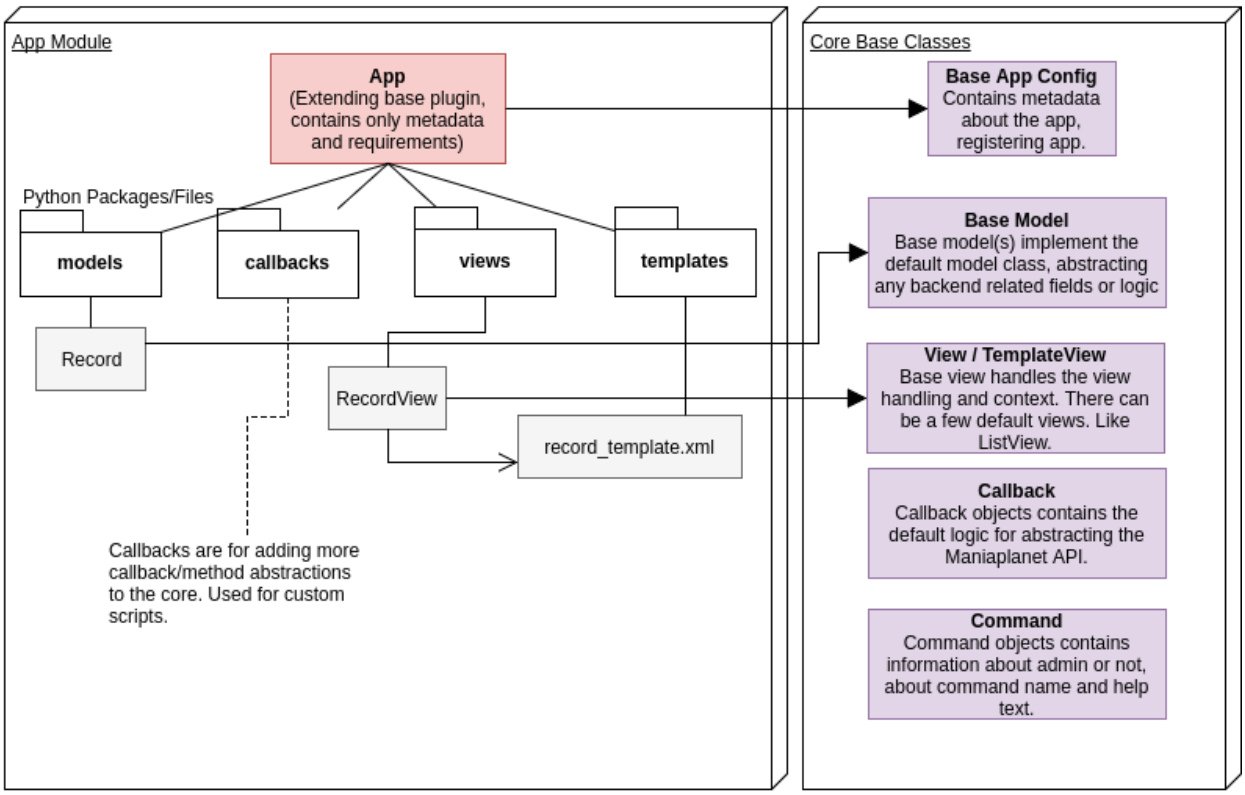
CHAPTER 15

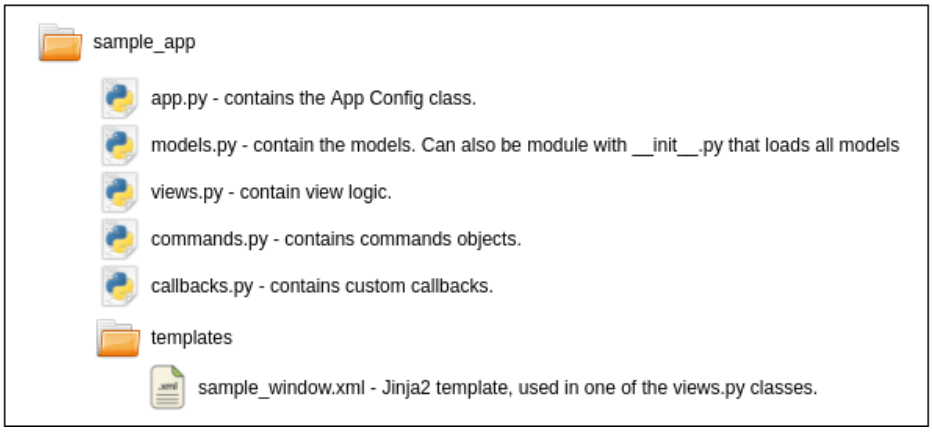
App Development

Contents

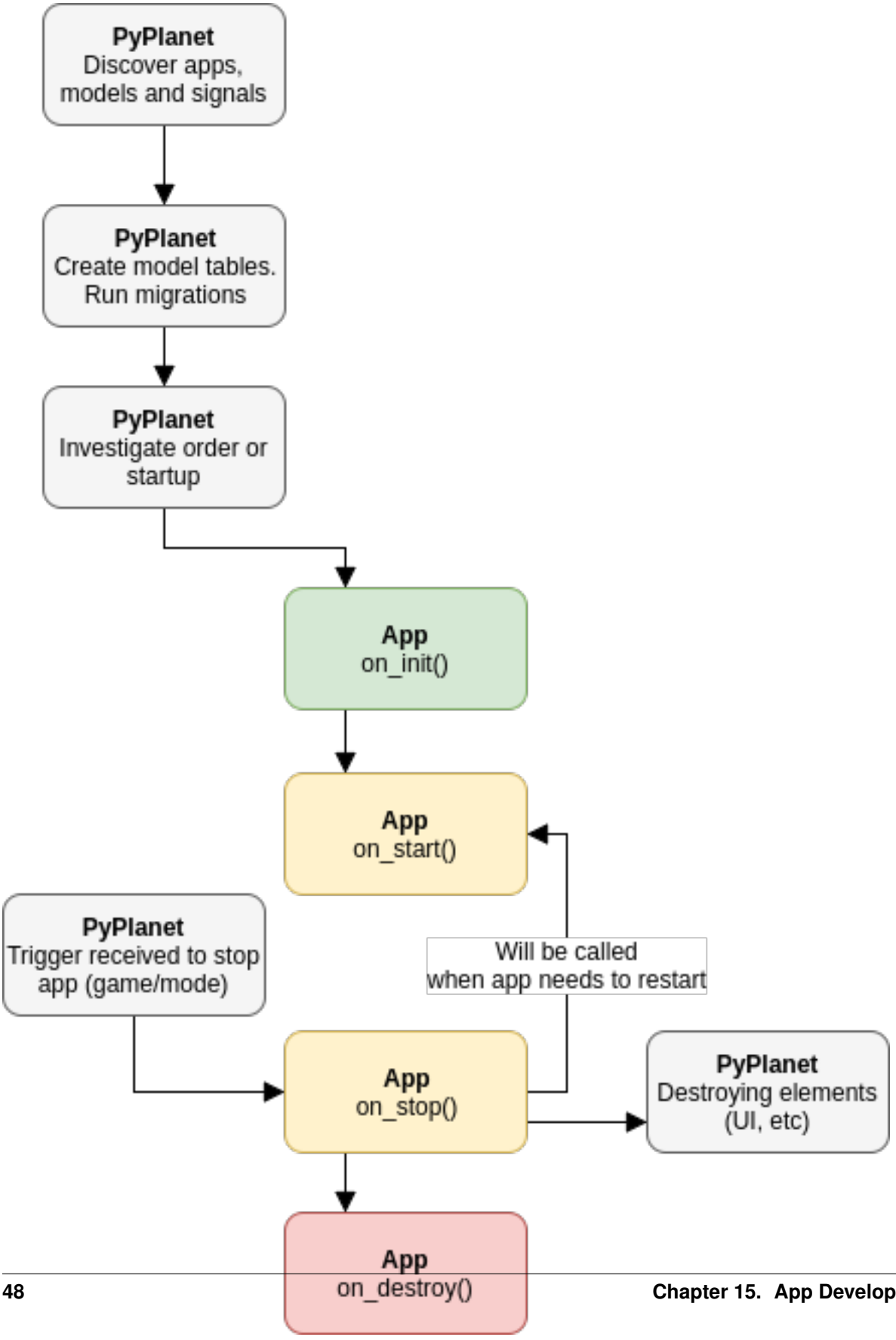
Apps Architecture

App Perspective





Life Cycle


PyPlanet
Discover apps, models and signals
PyPlanet
Create model tables. Run migrations
PyPlanet
Investigate order or startup
App
on_init()
App
on_start()
PyPlanet
Trigger received to stop app (game/mode)
Will be called when app needs to restart
App
on_stop()
PyPlanet
Destroying elements (UI, etc)
App
on_destroy()

Warning: Currently the life cycle **isn't fully implemented**. Only the `on_init` and `on_start` will be called, but please prepare your app to support the following life cycle methods.

on_init

The `on_init()` is called the moment after the apps have been ordered at the dependency trees. This means, there is not yet a stable point to communicate to apps, so it should only initiate local actions, such as clearing variables, initing related services (like startup of http server).

The `on_init()` method is a *coroutine* and will be waited on before starting the other apps init action.

on_start

The `on_start()` is called at the moment all apps, models and other components are ready and the apps should be started. In the method you should init the receivers inside of your app, make an active operation that would init remote connections. For example, you would really like to start showing UI for all players, or initiate local variables based on other apps or the player manager.

The `on_start()` method is a *coroutine* and will be waited on.

on_stop

The `on_stop()` is called when stopping the app internally (so not when exitting PyPlanet!). Some situations like game mode switching will make sure that no apps are being active at the moment of playing an incapable game-mode, game or another app is unloaded that was depending on your app.

PyPlanet will make sure your UI elements are hide from your players, so you don't have to do this. But remember that the app could start at any time, meaning that some context would not be valid anymore, and you should take care of this in the `on_start()` again.

The `on_stop()` method is a *coroutine* and will be waited on.

on_destroy

This method is only called when the app is going to be removed from memory, just before. Mostly only used to save some data.

The `on_destroy()` method is a *coroutine* and will be waited on.

Create app

You can create an app in different places. For private apps we recommend using the `apps` folder in your root project directory.

If you are planning to develop an app for other servers and you want to publish it on PyPi for example, we advise to create your own module folder in your development project root.

Tip: You can use the CLI tool to generate an API module for you.

```
pyplanet init_app app_module
```

1. Create Config

The main entry is the applications config class itself. It is an extended class of the base *pyplanet.apps.AppConfig*.

You have to create a file named `__init__.py` in your app module containing the implementation of the config class. Example is bellow.

```
class Admin(AppConfig):
  game_dependencies = ['trackmania', 'shootmania']
  # Game dependencies. We will check if the current game is in the list (or).
  # Leave undeclared for everything

  mode_dependencies = ['TimeAttack']
  # All the scripted mode file names that are supported by this app.
  # Leave undeclared for everything

  app_dependencies = ['core.maniaplanet']
  # Dependencies to other apps.
  # We will make sure that the dependent apps are started first!

  def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)

    self.property = 'anything here'

  # Implement the life cycle method if you need them. Make sure you call the super in␣
↪the methods!
```

2. Create models

In the same App module you can either create a single models file calling `models.py` or a module `models`. When you are using the module method, you need to import all the model files in the `models/__init__.py`.

Please take a look at the page *Define models* on how to create model declarations.

3. Add to configuration

Make sure you add your new App to your configuration.

```
APPS = {
  'default': [
    '...',
    'my_app',
    '...',
}
```

4. Enable debug

Make sure you enable the *DEBUG* mode during development, this prevents the PyPlanet team from thinking that your App is giving issues in production environments.

You can enable debug either with using the environment variable `PYPLANET_DEBUG` or by editing the configuration:

```
DEBUG = True
```

5. Start PyPlanet

Your ready to get started. Start PyPlanet!

Context (UI + Settings)

Every app has some special access to components such as settings and UI. This is needed to be able to *unregister* the apps things when it's unloaded/stopped, such as hiding all manialinks.

You can access this from your app instance like this:

```
self.context.ui
```

The way this is implemented will make sure that future updates won't break your local properties in the app class itself. For the full contents of this context, take a look at *App Context Class*.

Contrib + Core access

Inside of your app you can access the instance and it's contribution- and core components. To access the instance you can simply use this code statement:

```
self.instance
```

From there you can access most of the controllers components. For the full list of the properties of the `instance` class. Look at *Instance Class*

Models

Models are defined in either the `app/models.py` file or the `app/models/` folder (with loading from the `app/models/__init__.py`)

Models tables are created at the moment PyPlanet starts for the first time as it sees your model, and not yet have a table. To adjust models you should create migrations.

Define models

You have two base classes where your model class could inherit from, we recommend to use the `TimedModel` most of the times. There are a few exceptions where we recommend the base `Model`, for example glue models. Or very data-intensive or data where you don't need to know when it's created or updated.

The `TimedModel` includes these two fields for every model: `created_at` and `updated_at`. Those two fields will be filled and adjusted automatically when saving/updating.

The `Model` includes no fields and is the very base of the model declaration inherit tree.

For defining fields you can use the asterisk import from peewee to have all Fields available in your file:

```
from peewee import *
```

Examples of model declaration:

```
class Permission(Model):
    namespace = CharField(
        max_length=255,
        null=False,
        help_text='Namespace of the permission. Mostly the app.label.'
    )

    name = CharField(
        max_length=255,
        null=False,
        help_text='Name of permission, in format {app_name|core}:{name}'
    )

    description = TextField(
        null=True, default=None, help_text='Description of permission.'
    )

    min_level = IntegerField(
        default=1, help_text='Minimum required player level to be able to use this␣
↪permission.'
    )

    class Meta:
        indexes = (
            (('namespace', 'name'), True),
        )
```

For more examples take a look at: `pyplanet/apps/core/maniaplanet/models/*.py`. You will find the player and map model here with lots of examples.

For more information about fields please refer to the Peewee documentation: http://peewee.readthedocs.io/en/latest/.

For more information about operations on models, **don't look at the Peewee documentation at first**, but look further in this document.

Fields

Please take a look at: http://peewee.readthedocs.io/en/latest/peewee/models.html#fields

Operations on models

Create new object instance in the database

```
instance = Model(column='value', second_col=True)
await instance.save()
```

Delete instance from database

```
await instance.destroy()
```

Find instance by id or other unique value (search for one instance)

```
instance = await Model.get(id=1)
instance = await Model.get(login='toffe')
```

Find instances (query) by executing query with where condition

```
instances = await Model.execute(Model.select().where(Model.column == 1))
```

More examples will follow, feel free to ask for help on this topic in the meantime.

Warning: We use a customized version of the *Peewee* library to have support for async access to database. Because this reason we had to override some methods or create our own. Please don't take not that if you get a sync code exception that it's not *yet* supported by PyPlanet async wrapper.

Please contact us on Github if you think you have an issue with the Database Layer. It's one of the most important parts of PyPlanet!

Migrations

Migrations of models are handled with the .migrations module contents. It works quite like *Django* migrations work, except it automatically executes the migrations at first boot.

Create migrations

1. To create a migration, go to your app base folder and create a folder (if not yet exist), name the folder `'migrations'`.
2. You should create a new python file with the following name pattern:

   `001_name.py` Where 001 is the migration number, this should be unique and the *name* is a name to represent to the developer.
3. Past the following snippet and change it like you want.

```
sample_field = CharField(default='unknown')

def upgrade(migrator: SchemaMigrator):
    migrate(
        migrator.add_column(TestModel._meta.db_table, 'sample', sample_field)
    )

def downgrade(migrator: SchemaMigrator):
    pass
```

4. Change code as you need, but make sure you define defaults or nullable fields, and make sure you use the `db_table` from the meta class of the model.

5. Make sure you can upgrade at least. Downgrading is not yet included in the scope, but it's better to implement the downgrade as well.
6. Test, make sure it's able to migrate on at least these engines: MySQL or PostgreSQL.

Chat Messages

We implemented an abstraction that will provide auto multicall and auto prefixing for you. You can use the following statements for example:

```
# Send chat message to all players.
await self.instance.chat('Test')

# Send chat message to specific player or multiple players.
await self.instance.chat('Test', 'player_login')  # Sends to single player.
await self.instance.chat('Test', 'player_login', player_instance)  # Sends to both␣
↪players.

# Execute in chain (Multicall).
await self.instance.chat.execute(
  'global_message',
  self.instance.chat('Test', 'player_login'),
  self.instance.chat('Test2', 'player_login2'),
)

# You can combine this with other calls in a GBX multicall:
await self.instance.gbx.multicall(
  self.instance.gbx.prepare('SetServerName', 'Test'),
  self.instance.chat('Test2', 'player_login2'),
)
```

Dedicated/Script methods

From your app you can execute dedicated GBX methods (or scripted methods) with the following methods:

```
# Force player_login into spectator.
await self.instance.gbx('ForceSpectator', 'player_login', 1)

# Execute multiple gbx actions in a multicall (Is way faster).
await self.instance.gbx.multicall(
  self.instance.gbx('Method', 'arg1', 'arg2'),
  self.instance.gbx('Method', 'arg1', 'arg2'),
  self.instance.gbx('Method', 'arg1', 'arg2'),
)
```

Useful references

You might want to look at the following pages as well to get more information:

- *Signal Documentation* is useful when you are going to hook into Maniaplanet.
- *Architecture Overview* is useful when you want to know how the core is acting on some points.

Have any questions or bugs to report? Head towards our *Support page*.

CHAPTER 16

Signals (callbacks)

Contents

Maniaplanet

Flow

pyplanet.apps.core.maniaplanet.callbacks.flow.**loading_map_end = <pyplanet.core.events.callback.Callbac**

Signal Loading Map end.

Code `maniaplanet:loading_map_end`

Description Callback sent when the server finishes to load the map.

Original Callback *Script* Maniaplanet.LoadingMap_End

Parameters **map** (*pyplanet.core.maniaplanet.models.map.Map*) – Map instance from database. Updated with the provided data.

pyplanet.apps.core.maniaplanet.callbacks.flow.**loading_map_start = <pyplanet.core.events.callback.Callb**

Signal Loading Map start.

Code `maniaplanet:loading_map_start`

Description Callback sent when the server starts loading the map.

Original Callback *Script* Maniaplanet.LoadingMap_Start

Parameters **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**match_end = <pyplanet.core.events.callback.Callback object**

Signal Match End.

Code `maniaplanet:match_end`

Description Callback sent when the "EndMatch" section start.

Original Callback *Script* Maniaplanet.EndMatch_Start

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**match_end__end = <pyplanet.core.events.callback.Callback**

Signal Match End. (End event)

Code `maniaplanet:match_end__end`

Description Callback sent when the "EndMatch" section ends.

Original Callback *Script* Maniaplanet.EndMatch_End

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**match_start = <pyplanet.core.events.callback.Callback obj**

Signal Match Start.

Code `maniaplanet:match_start`

Description Callback sent when the "StartMatch" section start.

Original Callback *Script* Maniaplanet.StartMatch_Start

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**match_start__end = <pyplanet.core.events.callback.Callba**

Signal Match Start. (End event)

Code `maniaplanet:match_start__end`

Description Callback sent when the "StartMatch" section end.

Original Callback *Script* Maniaplanet.StartMatch_End

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**play_loop_end = <pyplanet.core.events.callback.Callback o**

Signal Play Loop End.

Code `maniaplanet:play_loop_end`

Description Callback sent when the "PlayLoop" section ends.

Original Callback *Script* Maniaplanet.EndPlayLoop

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**play_loop_start = <pyplanet.core.events.callback.Callbac**

Signal Play Loop Start.

Code `maniaplanet:play_loop_start`

Description Callback sent when the "PlayLoop" section starts.

Original Callback *Script* Maniaplanet.StartPlayLoop

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**podium_end = <pyplanet.core.events.callback.Callback objec**

Signal Podium end.

Code `maniaplanet:podium_end`

Description Callback sent when the podium sequence ends.

Original Callback *Script* Maniaplanet.Podium_End

Parameters **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**podium_start = <pyplanet.core.events.callback.Callback ob**

Signal Podium start.

Code `maniaplanet:podium_start`

Description Callback sent when the podium sequence starts.

Original Callback *Script* Maniaplanet.Podium_Start

Parameters **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**round_end = <pyplanet.core.events.callback.Callback object**

Signal Round Start.

Code `maniaplanet:round_end`

Description Callback sent when the "EndRound" section starts.

Original Callback *Script* Maniaplanet.EndRound_Start

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**round_end__end = <pyplanet.core.events.callback.Callback**

Signal Round Start. (End event)

Code `maniaplanet:round_end__end`

Description Callback sent when the "EndRound" section ends.

Original Callback *Script* Maniaplanet.EndRound_End

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**round_start = <pyplanet.core.events.callback.Callback obj**

Signal Round Start.

Code `maniaplanet:round_start`

Description Callback sent when the "StartRound" section starts.

Original Callback *Script* Maniaplanet.StartRound_Start

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**round_start__end = <pyplanet.core.events.callback.Callba**

Signal Round Start. (End event)

Code `maniaplanet:round_start__end`

Description Callback sent when the "StartRound" section ends.

Original Callback *Script* Maniaplanet.StartRound_End

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**server_end = <pyplanet.core.events.callback.Callback objec**

Signal Server End signal

Code `maniaplanet:server_end`

Description This callback is called when the server script is end. The begin of the event.

Original Callback *Script* Maniaplanet.EndServer_Start

Parameters

- **restarted** – Boolean giving information if the script has restarted.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**server_end__end = <pyplanet.core.events.callback.Callbac**

Signal Server End signal (end event)

Code `maniaplanet:server_end__end`

Description This callback is called when the server script is end. The end of the event.

Original Callback *Script* Maniaplanet.EndServer_End

Parameters

- **restarted** – Boolean giving information if the script has restarted.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**server_start = <pyplanet.core.events.callback.Callback ob**

Signal Server Start signal

Code `maniaplanet:server_start`

Description This callback is called when the server script is (re)started. The begin of the event.

Original Callback *Script* Maniaplanet.StartServer_Start

Parameters

- **restarted** – Boolean giving information if the script has restarted.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**server_start__end = <pyplanet.core.events.callback.Callb**

Signal Server Start signal (end of event).

Code `maniaplanet:server_start__end`

Description This callback is called when the server script is (re)started. The end of the event.

Original Callback *Script* Maniaplanet.StartServer_End

Parameters

- **restarted** – Boolean giving information if the script has restarted.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**status_changed = <pyplanet.core.events.callback.Callback**

Signal Server Status Changed.

Code `maniaplanet:status_changed`

Description Callback sent when the podium sequence ends.

Original Callback *Native* Maniaplanet.Podium_End

Parameters

- **1** (*int*) – Status Code.
- **2** (*str*) – Status Name.

pyplanet.apps.core.maniaplanet.callbacks.flow.**turn_end = <pyplanet.core.events.callback.Callback object>**

Signal Turn End.

Code `maniaplanet:turn_end`

Description Callback sent when the "EndTurn" section starts.

Original Callback *Script* Maniaplanet.EndTurn_Start

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**turn_end__end = <pyplanet.core.events.callback.Callback o**

Signal Turn End. (End event)

Code `maniaplanet:turn_end__end`

Description Callback sent when the "EndTurn" section ends.

Original Callback *Script* Maniaplanet.EndTurn_End

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**turn_start = <pyplanet.core.events.callback.Callback objec**

Signal Turn Start.

Code `maniaplanet:turn_start`

Description Callback sent when the "StartTurn" section starts.

Original Callback *Script* Maniaplanet.StartTurn_Start

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**turn_start__end = <pyplanet.core.events.callback.Callbac**

Signal Turn Start. (End event).

Code `maniaplanet:turn_start__end`

Description Callback sent when the "StartTurn" section ends.

Original Callback *Script* Maniaplanet.StartTurn_End

Parameters

- **count** – Each time this section is played, this number is incremented by one.
- **time** – Server time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.flow.**unloading_map_end = <pyplanet.core.events.callback.Callb**

Signal Unloading of the Map ends.

Code `maniaplanet:unloading_map_end`

Description Callback sent when the server finishes to unload a map.

Original Callback *Script* Maniaplanet.UnloadingMap_End

Parameters **map** (*pyplanet.core.maniaplanet.models.map.Map*) – Map instance from database. Updated with the provided data.

pyplanet.apps.core.maniaplanet.callbacks.flow.**unloading_map_start = <pyplanet.core.events.callback.Ca**

Signal Unloading of the Map starts.

Code `maniaplanet:unloading_map_start`

Description Callback sent when the server starts to unload a map.

Original Callback *Script* Maniaplanet.UnloadingMap_Start

Parameters **map** (*pyplanet.core.maniaplanet.models.map.Map*) – Map instance from database. Updated with the provided data.

Map

pyplanet.apps.core.maniaplanet.callbacks.map.**map_begin = <pyplanet.core.events.callback.Callback object>**

Signal Begin of map.

Code `maniaplanet:map_begin`

Description Callback sent when map begins.

Original Callback *Native* Maniaplanet.BeginMap

Parameters **map** (*pyplanet.apps.core.maniaplanet.models.map.Map*) – Map instance.

pyplanet.apps.core.maniaplanet.callbacks.map.**map_end = <pyplanet.core.events.callback.Callback object>**

Signal End of map.

Code `maniaplanet:map_end`

Description Callback sent when map ends.

Original Callback *Native* Maniaplanet.EndMap

Parameters **map** (*pyplanet.apps.core.maniaplanet.models.map.Map*) – Map instance.

pyplanet.apps.core.maniaplanet.callbacks.map.**map_start = <pyplanet.core.events.callback.Callback object>**

Signal Begin of map. (Scripted!)

Code `maniaplanet:map_begin`

Description Callback sent when map starts (same as begin, but scripted).

Original Callback *Script* Maniaplanet.StartMap_Start

Parameters

- **time** – Time when callback has been sent.
- **count** – Counts of the callback that was sent.
- **restarted** – Is the map restarted.
- **map** (*pyplanet.apps.core.maniaplanet.models.map.Map*) – Map instance.

pyplanet.apps.core.maniaplanet.callbacks.map.**map_start__end = <pyplanet.core.events.callback.Callback o**

Signal Begin of map, end of event. (Scripted!)

Code `maniaplanet:map_start__end`

Description Callback sent when map starts (same as begin, but scripted). End of the event

Original Callback *Script* Maniaplanet.StartMap_End

Parameters

- **time** – Time when callback has been sent.
- **count** – Counts of the callback that was sent.
- **restarted** – Is the map restarted.
- **map** (*pyplanet.apps.core.maniaplanet.models.map.Map*) – Map instance.

pyplanet.apps.core.maniaplanet.callbacks.map.**playlist_modified = <pyplanet.core.events.callback.Callba**

Signal Maplist changes.

Code `maniaplanet:playlist_modified`

Description Callback sent when map list changes.

Original Callback *Native* Maniaplanet.MapListModified

Parameters

- **1** (*int*) – Current map index.
- **2** (*int*) – Next map index.
- **3** (*bool*) – Is List Modified.

Player

pyplanet.apps.core.maniaplanet.callbacks.player.**player_chat = <pyplanet.core.events.callback.Callback o**

Signal Player has been writing a chat entry. When the server writes something we **wont** inform it in here!

Code `maniaplanet:player_chat`

Description Callback sent when a player chats.

Original Callback *Native* Maniaplanet.PlayerChat

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **text** – Text of chat
- **cmd** – Boolean if it's a command. Be aware, you should use the `command` manager for commands!

pyplanet.apps.core.maniaplanet.callbacks.player.**player_connect = <pyplanet.core.events.callback.Callba**

Signal Player has been connected.

Code `maniaplanet:player_connect`

Description Callback sent when a player connects and we fetched our data.

Original Callback *Native* Maniaplanet.PlayerConnect

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **is_spectator** – Boolean determinating if the player joined as spectator.
- **source** – Raw payload, best to not use!

pyplanet.apps.core.maniaplanet.callbacks.player.**player_disconnect = <pyplanet.core.events.callback.C**

Signal Player has been disconnected.

Code `maniaplanet:player_disconnect`

Description Callback sent when a player disconnects.

Original Callback *Native* Maniaplanet.PlayerDisconnect

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **reason** – Reason of leave
- **source** – Raw payload, best to not use!

pyplanet.apps.core.maniaplanet.callbacks.player.**player_info_changed = <pyplanet.core.events.callback**

Signal Player has changed status.

Code `maniaplanet:player_info_changed`

Description Callback sent when a player changes from state or information.

Original Callback *Native* Maniaplanet.PlayerInfoChanged

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance (COULD BE NONE SOMETIMES!).
- **player_login** – Player login string.
- **is_spectator** – Is player spectator (bool).
- **is_temp_spectator** – Is player temporary spectator (bool).
- **is_pure_spectator** – Is player pure spectator (bool).
- **auto_target** – Player using auto target.
- **target_id** – The target player id (not login!).
- **target** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – The target player instance or None if not found/none spectating.
- **flags** – Raw flags.
- **spectator_status** – Raw spectator status.
- **team_id** – Team ID of player.
- **player_id** – Player ID (server id).

User Interface

pyplanet.apps.core.maniaplanet.callbacks.ui.**manialink_answer = <pyplanet.core.events.callback.Callback**

Signal Player has raised an action on the Manialink.

Code `maniaplanet:manialink_answer`

Description Callback sent when a player clicks on an event of a manialink.

Original Callback *Native* Maniaplanet.PlayerManialinkPageAnswer

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **action** – Action name
- **values** – Values (in dictionary).

Warning: Don't use this callback directly, use the abstraction of ''View'' and ''StaticManialink'' to handle events of your manialink!

Other

pyplanet.apps.core.maniaplanet.callbacks.other.**bill_updated = <pyplanet.core.events.callback.Callback o**

Signal Bill has been updated.

Code `maniaplanet:bill_updated`

Description Callback sent when a bill has been updated.

Original Callback *Native* Maniaplanet.BillUpdated

Parameters

- **1** (*int*) – Bill id.
- **2** (*int*) – State.
- **3** (*str*) – State name.
- **4** (*int*) – Transaction id.

pyplanet.apps.core.maniaplanet.callbacks.other.**channel_progression_end = <pyplanet.core.events.calll**

Signal Signal sent when channel progression sequence ends.

Code `maniaplanet:channel_progression_end`

Description Callback sent when the channel progression sequence ends.

Original Callback *Script* Maniaplanet.ChannelProgression_End

Parameters **time** – Time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.other.**channel_progression_start = <pyplanet.core.events.c**

Signal Signal sent when channel progression sequence starts.

Code `maniaplanet:channel_progression_start`

Description Callback sent when the channel progression sequence starts.

Original Callback *Script* Maniaplanet.ChannelProgression_Start

Parameters **time** – Time when callback has been sent.

pyplanet.apps.core.maniaplanet.callbacks.other.**server_chat = <pyplanet.core.events.dispatcher.Signal obj**

Signal Server send a chat message.

Code `maniaplanet:server_chat`

Description Custom signal called when the server outputs a message.

Origin Callback None (via Chat callback).

pyplanet.apps.core.maniaplanet.callbacks.other.**vote_updated = <pyplanet.core.events.callback.Callback o**

Signal Vote has been updated.

Code `maniaplanet:vote_updated`

Description Callback sent when a call vote has been updated.

Original Callback *Native* Maniaplanet.VoteUpdated

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **state** – State name
- **cmd_name** – Command name
- **cmd_param** – Parameter given with command.

Shootmania

Base

Weapons [1-Laser, 2-Rocket, 3-Nucleus, 5-Arrow]

pyplanet.apps.core.shootmania.callbacks.base.**action_custom_event = <pyplanet.core.events.callback.Cal**

Signal Handle Action Custom Event.

Code `shootmania:action_custom_event`

Description Callback sent when an action triggers a custom event.

Original Callback *Script* Shootmania.Event.OnActionCustomEvent

Parameters

- **time** – Time of server when callback is sent.
- **shooter** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Shooter player instance if any
- **victim** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Victim player instance if any
- **actionid** – Action Identifier.
- ***** – Any other params, like `param1`, `param2`, etc...

pyplanet.apps.core.shootmania.callbacks.base.**action_event = <pyplanet.core.events.callback.Callback obj**

Signal Handle Action Event.

Code `shootmania:action_event`

Description Callback sent when an action triggers an event.

Original Callback *Script* Shootmania.Event.OnActionEvent

Parameters

- **time** – Time of server when callback is sent.
- **login** – Player login
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **action_input** – Action input.

pyplanet.apps.core.shootmania.callbacks.base.**on_armor_empty = <pyplanet.core.events.callback.Callback o**

Signal Armor empty, player eliminated.

Code `shootmania:on_armor_empty`

Description Callback sent when a player is eliminated.

Original Callback *Script* Shootmania.Event.OnArmorEmpty

Parameters

- **shooter** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – shooter, Player instance
- **time** – Time of server when callback is sent.
- **weapon** – Weapon number.
- **victim** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – victim, Player instance
- **distance** – Distance between victim and shooter.
- **shooter_position** – Position of shooter.
- **victim_position** – Position of victim.

pyplanet.apps.core.shootmania.callbacks.base.**on_capture = <pyplanet.core.events.callback.Callback object**

Signal Landmark has been captured

Code `shootmania:on_capture`

Description Callback sent when a landmark is captured.

Original Callback *Script* Shootmania.Event.OnCapture

time=source['time'], players=players, landmark=source['landmark']

Parameters

- **time** – Time of server when callback is sent.
- **players** (*pyplanet.apps.core.maniaplanet.models.player.Player[]*) – Player list (instances).
- **landmark** – Landmark information, raw!

pyplanet.apps.core.shootmania.callbacks.base.**on_command = <pyplanet.core.events.callback.Callback object**

Signal On Command

Code `shootmania:on_command`

Description Callback sent when a command is executed on the server.

Original Callback *Script* Shootmania.Event.OnCommand

Parameters

- **time** – Time of server when callback is sent.
- **name** – Name of the command
- **value** (*dict*) – Value in dictionary of the command.

pyplanet.apps.core.shootmania.callbacks.base.**on_fall_damage = <pyplanet.core.events.callback.Callback o**

Signal Fall Damage

Code `shootmania:on_fall_damage`

Description Callback sent when a player suffers fall damage.

Original Callback *Script* Shootmania.Event.OnFallDamage

Parameters

- **time** – Time of server when callback is sent.
- **victim** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – victim, Player instance

pyplanet.apps.core.shootmania.callbacks.base.**on_hit = <pyplanet.core.events.callback.Callback object>**

Signal Player hit.

Code `shootmania:on_hit`

Description Callback sent when a player is hit.

Original Callback *Script* Shootmania.Event.OnHit

Parameters

- **shooter** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – shooter, Player instance
- **time** – Time of server when callback is sent.
- **weapon** – Weapon number.
- **victim** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – victim, Player instance
- **damage** – Damage done.
- **points** – Points scored by hit.
- **distance** – Distance between victim and shooter.
- **shooter_position** – Position of shooter.
- **victim_position** – Position of victim.

pyplanet.apps.core.shootmania.callbacks.base.**on_near_miss = <pyplanet.core.events.callback.Callback obj**

Signal Near Miss.

Code `shootmania:on_near_miss`

Description Callback sent when a player dodges a projectile.

Original Callback *Script* Shootmania.Event.OnNearMiss

Parameters

- **shooter** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – shooter, Player instance
- **time** – Time of server when callback is sent.
- **weapon** – Weapon number.
- **victim** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – victim, Player instance
- **distance** – Distance between victim and shooter.
- **shooter_position** – Position of shooter.
- **victim_position** – Position of victim.

pyplanet.apps.core.shootmania.callbacks.base.**on_shoot = <pyplanet.core.events.callback.Callback object>**

Signal Player shoot.

Code `shootmania:on_shoot`

Description Callback sent when a player shoots.

Original Callback *Script* Shootmania.Event.OnShoot

Parameters

- **shooter** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Shooter, Player instance
- **time** – Time of server when callback is sent.
- **weapon** – Weapon number.

pyplanet.apps.core.shootmania.callbacks.base.**on_shot_deny = <pyplanet.core.events.callback.Callback obj**

Signal Player denies a projectile.

Code `shootmania:on_shot_deny`

Description Callback sent when a player denies a projectile.

Original Callback *Script* Shootmania.Event.OnShotDeny

Parameters

- **time** – Time of server when callback is sent.
- **shooter** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – shooter, Player instance
- **victim** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – victim, Player instance
- **shooter_weapon** – Weapon number of shooter.
- **victim_weapon** – Weapon number of victim that denied the shot.
- **distance** – Distance between victim and shooter.
- **shooter_position** – Position of shooter.
- **victim_position** – Position of victim.

pyplanet.apps.core.shootmania.callbacks.base.**player_added = <pyplanet.core.events.callback.Callback obj**

Signal On player added.

Code `shootmania:player_added`

Description Callback sent when a player joins the server.

Original Callback *Script* Shootmania.Event.OnPlayerAdded

Parameters

- **time** – Time of server when callback is sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **team** – Team nr.
- **language** – Language code, like 'en'.

- **ladder_rank** – Current ladder rank.
- **ladder_points** – Current ladder points.

pyplanet.apps.core.shootmania.callbacks.base.**player_removed = <pyplanet.core.events.callback.Callback o**

Signal On player removed.

Code `shootmania:player_removed`

Description Callback sent when a player leaves the server.

Original Callback *Script* Shootmania.Event.OnPlayerRemoved

Parameters

- **time** – Time of server when callback is sent.
- **login** – Player login string
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.

pyplanet.apps.core.shootmania.callbacks.base.**player_request_action_change = <pyplanet.core.events.**

Signal Player requests action change.

Code `shootmania:player_request_action_change`

Description Callback sent when a player requests to use another action.

Original Callback *Script* Shootmania.Event.OnPlayerRequestActionChange

Parameters

- **time** – Time of server when callback is sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **action_change** – Can be -1 (request previous action) or 1 (request next action)

pyplanet.apps.core.shootmania.callbacks.base.**player_request_respawn = <pyplanet.core.events.callback**

Signal On player request respawn.

Code `shootmania:player_request_respawn`

Description Callback sent when a player presses the respawn button.

Original Callback *Script* Shootmania.Event.OnPlayerRequestRespawn

Parameters

- **time** – Time of server when callback is sent.
- **login** – Player login string
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.

pyplanet.apps.core.shootmania.callbacks.base.**player_throws_object = <pyplanet.core.events.callback.Ca**

Signal Player Throws an object.

Code `shootmania:player_touch_object`

Description Callback sent when a player throws an object.

Original Callback *Script* Shootmania.Event.OnPlayerThrowsObject

Parameters

- **time** – Time of server when callback is sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **object_id** – Object Identifier.
- **model_id** – Model identifier.
- **model_name** – Model name.

pyplanet.apps.core.shootmania.callbacks.base.**player_touches_object = <pyplanet.core.events.callback.C**

Signal Player Touches Object.

Code `shootmania:player_touches_object`

Description Callback sent when a player touches an object.

Original Callback *Script* Shootmania.Event.OnPlayerTouchesObject

Parameters

- **time** – Time of server when callback is sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **object_id** – Object Identifier.
- **model_id** – Model identifier.
- **model_name** – Model name.

pyplanet.apps.core.shootmania.callbacks.base.**player_triggers_sector = <pyplanet.core.events.callback**

Signal Player Triggers Sector.

Code `shootmania:player_triggers_sector`

Description Callback sent when a player triggers a sector.

Original Callback *Script* Shootmania.Event.OnPlayerTriggersSector

Parameters

- **time** – Time of server when callback is sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **sector_id** – Sector Identifier.

pyplanet.apps.core.shootmania.callbacks.base.**scores = <pyplanet.core.events.callback.Callback object>**

Signal Score callback, called after the map. (Around the podium time).

Code `shootmania:scores`

Description Teams and players scores.

Original Callback *Script* Shootmania.Scores

Parameters

- **players** (`list`) – Player score payload. Including player instance etc.
- **teams** (`list`) – Team score payload.

- **winner_team** – The winning team.
- **use_teams** – Use teams.
- **winner_player** – The winning player.
- **section** – Section, current progress of match. Important to check before you save results!!

Elite

Victory Types 1 = time limit reached, 2 = capture, 3 = attacker eliminated, 4 = defenders eliminated.

pyplanet.apps.core.shootmania.callbacks.elite.**turn_end = <pyplanet.core.events.callback.Callback object>**

Signal Elite turn start.

Code `shootmania:elite_turn_end`

Description Information about the ending turn.

Original Callback *Script* Shootmania.Elite.EndTurn

Parameters **victory_type** – Describe how the turn was won. 1 = time limit, 2 = capture, 3 = attacker eliminated, 4 = defenders eliminated

pyplanet.apps.core.shootmania.callbacks.elite.**turn_start = <pyplanet.core.events.callback.Callback objec**

Signal Elite turn start.

Code `shootmania:elite_turn_start`

Description Information about the starting turn.

Original Callback *Script* Shootmania.Elite.StartTurn

Parameters

- **attacker** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance of attacker.
- **defenders** (*pyplanet.apps.core.maniaplanet.models.player.Player[]*) – List with player instances of defenders.

Joust

pyplanet.apps.core.shootmania.callbacks.joust.**player_reload = <pyplanet.core.events.callback.Callback o**

Signal Player reloads its weapon and capture pole.

Code `shootmania:joust_player_reload`

Description Callback sent when a player capture a pole to reload its weapons.

Original Callback *Script* Shootmania.Joust.OnReload

Parameters

- **login** – Player login.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.

pyplanet.apps.core.shootmania.callbacks.joust.**results = <pyplanet.core.events.callback.Callback object>**

Signal End of round with results of Joust round.

Code `shootmania:joust_results`

Description Callback sent at the end of the round with the scores of the two players.

Original Callback *Script* Shootmania.Joust.RoundResult

Parameters **players** (`list`) – Player score list, contains player + score.

`pyplanet.apps.core.shootmania.callbacks.joust.`**selected_players = <pyplanet.core.events.callback.Callba**

Signal Round starts with selected players.

Code `shootmania:joust_selected_players`

Description Callback sent at the beginning of the round with the logins of the players selected to play the round.

Original Callback *Script* Shootmania.Joust.SelectedPlayers

Parameters **players** (*pyplanet.apps.core.maniaplanet.models.player.Player[]*) – Player list (instances).

Royal

`pyplanet.apps.core.shootmania.callbacks.royal.`**player_score_points = <pyplanet.core.events.callback.Ca**

Signal Player score points.

Code `shootmania:royal_player_score_points`

Description Callback sent when a player scores some points.

Original Callback *Script* Shootmania.Royal.Points

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **type** – Type of score, like 'Pole', 'Hit', or 'Survival'.
- **points** – Points that the player gains.

`pyplanet.apps.core.shootmania.callbacks.royal.`**player_spawn = <pyplanet.core.events.callback.Callback ob**

Signal Player spawns.

Code `shootmania:royal_player_spawn`

Description Callback sent when a player is spawned.

Original Callback *Script* Shootmania.Royal.PlayerSpawn

Parameters **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.

`pyplanet.apps.core.shootmania.callbacks.royal.`**results = <pyplanet.core.events.callback.Callback object>**

Signal End of round with the winner of the Royal round.

Code `shootmania:royal_results`

Description Callback sent at the end of the round with the player instance of the winner.

Original Callback *Script* Shootmania.Royal.RoundWinner

Parameters **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance that won the round.

Trackmania

pyplanet.apps.core.trackmania.callbacks.**finish = <pyplanet.core.events.dispatcher.Signal object>**

Signal Player finishes a lap or the race.

Code `trackmania:finish`

Description Player finishes a lap or the complete race. Custom signal!.

Original Callback *None*

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **race_time** (*int*) – Time in milliseconds of the complete race.
- **lap_time** (*int*) – Time in milliseconds of the current lap.
- **cps** – Deprecated!
- **lap_cps** (*list*) – Current lap checkpoint times.
- **race_cps** (*list*) – Complete race checkpoint times.
- **flow** (*pyplanet.apps.core.maniaplanet.models.player.PlayerFlow*) – Flow instance.
- **is_end_race** (*bool*) – Is this the finish and end of race.
- **is_end_lap** (*bool*) – Is this the finish and end of current lap.
- **raw** – Prevent to use this!

pyplanet.apps.core.trackmania.callbacks.**give_up = <pyplanet.core.events.callback.Callback object>**

Signal Player gives up.

Code `trackmania:give_up`

Description Callback sent when a player gives up his current run/round.

Original Callback *Script* Trackmania.Event.GiveUp

Parameters

- **time** – Server time when callback has been sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **flow** (*pyplanet.apps.core.maniaplanet.models.player.PlayerFlow*) – Flow class instance.

pyplanet.apps.core.trackmania.callbacks.**respawn = <pyplanet.core.events.callback.Callback object>**

Signal Player respawn at cp.

Code `trackmania:respawn`

Description Callback sent when a player respawns at the last checkpoint/start.

Original Callback *Script* Trackmania.Event.Respawn

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **flow** (*pyplanet.apps.core.maniaplanet.models.player.PlayerFlow*) – Flow class instance.
- **race_cp** – Checkpoint times in current **race**.
- **lap_cp** – Checkpoint times in current **lap**.
- **race_time** – Total race time in milliseconds.
- **lap_time** – Current lap time in milliseconds.

pyplanet.apps.core.trackmania.callbacks.**scores = <pyplanet.core.events.callback.Callback object>**

Signal Score callback, called after the map. (Around the podium time).

Code `trackmania:scores`

Description Teams and players scores.

Original Callback *Script* Trackmania.Scores

Parameters

- **players** (`list`) – Player score payload. Including player instance etc.
- **teams** (`list`) – Team score payload.
- **winner_team** – The winning team.
- **use_teams** – Use teams.
- **winner_player** – The winning player.
- **section** – Section, current progress of match. Important to check before you save results!!

pyplanet.apps.core.trackmania.callbacks.**start_countdown = <pyplanet.core.events.callback.Callback object**

Signal Player starts his round, the countdown starts right now.

Code `trackmania:start_countdown`

Description Callback sent when a player see the 3,2,1,Go! countdown.

Original Callback *Script* Trackmania.Event.StartCountdown

Parameters

- **time** – Server time when callback has been sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **flow** (*pyplanet.apps.core.maniaplanet.models.player.PlayerFlow*) – Flow class instance.

pyplanet.apps.core.trackmania.callbacks.**start_line = <pyplanet.core.events.callback.Callback object>**

Signal Player drives off from the start line.

Code `trackmania:start_line`

Description Callback sent when a player starts to race (at the end of the 3,2,1,GO! sequence).

Original Callback *Script* Trackmania.Event.StartLine

Parameters

- **time** – Server time when callback has been sent.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **flow** (*pyplanet.apps.core.maniaplanet.models.player.PlayerFlow*) – Flow class instance.

pyplanet.apps.core.trackmania.callbacks.**stunt = <pyplanet.core.events.callback.Callback object>**

Signal Player did a stunt.

Code `trackmania:stunt`

Description Callback sent when a player did a stunt.

Original Callback *Script* Trackmania.Event.Stunt

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **race_time** – Total race time in milliseconds.
- **lap_time** – Current lap time in milliseconds.
- **stunt_score** – Current stunt score.
- **figure** – Figure of stunt.
- **angle** – Angle of stunt.
- **points** – Points got by figure.
- **combo** – Combo counter
- **is_straight** – Is the jump/stunt straight.
- **is_reverse** – Is jump/stunt reversed.
- **is_master_jump** – Is master jump.
- **factor** – Factor multiplier of points (figure).

pyplanet.apps.core.trackmania.callbacks.**warmup_end = <pyplanet.core.events.callback.Callback object>**

Signal Warmup Ends

Code `trackmania:warmup_end`

Description Callback sent when the warmup ends.

Original Callback *Script* Trackmania.WarmUp.End

pyplanet.apps.core.trackmania.callbacks.**warmup_end_round = <pyplanet.core.events.callback.Callback objec**

Signal Warmup Round Ends.

Code `trackmania:warmup_end_round`

Description Callback sent when a warm up round ends.

Original Callback *Script* Trackmania.WarmUp.EndRound

Parameters

- **current** – Current round number.
- **total** – Total warm up rounds.

pyplanet.apps.core.trackmania.callbacks.**warmup_start = <pyplanet.core.events.callback.Callback object>**

Signal Warmup Starts

Code `trackmania:warmup_start`

Description Callback sent when the warmup starts.

Original Callback *Script* Trackmania.WarmUp.Start

pyplanet.apps.core.trackmania.callbacks.**warmup_start_round = <pyplanet.core.events.callback.Callback ob**

Signal Warmup Round Starts.

Code `trackmania:warmup_start_round`

Description Callback sent when a warm up round start.

Original Callback *Script* Trackmania.WarmUp.StartRound

Parameters

- **current** – Current round number.
- **total** – Total warm up rounds.

pyplanet.apps.core.trackmania.callbacks.**warmup_status = <pyplanet.core.events.callback.Callback object>**

Signal Status of Trackmania warmup. (mostly as response).

Code `trackmania:warmup_status`

Description The status of Trackmania's the warmup.

Original Callback *Script* Trackmania.WarmUp.Status

Parameters

- **responseid** – Internally used. Ignore
- **available** (*bool*) – Is warmup available in the game mode. (Boolean).
- **active** (*bool*) – Is warmup active and ongoing right now.

pyplanet.apps.core.trackmania.callbacks.**waypoint = <pyplanet.core.events.callback.Callback object>**

Signal Player crosses a checkpoint.

Code `trackmania:waypoint`

Description Callback sent when a player crosses a checkpoint.

Original Callback *Script* Trackmania.Event.WayPoint

player=player, race_time=source['racetime'], flow=flow, raw=source

Parameters

- **race_time** – Total race time in milliseconds.
- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance
- **flow** (*pyplanet.apps.core.maniaplanet.models.player.PlayerFlow*) – Flow class instance.
- **raw** – Raw data, prevent to use this!

Note: This signal is not called when the player finishes or passes finish line during laps map.

CHAPTER 17

API Documentation

Modules:

pyplanet.apps

class pyplanet.apps.**Apps**(*instance*)

The apps class contains the context applications, loaded or not loaded in order of declaration or requirements if given by app configuration.

The apps should contain a configuration class that could be loaded for reading out metadata, options and other useful information such as description, author, version and more.

check()

Check and remove unsupported apps based on the current game and script mode.

discover()

The discover function will discover all models, signals and more from apps in the right order.

init()

populate(*apps*, *in_order=False*)

Loads application into the apps registry. Once you populate, the order isn't yet been decided. After all imports are done you should shuffle the apps list so it's in the right order of execution!

Parameters

- **apps** (list) – Apps list.
- **in_order** – Is the list already in order?

start()

class pyplanet.apps.**AppConfig**(*app_name*, *app_module*, *instance*)

This class is the base class for the Applications metadata class. The class holds information and hooks that will be executed after initiation for example.

```
class MyApp(AppConfig):

        async def on_start(self):
                print('we are staring!!')
```

app_dependencies = None

game_dependencies = None

human_name = None

static import_app(*entry*, *instance*)

is_game_supported(*game*)

is_mode_supported(*mode*)

label = None

mode_dependencies = None

name = None

on_destroy()
On destroy is being called when unloading the app from the memory.

on_init()
The on_init will be called before all apps are started (just before the on_ready). This will allow the app to register things like commands, permissions and other things that are important and don't require other apps to be ready.

on_start()
The on_start call is being called after all apps has been started successfully. You should register any stuff that is related to any other apps and signals like your *self* context for signals if they are classmethods.

on_stop()
The on_stop will be called before stopping the app.

path = None

ui
Deprecated since version 0.0.1: Use `context.ui` instead.

class `pyplanet.apps.config.`**_AppContext**(*app*)
The app context holds instances of core/contrib components that must be managed on a per app base. Such as the UI registration and distribution.

setting = None
Setting Contrib Component. See *Setting Classes*.

ui = None
UI Component. See *UI Classes*.

pyplanet.views

pyplanet.views

class pyplanet.views.base.**View**(*manager=None*, *id=None*, *version='3'*, *body=None*, *template=None*, *timeout=0*, *hide_click=False*, *data=None*, *player_data=None*, *disable_alt_menu=False*, *throw_exceptions=False*)

Base view. The base view will inherit from StaticManiaLink class.

destroy()

Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

destroy_sync()

Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

This method is sync and will call a async method (destroying of the manialink at our players) async but will not be executed at the same time. Be aware with this one!

display(*player_logins=None*, ***kwargs*)

Display the manialink. Will also render if no body is given. Will show per player or global. depending on the data given and stored!

Parameters **player_logins** – Only display to the list of player logins given.

handle_catch_all(*player*, *action*, *values*, ***kwargs*)

Override this class to handle all other actions related to this view/manialink.

Parameters

- **player** – Player instance.
- **action** – Action name/string
- **values** – Values provided by the user client.
- **kwargs** –

  –

hide(*player_logins=None*)

Hide manialink globally of only for the logins given in parameter.

Parameters **player_logins** – Only hide for list of players, None for all players on the server.

render(*player_login=None*, *data=None*, *player_data=None*, *template=None*)

Render template. Will render template and return body.

Parameters

- **player_login** – Render data only for player, set to None to globally render (and ignore player_data).
- **data** – Data to append.
- **player_data** – Data to append.
- **template** (pyplanet.core.ui.template.Template) – Template instance to use.

Returns Body, rendered manialink + script.

subscribe(*action*, *target*)
Subscribe to a action given by the manialink.

Parameters

- **action** – Action name.
- **target** – Target method.

Returns

class pyplanet.views.template.**TemplateView**(*manager=None*, *id=None*, *version='3'*, *body=None*, *template=None*, *timeout=0*, *hide_click=False*, *data=None*, *player_data=None*, *disable_alt_menu=False*, *throw_exceptions=False*)
The TemplateView will provide a view based on a XML template (ManiaLink for example). The view contains some class properties that are required to work. Those are described bellow.

To use the TemplateView. Initiate it in your own View class, and override one of the following methods:

Method get_context_data() Return the global context data here. Make sure you use the super() to retrieve the current context.

Method get_player_data() Retrieve the player specific dictionary. Return dict with player as key and value should contain the data dict.

Method get_template() Return the template instance from Jinja2. You mostly should not override this method.

As alternative you can manipulate the instance.data and instance.player_data too.

Properties that are useful to change:

Prop data Global context data. Dict.

Prop player_data Player context data. Dict with player as key.

Prop hide_click Should the manialink disappear after clicking a button/text.

Prop timeout Timeout to hide manialink in seconds.

Example usage:

```
class AlertView(TemplateView):
        template_name = 'my_app/test.xml' # template should be in: ./my_app/
↪templates/test.xml
        # Some prefixes that can be used in the template_name:
        #
        # - core.views: ``pyplanet.views.templates``.
        # - core.pyplanet: ``pyplanet.apps.core.pyplanet.templates``.
        # - core.maniaplanet: ``pyplanet.apps.core.pyplanet.templates``.
        # - core.trackmania: ``pyplanet.apps.core.trackmania.templates``.
        # - core.shootmania: ``pyplanet.apps.core.shootmania.templates``.
        # - [app_label]: ``[app path]/templates``.

        async def get_context_data(self):
                context = await super().get_context_data()
                context['title'] = 'Sample'
                return context
```

destroy()
Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

destroy_sync()
Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

This method is sync and will call a async method (destroying of the manialink at our players) async but will not be executed at the same time. Be aware with this one!

display(*player_logins=None*, ***kwargs*)
Display the manialink. Will also render if no body is given. Will show per player or global. depending on the data given and stored!

Parameters **player_logins** – Only display to the list of player logins given.

get_context_data()
Get global and local context data, used to render template.

get_player_data()
Get data per player, return dict with login => data dict.

handle_catch_all(*player*, *action*, *values*, ***kwargs*)
Override this class to handle all other actions related to this view/manialink.

Parameters

- **player** – Player instance.
- **action** – Action name/string
- **values** – Values provided by the user client.
- **kwargs** –

  –

hide(*player_logins=None*)
Hide manialink globally of only for the logins given in parameter.

Parameters **player_logins** – Only hide for list of players, None for all players on the server.

render(**args*, *player_login=None*, ***kwargs*)
Render template for player. This will only render the body and return it. Not send it!

Parameters **player_login** – Render data only for player, set to None to globally render (and ignore player_data).

Returns Body, rendered manialink + script.

subscribe(*action*, *target*)
Subscribe to a action given by the manialink.

Parameters

- **action** – Action name.
- **target** – Target method.

Returns

pyplanet.views.generics

class pyplanet.views.generics.alert.**AlertView**(*message*, *size='md'*, *buttons=None*, *manager=None*, *target=None*, ***data*)
The AlertView can be used to show several generic alerts to a player. You can use 3 different sizes, and adjust the message text.

The 3 sizes: sm, md and lg.

__init__(*message*, *size='md'*, *buttons=None*, *manager=None*, *target=None*, ***data*)
Create an AlertView instance.

Parameters

- **message** (*str*) – The message to display to the end-user, Use \n for new lines. You can use symbols from FontAwesome by using Unicode escaped strings.
- **size** (*str*) – Size to use, this parameter should be a string, and one of the following choices: 'sm', 'md' or 'lg. Defaults to 'md'.
- **buttons** (list) – Buttons to display, Should be an array with dictionary which contain: name.
- **manager** (*pyplanet.core.ui._BaseUIManager*) – UI Manager to use, You should always keep this undefined unless you know what your doing!
- **target** – Target coroutine method called as handle of button clicks.

close(*player*, ***kwargs*)
Close the alert.

class pyplanet.views.generics.alert.**PromptView**(*message*, *size='md'*, *buttons=None*, *manager=None*, *default=''*, *validator=None*)
The PromptView is like the AlertView but can ask for a text entry.

The 3 sizes: sm, md and lg.

You can listen for the results of the players input with the wait_for_input() async handler (future). Example:

```
prompt = PromptView('Please enter your name')
await prompt.display(['login'])

user_input = await prompt.wait_for_input()
print(user_input)
```

You can do validations before it's okay with giving a function to the argument validator. Example:

```
def my_validator(value):
        try:
                int(value)
                return True, None
        except:
                return False, 'Value should be an integer!'

prompt = PromptView('Please enter your name', validator=my_validator)
await prompt.display(['login'])

user_input = await prompt.wait_for_input()
print(user_input)
```

wait_for_input()
Wait for input and return it.

Returns Returns the string value of the user.

class pyplanet.views.generics.list.**ListView**(**args*, ***kwargs*)
The ListView is an abstract list that uses a database query to show and manipulate the list that is presented to the

end-user. The ListView is able to automatically manage the searching, ordering and pagination of your query contents.

The columns could be specified, for each column you can change behaviour, such as searchable and sortable. But also custom rendering of the values that will be displayed.

You can override `get_fields()`, `get_actions()`, `get_query()` if you need any customization or use a self method or variable in one of your properties.

Note: The design and some behaviour can change in updates of PyPlanet. We aim to provide backward compatibility as much as we can. If we are going to break things we will make it deprecated, or if we are in a situation of not having enough time to provide a transition time, we are going to create a separate solution (like a second version).

```
class SampleListView(ListView):
        query = Model.select()
        model = Model
        title = 'Select your item'
        fields = [
                {'name': 'Name', 'index': 'name', 'searching': True, 'sorting':␣
↪True},
                {'name': 'Author', 'index': 'author', 'searching': True, 'sorting
↪': True},
        ]
        actions = [
                {
                        'name': 'Delete',
                        'action': self.action_delete,
                        'style': 'Icons64x64_1',
                        'substyle': 'Close'
                },
        ]

        async def action_delete(self, player, values, instance, **kwargs):
                print('Delete value: {}'.format(instance))
```

close (*player*, **args*, ***kwargs*)
Close the link for a specific player. Will hide manialink and destroy data for player specific to save memory.

Parameters **player** (*pyplanet.apps.core.maniaplanet.models.Player*) – Player model instance.

display (*player=None*)
Display list to player.

Parameters **player** (*str, pyplanet.apps.core.maniaplanet.models.Player*) – Player login or model instance.

refresh (*player*, **args*, ***kwargs*)
Refresh list with current properties for a specific player. Can be used to show new data changes.

Parameters **player** (*pyplanet.apps.core.maniaplanet.models.Player*) – Player model instance.

single_list = True
Change this to False to have multiple lists open at the same time.

class pyplanet.views.generics.list.**ManualListView** (*data=None*, **args*, ***kwargs*)
The ManualListView will act as a ListView, but not based on a model or query.

get_data()
Override this method, return a list with dictionaries inside.

pyplanet.core.exceptions

exception pyplanet.core.exceptions.**AppRegistryNotReady**
The registry was not yet ready to invoke

exception pyplanet.core.exceptions.**ImproperlyConfigured**
The configuration is not given or is invalid.

exception pyplanet.core.exceptions.**InvalidAppModule**
The given app string is invalid or the app itself is misconfigured!

exception pyplanet.core.exceptions.**SignalException**
Signal receiver thrown an exception!

exception pyplanet.core.exceptions.**SignalGlueStop**
Throw this exception inside of your glue method to stop executing the signal.

exception pyplanet.core.exceptions.**TransportException**
The XML-RPC tunnel got a transport error.

pyplanet.core.instance

pyplanet.core.instance.**Controller = <pyplanet.core.controller._Controller object>**
Controller access point to prevent circular imports. This is a lazy provided way to get the instance from anywhere! :type Controller: pyplanet.core.Controller :type: pyplanet.core.Controller

class pyplanet.core.instance.**Instance**(*process_name*)
Controller Instance. The very base of the controller, containing class instances of all core components.

Variables

- **process_name** – Process and pool name.
- **loop** – AsyncIO Event Loop.
- **game** – Game Information class.
- ***apps*** – Apps component.
- **gbx** – Gbx component.
- **db** – Database component.
- ***storage*** – Storage component.
- **signal_manager** – Signal Manager.
- **ui_manager** – UI Manager (global). Please use the APP context UI manager instead!
- **map_manager** – Contrib: Map Manager.
- **player_manager** – Contrib: Player Manager.
- **permission_manager** – Contrib: Permission Manager.
- **command_manager** – Contrib: Command Manager.

- **setting_manager** – Contrib: Setting Manager. Please use the APP context setting manager instead!
- **mode_manager** – Contrib. Mode Manager.

performance_mode
Gives back a boolean, True if we are in performance mode.

Returns Performance mode boolean.

start(*run_forever=True*)
Start wrapper.

pyplanet.core.ui

class pyplanet.core.ui.template.**Template**(*file*)
Template class manages the template file source and the rendering of it.

Will also take care of the loader of the Jinja2 template engine.

Some notable prefixes:

•core.views: `pyplanet.views.templates`.

•core.pyplanet: `pyplanet.apps.core.pyplanet.templates`.

•core.maniaplanet: `pyplanet.apps.core.pyplanet.templates`.

•core.trackmania: `pyplanet.apps.core.trackmania.templates`.

•core.shootmania: `pyplanet.apps.core.shootmania.templates`.

•[app_label]: `[app path]/templates`.

The UI Properties will be set and hold in the class definition bellow.

class pyplanet.core.ui.ui_properties.**UIProperties**(*instance*)
Set the custom Script UI Properties.

Tip: Look at the possible UI Properties right here:

•Trackmania: https://github.com/maniaplanet/script-xmlrpc/blob/master/XmlRpcListing.md#trackmaniauisetproperties

•Shootmania: https://github.com/maniaplanet/script-xmlrpc/blob/master/XmlRpcListing.md#shootmaniauisetproperties

---

Access this class with:

```
self.instance.ui_manager.properties
```

get_attribute(*element: str*, *attribute: str*, *default=<object object>*)
Get an attribute value of an element.

Parameters

- **element** – Element name
- **attribute** – Attribute name
- **default** – Default if not found.

Returns Boolean if it's set correctly.

get_visibility(*element: str*, *default=<object object>*)
Set the visibility of the UI Property and don't complain about failing to set. Must be set at the start of the app(s).

Parameters

- **element** – Element name, such as notices, map_info and chat. Full list: https://github.com/maniaplanet/script-xmlrpc/blob/master/XmlRpcListing.md#shootmaniauisetproperties
- **default** – The default value, or an exception if not given.

Returns The boolean if it's visible or raise exception if not exists (or the default if default is given).

set_attribute(*element: str*, *attribute: str*, *value*)
Set an attribute of an element and silent if it's not found. Useful to change positions but unsure if it will and still exists. Returns boolean if it's set successfully.

Parameters

- **element** – Element name
- **attribute** – Attribute name
- **value** – New value of the attribute.

Returns Boolean if it's set correctly.

set_visibility(*element: str*, *visible: bool*)
Set the visibility of the UI Property and don't complain about failing to set. Must be set at the start of the app(s).

Parameters

- **element** – Element name, such as notices, map_info and chat. Full list: https://github.com/maniaplanet/script-xmlrpc/blob/master/XmlRpcListing.md#shootmaniauisetproperties
- **visible** – Boolean if the element should be visible.

Returns Boolean, true if is set, false if failed to set.

class pyplanet.core.ui.components.**StaticManiaLink**(*manager=None*, *id=None*, *version='3'*, *body=None*, *template=None*, *timeout=0*, *hide_click=False*, *data=None*, *player_data=None*, *disable_alt_menu=False*, *throw_exceptions=False*)
The StaticManiaLink is mostly used in PyPlanet for general views. Please use the `View` classes instead of this core ui component!

destroy()
Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

destroy_sync()
Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

This method is sync and will call a async method (destroying of the manialink at our players) async but will not be executed at the same time. Be aware with this one!

display(*player_logins=None*, ***kwargs*)
Display the manialink. Will also render if no body is given. Will show per player or global. depending on the data given and stored!

Parameters **player_logins** – Only display to the list of player logins given.

handle_catch_all(*player*, *action*, *values*, ***kwargs*)
Override this class to handle all other actions related to this view/manialink.

Parameters

- **player** – Player instance.
- **action** – Action name/string
- **values** – Values provided by the user client.
- **kwargs** –

  –

hide(*player_logins=None*)
Hide manialink globally of only for the logins given in parameter.

Parameters **player_logins** – Only hide for list of players, None for all players on the server.

render(*player_login=None*, *data=None*, *player_data=None*, *template=None*)
Render template. Will render template and return body.

Parameters

- **player_login** – Render data only for player, set to None to globally render (and ignore player_data).
- **data** – Data to append.
- **player_data** – Data to append.
- **template** (pyplanet.core.ui.template.Template) – Template instance to use.

Returns Body, rendered manialink + script.

subscribe(*action*, *target*)
Subscribe to a action given by the manialink.

Parameters

- **action** – Action name.
- **target** – Target method.

Returns

class pyplanet.core.ui.components.**DynamicManiaLink**(*id*)
The DynamicManiaLink is a special manialink with data-bindings and automatically updates via maniascript. Please use the View classes instead!

> **Warning:** This feature is not yet implemented.

destroy()
Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

destroy_sync()
Destroy the Manialink with it's handlers and references. Will also hide the Manialink for all users!

This method is sync and will call a async method (destroying of the manialink at our players) async but will not be executed at the same time. Be aware with this one!

display(*player_logins=None*, ***kwargs*)
Display the manialink. Will also render if no body is given. Will show per player or global. depending on the data given and stored!

Parameters **player_logins** – Only display to the list of player logins given.

handle_catch_all(*player*, *action*, *values*, ***kwargs*)
Override this class to handle all other actions related to this view/manialink.

Parameters

- **player** – Player instance.
- **action** – Action name/string
- **values** – Values provided by the user client.
- **kwargs** –

  –

hide(*player_logins=None*)
Hide manialink globally of only for the logins given in parameter.

Parameters **player_logins** – Only hide for list of players, None for all players on the server.

render(*player_login=None*, *data=None*, *player_data=None*, *template=None*)
Render template. Will render template and return body.

Parameters

- **player_login** – Render data only for player, set to None to globally render (and ignore player_data).
- **data** – Data to append.
- **player_data** – Data to append.
- **template** (pyplanet.core.ui.template.Template) – Template instance to use.

Returns Body, rendered manialink + script.

subscribe(*action*, *target*)
Subscribe to a action given by the manialink.

Parameters

- **action** – Action name.
- **target** – Target method.

Returns

exception pyplanet.core.ui.exceptions.**ManialinkMemoryLeakException**
Is thrown when a memory leak is detected in a view. Raised when a manialink responds to a view, but the view is vanished for the specified player(s).

exception pyplanet.core.ui.exceptions.**UIException**
Base exception for UI core component.

exception pyplanet.core.ui.exceptions.**UIPropertyDoesNotExist**
Thrown when UI Property with element doesn't exist.

class pyplanet.core.ui.loader.**PyPlanetLoader**
Lazy loader for the pyplanet jinja2 loader.

pyplanet.core.storage

exception pyplanet.core.storage.exceptions.**StorageException**
Base storage exception.

class pyplanet.core.storage.storage.**Storage**(*instance*, *driver: pyplanet.core.storage.interface.StorageDriver*, *config*)
The storage component manager is managing the storage access trough drivers that can be customized.

> **Warning:** Some drivers are work in progress!

driver
Get the raw driver. Be careful with this!

Returns Driver Instance

Return type pyplanet.core.storage.interface.StorageDriver

open(*file: str*, *mode: str = 'rb'*, ***kwargs*)
Open a file on the server. Use relative path to the dedicated root. Use the other open methods to relative from another base path.

Parameters

- **file** – Filename/path, relative to the dedicated root path.
- **mode** – Mode to open, see the python *open* manual for supported modes.

Returns File handler.

open_map(*file: str*, *mode: str = 'rb'*, ***kwargs*)
Open a file on the server. Relative to the Maps folder (UserData/Maps).

Parameters

- **file** – Filename/path, relative to the dedicated maps folder.
- **mode** – Mode to open, see the python *open* manual for supported modes.

Returns File handler.

open_match_settings(*file: str*, *mode: str = 'r'*, ***kwargs*)
Open a file on the server. Relative to the MatchSettings folder (UserData/Maps/MatchSettings).

Parameters

- **file** – Filename/path, relative to the dedicated matchsettings folder.
- **mode** – Mode to open, see the python *open* manual for supported modes.

Returns File handler.

remove_map(*file: str*)
Remove a map file with filename given.

Parameters **file** – Filename, relative to Maps folder.

pyplanet.core.storage.drivers

class pyplanet.core.storage.drivers.local.**LocalDriver**(*instance*, *config: dict = None*)
Local storage driver is using the Python build-in file access utilities for accessing a local storage-like system.

Option BASE_PATH Override the maniaplanet given base path.

class pyplanet.core.storage.drivers.asyncssh.**SFTPDriver**(*instance*, *config: dict = None*)
SFTP storage driver is using the asyncssh module to access storage that is situated remotely.

> **Warning:** This driver is not ready for production use!!

Option HOST Hostname of destinotion server.

Option PORT Port destinotion server.

Option USERNAME Username of the user account.

Option PASSWORD Password of the user account. (optional if you use public/private keys).

Option KNOWN_HOSTS File to the Known Hosts file.

Option CLIENT_KEYS Array with client private keys.

Option PASSPHRASE Passphrase to unlock private key(s).

Option KWARGS Any other options that will be passed to asyncssh.

connect_sftp()
Get sftp client.

Returns Sftp client.

Return type asyncssh.SFTPClient

pyplanet.core.events

The events manager contains the class that manages custom and abstract callbacks into the system callbacks. Once a signals is registered here it could be used by string reference. This makes it easy to have dynamically signals being created by other apps in a single place so it could be used over all apps.

For example you would create your own custom signal if you have a app for your own created game mode script that abstracts all the raw XML-RPC events into nice structured and maybe even including fetched data from external sources.

class pyplanet.core.events.manager.**_SignalManager**
Signal Manager class.

> **Note:** Access this via instance.signal_manager.

finish_reservations()
The method will copy all reservations to the actual signals. (PRIVATE)

finish_start(**args*, ***kwargs*)
Finish startup the core, this will copy reservations. (PRIVATE).

get_callback(*call_name*)
Get signal by XML-RPC (script) callback.

Parameters **call_name** – Callback name.

Returns Signal class or nothing.

Return type pyplanet.core.events.Signal

get_signal(*key*)
Get signal by key (namespace:code).

Parameters **key** – namespace:code key.

Returns signal or none

Return type pyplanet.core.events.Signal

init_app(*app*)
Initiate app, load all signal/callbacks files. (just import, they should register with decorators).

Parameters **app** (pyplanet.apps.AppConfig) – App instance

listen(*signal*, *target*, *conditions=None*, ***kwargs*)
Register a listing client to the signal given (signal instance or string).

Parameters

- **signal** – Signal instance or string: "namespace:code"
- **target** – Target method to call.
- **conditions** – Reserved for future purposes.

register_signal(*signal*, *app=None*, *callback=False*)
Register a signal to be known in the signalling system.

Parameters

- **signal** – Signal(s)
- **app** – App context/instance.
- **callback** – Will a callback handle the response (mostly raw callbacks).

pyplanet.core.events.callback

This file contains a glue between core callbacks and desired callbacks.

class pyplanet.core.events.callback.**Callback**(*call*, *namespace*, *code*, *target=None*)
A callback signal is an double signal. Once for the GBX Callback itself (the Gbx callback named). And the destination Between those two signals is a sort of *processor* that confirms it into the PyPlanet style objects.

For example, a player connect will result in a player database object instead of the plain Maniaplanet payload. This will make it possible to develop your app as fast as possible, without any overhead and make it better with callback payload changes!

glue(*signal*, *source*, ***kwargs*)
The glue method converts the source signal (gbx callback) into the pyplanet signal.

pyplanet.core.events.callback.**handle_generic**(*source*, *signal*, ***kwargs*)
The handle_generic is a simple handle (*processing glue*) for just forwarding the payload from the maniaplanet server into the signal payload.

pyplanet.core.events.dispatcher

This file has been forked from Django and PyDispatcher. The PyDispatcher is licensed under BSD.

class pyplanet.core.events.dispatcher.**Signal**(*code=None*, *namespace=None*, *process_target=None*, *use_caching=False*)
A signal is a destination tho distribute to where multiple listeners get the message. (event distribution).

class Meta
The meta-class contains the code of the signal, used for string notation. An optional namespace could be given to override the app label namespace.

> **Warning:** Only change or access this if you override the `Signal` class in your own class.

has_listeners()
Has the signal listeners.

Returns

process(***data*)
This method processed data into abstract data. You can give your own function in the init of the Signal or override the method.

Parameters **data** – Raw data input

Returns Parsed data output

register(*receiver*, *weak=True*, *dispatch_uid=None*)
Connect receiver to sender for signal.

Parameters

- **receiver** – A function or an instance method which is to receive signals. Receivers must be hashable objects. If weak is True, then receiver must be weak referenceable.Receivers must be able to accept keyword arguments. If a receiver is connected with a dispatch_uid argument, it will not be added if another receiver was already connected with that dispatch_uid.
- **weak** – Whether to use weak references to the receiver. By default, the module will attempt to use weak references to the receiver objects. If this parameter is false, then strong references will be used.
- **dispatch_uid** – An identifier used to uniquely identify a particular instance of a receiver. This will usually be a string, though it may be anything hashable.

send(*source*, *raw=False*, *catch_exceptions=False*, *gather=True*)
Send signal with source. If any receiver raises an error, the error propagates back through send, terminating the dispatch loop. So it's possible that all receivers won't be called if an error is raised.

Parameters

- **source** – The data to be send to the processor which produces data that will be send to the receivers.

- **raw** – Optional bool parameter to just send the source to the receivers without any processing.
- **catch_exceptions** – Catch and return the exceptions.
- **gather** – Execute multiple receivers at the same time (parallel). On by default!

Returns Return a list of tuple pairs [(receiver, response), ...].

send_robust (*source=None*, *raw=False*, *gather=True*)
Send signal from sender to all connected receivers catching errors.

Parameters

- **source** – The data to be send to the processor which produces data that will be send to the receivers.
- **raw** – Optional bool parameter to just send the source to the receivers without any processing.
- **gather** – Execute multiple receivers at the same time (parallel). On by default!

Returns Return a list of tuple pairs [(receiver, response), ...]. If any receiver raises an error (specifically any subclass of Exception), return the error instance as the result for that receiver.

set_self (*receiver*, *slf*)
Set the self instance on a receiver.

Deprecated since version 0.0.1.

Parameters

- **receiver** – Receiver function.
- **slf** – Self instance

unregister (*receiver=None*, *dispatch_uid=None*)
Disconnect receiver from sender for signal. If weak references are used, disconnect need not be called. The receiver will be removed from dispatch automatically.

Parameters

- **receiver** – The registered receiver to disconnect. May be none if dispatch_uid is specified.
- **dispatch_uid** – the unique identifier of the receiver to disconnect

pyplanet.god

Error: This package is strictly private and should not be changed inside of one of your apps/customizations!

class pyplanet.god.pool.**EnvironmentPool** (*pool_names*, *max_restarts=0*)
This class manages the pool instances for the current environment/installation.

Warning: You should not have to use this class in any moment!

populate()
Populate the pool instance processes, (prepares the processes).

restart(*name=None*)
Restart single process, or all if no name is given.

Parameters **name** – Name or none for all pools.

shutdown()
Shutdown all processes.

start()
Start all processes.

watchdog()
Watch all the processes. (Blocking method!).

class pyplanet.god.process.**InstanceProcess**(*queue*, *environment_name='default'*, *pool=None*)
The InstanceProcess is the encapsulation around the real controller instance.

> **Warning:** This code is still being executed at the main process!!

did_die
Boolean determinating if the process did die.

exitcode
Exit code of process.

Returns Exit code.

is_alive()
Call process method is_alive()

shutdown()
Shutdown (terminate) process.

start()
Start the process.

will_restart
Boolean: Is the process able to restart (not reached max_restarts).

pyplanet.contrib.map

The map contrib will provide map list and information to the apps and core.

class pyplanet.contrib.map.**MapManager**(*instance*)
Map Manager. Manages the current map pool and the current and next map.

Todo

Write introduction.

Warning: Don't initiate this class yourself.

add_map(*filename*, *insert=True*)
Add or insert map to current online playlist.

Parameters

- **filename** (*str*) – Load from filename relative to the 'Maps' directory on the dedicated host server.
- **insert** (*bool*) – Insert after the current map, this will make it play directly after the current map. True by default.

Raise pyplanet.contrib.map.exceptions.MapIncompatible

Raise pyplanet.contrib.map.exceptions.MapException

current_map
The current map, database model instance.

Return type pyplanet.apps.core.maniaplanet.models.Map

get_map(*uid=None*)
Get map instance by uid.

Parameters **uid** – By uid (pk).

Returns Player or exception if not found

get_map_by_index(*index*)
Get map instance by index id (primary key).

Parameters **index** – Primary key index.

Returns Map instance or raise exception.

handle_map_change(*info*)
This will be called from the glue that creates the signal 'maniaplanet:map_begin' or 'map_end'.

Parameters **info** – Mapinfo in dict.

Returns Map instance.

Return type pyplanet.apps.core.maniaplanet.models.map.Map

maps
Get the maps that are currently loaded on the server. The list should contain model instances of the currently loaded matchsettings. This list should be up-to-date.

Return type *list*

next_map
The next scheduled map.

Return type pyplanet.apps.core.maniaplanet.models.Map

playlist_has_map(*uid*)
Check if our current playlist has a map with the UID given.

Parameters **uid** – UID String

Returns Boolean, True if it's in our current playlist (match settings in our session).

previous_map
The previously played map, or None if not known!

Return type pyplanet.apps.core.maniaplanet.models.Map

remove_map(*map*, *delete_file=False*)
Remove and optionally delete file from server and playlist.

Parameters

- **map** – Map instance or filename in string.
- **delete_file** (*bool*) – Boolean to decide if we are going to remove the file from the server too. Defaults to False.

Raise pyplanet.contrib.map.exceptions.MapException

Raise pyplanet.core.storage.exceptions.StorageException

save_matchsettings(*filename=None*)
Save the current playlist and configuration to the matchsettings file.

Parameters **filename** (*str*) – Give the filename of the matchsettings, Leave empty to use the current loaded and configured one.

Raise pyplanet.contrib.map.exceptions.MapException

Raise pyplanet.core.storage.exceptions.StorageException

set_current_map(*map*)
Set the current map and jump to it.

Parameters **map** – Map instance or uid.

set_next_map(*map*)
Set the next map. This will prepare the manager to set the next map and will communicate the next map to the dedicated server.

The Map parameter can be a map instance or the UID of the map.

Parameters **map** (*pyplanet.apps.core.maniaplanet.models.Map, str*) – Map instance or UID string.

upload_map(*fh*, *filename*, *insert=True*, *overwrite=False*)
Upload and add/insert the map to the current online playlist.

Parameters

- **fh** – File handler, bytesio object or any readable context.
- **filename** (*str*) – The filename when saving on the server. Must include the map.gbx! Relative to 'Maps' folder.
- **insert** (*bool*) – Insert after the current map, this will make it play directly after the current map. True by default.
- **overwrite** (*bool*) – Overwrite current file if exists? Default False.

Raise pyplanet.contrib.map.exceptions.MapIncompatible

Raise pyplanet.contrib.map.exceptions.MapException

Raise pyplanet.core.storage.exceptions.StorageException

exception pyplanet.contrib.map.exceptions.**MapException**
Generic map exception by manager.

exception pyplanet.contrib.map.exceptions.**MapIncompatible**
The map you want to add/insert/upload is invalid and not suited for the current server config.

exception pyplanet.contrib.map.exceptions.**MapNotFound**
Map not found

pyplanet.contrib.player

The player contrib will provide player list and information to the apps and core.

class pyplanet.contrib.player.**PlayerManager**(*instance*)
Player Manager.

Todo

Write introduction.

Warning: Don't initiate this class yourself.

get_player(*login=None*, *pk=None*, *lock=True*)
Get player by login or primary key.

Parameters

- **login** – Login.
- **pk** – Primary Key identifier.
- **lock** – Lock for a sec when receiving.

Returns Player or exception if not found

Return type pyplanet.apps.core.maniaplanet.models.Player

handle_connect(*login*)
Handle a connection of a player, this call is being called inside of the Glue of the callbacks.

Parameters **login** – Login, received from dedicated.

Returns Database Player instance.

Return type pyplanet.apps.core.maniaplanet.models.Player

handle_disconnect(*login*)
Handle a disconnection of a player, this call is being called inside of the Glue of the callbacks.

Parameters **login** – Login, received from dedicated.

Returns Database Player instance.

Return type pyplanet.apps.core.maniaplanet.models.Player

on_start()
Handle startup, just before the apps will start. We will throw connects for the players so we know that the current playing players are also initiated correctly!

online
Online player list.

Exceptions for Map Manager.

exception pyplanet.contrib.player.exceptions.**PlayerNotFound**
Player not found

pyplanet.contrib.command

The commands contributed package contains command management and callback logic.

class pyplanet.contrib.command.**CommandManager**(*instance*)
The Command Manager contributed extension is a manager that controls all chat-commands in the game. Your app needs to use this manager to register any custom commands you want to provide.

Todo

Write introduction.

> **Warning:** Don't initiate this class yourself.

register(**commands*)
Register your command.

Parameters **commands** (*pyplanet.contrib.command.command.Command*) – Command instance.

class pyplanet.contrib.command.**Command**(*command*, *target*, *aliases=None*, *admin=False*, *namespace=None*, *parser=None*, *perms=None*, *description=None*)
The command instance describes the command itself, the target to fire and all other related information, like admin command or aliases.

Todo

Write introduction + examples

---

add_param(*name: str*, *nargs=1*, *type=<class 'str'>*, *default=None*, *required: bool = True*, *help: str = None*, *dest: str = None*)
Add positional parameter.

Parameters

- **name** – Name of parameter, will be used to store result into!
- **nargs** – Number of arguments, use integer or '*' for multiple or infinite.
- **type** – Type of value, keep str to match all types. Use any other to try to parse to the type.
- **default** – Default value when no value is given.
- **required** – Set the parameter required state, defaults to true.
- **help** – Help text to display when parameter is invalid or not given and required.
- **dest** – Destination to save into namespace result (defaults to name).

Returns parser instance

Return type pyplanet.contrib.command.command.Command

get_params(*input*)
Get params in array from input in array.

Parameters **input** (list) – Array of raw input.

Returns Array of parameters, stripped of the command name and namespace, if defined.

Return type *list*

handle(*instance*, *player*, *argv*)
Handle command parsing and execution.

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player object.
- **argv** – Arguments in array

match(*raw*)
Try to match the command with the given input in array style (splitted by spaces).

Parameters **raw** (list) – Raw input, split by spaces.

Returns Boolean if command matches.

usage_text
The usage text line for the command.

class pyplanet.contrib.command.**ParameterParser**(*prog=None*)
Parameter Parser.

Todo

Write introduction + examples.

---

add_param(*name: str*, *nargs=1*, *type=<class 'str'>*, *default=None*, *required: bool = True*, *help: str = None*, *dest: str = None*)
Add positional parameter.

Parameters

- **name** – Name of parameter, will be used to store result into!
- **nargs** – Number of arguments, use integer or '*' for multiple or infinite.
- **type** – Type of value, keep str to match all types. Use any other to try to parse to the type.
- **default** – Default value when no value is given.
- **required** – Set the parameter required state, defaults to true.
- **help** – Help text to display when parameter is invalid or not given and required.
- **dest** – Destination to save into namespace result (defaults to name).

Returns parser instance

Return type *pyplanet.contrib.command.ParameterParser*

errors
Get errors.

Returns array of strings.

Return type *list*

is_valid()
Is data valid?

Returns boolean

parse(*argv*)
Parse arguments.

Parameters **argv** – arguments.

parse_parameter(*param*, *input*, *position*)
Validate and parse param value at input position.

Parameters

- **param** (*dict*) – Param dict given.
- **input** (list) – Full params input (without command or namespace!)
- **position** (*int*) – Current seek position.

Returns value.

exception pyplanet.contrib.command.exceptions.**InvalidParamException**
Invalid parameter arguments given!

exception pyplanet.contrib.command.exceptions.**NotValidated**
Your parser hasn't been called with .parse() before, so no parsing took place!

pyplanet.contrib.permission

The permission contrib will provide permission abilities to players and admin levels.

class pyplanet.contrib.permission.**PermissionManager**(*instance*)
Permission Manager manges the permissions of all apps and players.

Todo

Write introduction.

Warning: Don't initiate this class yourself.

get_perm(*namespace*, *name*)
Get permission by namespace and name.

Parameters

- **namespace** (*str*) – Namespace of the permission
- **name** (*str*) – Name of the permission.

has_permission(*player*, *permission*)
Check if the player has the right permission.

Parameters

- **player** – player instance.
- **permission** – permission name.

Returns boolean if player is allowed.

on_start()
Handle startup, just before the apps will start. We will make sure we are ready to get requests for permissions.

register(*name*, *description=''*, *app=None*, *min_level=1*, *namespace=None*)
Register a new permission.

Parameters

- **name** – Name of permission
- **description** – Description in english.
- **app** – App instance to retrieve the label.
- **min_level** – Minimum level required.
- **namespace** – Namespace, only for core usage!

Returns Permission instance.

pyplanet.contrib.setting

class pyplanet.contrib.setting.manager.**AppSettingManager**(*instance*, *app*)
The local app setting manager is the one you should use to register settings to inside of your app.

You can use this manager like this:

```
from pyplanet.contrib.setting import Setting

async def on_start(self):
        await self.context.setting.register(
                Setting('feature_a', 'Enable feature A', Setting.CAT_FEATURES,␣
↪type=bool, description='Enable feature A'),
                Setting('feature_b', 'Enable feature B', Setting.CAT_FEATURES,␣
↪type=bool, description='Enable feature B'),
        )
```

For more information about the settings, categories, types, and all other options. Look at the `Settings` documentation.

Warning: Don't initiate this class yourself.

get_all(*prefetch_values=True*)
Retrieve a list of settings, with prefetched values, so get_value is almost instant (or use ._value, not recommended).

Parameters **prefetch_values** – Prefetch the values in this call. Defaults to True.

Returns List with setting objects.

get_categories()
Get all the categories we have registered. Returns a dict with label as key, and count + name as values.

get_setting(*key*, *prefetch_values=True*)
Get setting by key and optionally fetch the value if not yet fetched.

Parameters

- **key** – Key string
- **prefetch_values** – Prefetch the values if not yet fetched?

Returns Setting instance.

Raise SettingException

register(**settings*)
Register your setting(s). This will create default values when the setting has not yet been inited before.

Parameters **settings** (*pyplanet.contrib.setting.setting._Setting*) – Setting(s) given.

class pyplanet.contrib.setting.manager.**GlobalSettingManager**(*instance*)
Global Setting manager is available at the instance. `instance.setting_manager`.

> **Warning:** Don't use the setting_manager for registering app settings! Use the app setting manager instead!
>
> Don't initiate this class yourself.

create_app_manager(*app_config*)
Create app setting manager.

Parameters **app_config** (*pyplanet.apps.config.AppConfig*) – App Config instance.

Returns Setting Manager

Return type *pyplanet.contrib.setting.manager.AppSettingManager*

get_all(*prefetch_values=True*)
Retrieve a list of settings, with prefetched values, so get_value is almost instant (or use ._value, not recommended).

Parameters **prefetch_values** – Prefetch the values in this call. Defaults to True.

Returns List with setting objects.

get_app_manager(*app*)
Get the app manager for a specified app label or config instance.

Parameters **app** – App label in string or the app config instance.

Returns App manager instance.

Return type *pyplanet.contrib.setting.manager.AppSettingManager*

get_apps(*prefetch_values=True*)
Get all the app label + names for all the settings we can find in our registry. Returns a dict with label as key, and count + name as values.

Parameters **prefetch_values** – Prefetch the values in this call. Defaults to True.

Returns List with setting objects.

get_categories(*prefetch_values=True*)
Get all the categories we have registered. Returns a dict with label as key, and count + name as values.

Parameters **prefetch_values** – Prefetch the values in this call. Defaults to True.

Returns List with setting objects.

get_setting(*app_label*, *key*, *prefetch_values=True*)
Get setting by key and optionally fetch the value if not yet fetched.

Parameters

- **app_label** – Namespace (mostly app label).
- **key** – Key string
- **prefetch_values** – Prefetch the values if not yet fetched?

Returns Setting instance.

Raise SettingException

recursive_settings
Retrieve all settings, of all submanagers.

The setting contrib contains code for managing and providing settings contexts.

class pyplanet.contrib.setting.**Setting**(*key: str*, *name: str*, *category: str*, *type=<class 'str'>*, *description: str = None*, *choices=None*, *default=None*, *change_target=None*)
The setting class is for defining a setting for the end-user. This setting can be changed with /settings and //settings.

With this class you can define or manage your setting that is going to be public for all other apps and end-user.

You can get notified of changes with the `change_target` in the init of this class. Point this to a method (async or sync) with the following params: `old_value` and `new_value`.

Example:

```
my_setting = Setting(
        'dedimania_code', 'Dedimania Server Code', Setting.CAT_KEYS, type=str,
        description='The secret dedimania code. Get one at $lhttp://dedimania.net/
↪tm2stats/?do=register',
        default=None
)

my_other_setting = Setting(
        'sample_boolean', 'Booleans for the win!', Setting.CAT_BEHAVIOUR,␣
↪type=bool, description='Example',
)
```

__init__(*key: str*, *name: str*, *category: str*, *type=<class 'str'>*, *description: str = None*, *choices=None*, *default=None*, *change_target=None*)
Create setting with properties.

Parameters

- **key** – Key of setting, this is mainly only used for the backend and for referencing the setting. You should keep this unique in your app!
- **name** – Name of the setting that will be displayed as a small label to the player.
- **category** – Category from Categories.*. Must be provided!
- **type** – Type of value to expect, use python types here. str by default.
- **description** – Description to provide help and instructions to the player.
- **choices** – List or tuple with choices, only when wanting to restrict values to selected options.

- **default** – Default value if not provided from database. This will be returned. Defaults to None.
- **change_target** – Target method to call when the setting value has been changed.

__weakref__
list of weak references to the object (if defined)

clear()
Clear the value in the data storage. This will set the value to None, and will return the default value on request of data.

Raise NotFound / SerializationException

get_model()
Get the model for the setting. This will return the model instance or raise an exception when not found.

Returns Model instance

Raise NotFound

get_value(*refresh=False*)
Get the value or the default value for the setting model.

Parameters **refresh** – Force a refresh of the value.

Returns Value in the desired type and unserialized from database/storage.

Raise NotFound / SerializationException

initiate_setting()
Initiate database record for setting.

serialize_value(*value*)
Serialize the python value to the data store value, based on the type of the setting.

Parameters **value** – Python Value.

Returns Database Value

set_value(*value*)
Set the value, this will serialize and save the setting to the data storage.

Parameters **value** – Python value input.

Raise NotFound / SerializationException

type_name
Get the name of the specified type in string format, suited for displaying to end-user.

Returns User friendly name of type.

unserialize_value(*value*)
Unserialize the datastorage value to the python value, based on the type of the setting.

Parameters **value** – Value from database.

Returns Python value.

Raises ***pyplanet.contrib.setting.exceptions.SerializationException*** – SerializationException

class pyplanet.contrib.setting.**GlobalSettingManager**(*instance*)
Global Setting manager is available at the instance. `instance.setting_manager`.

Warning: Don't use the setting_manager for registering app settings! Use the app setting manager instead!

Don't initiate this class yourself.

create_app_manager(*app_config*)
Create app setting manager.

Parameters **app_config** (*pyplanet.apps.config.AppConfig*) – App Config instance.

Returns Setting Manager

Return type *pyplanet.contrib.setting.manager.AppSettingManager*

get_all(*prefetch_values=True*)
Retrieve a list of settings, with prefetched values, so get_value is almost instant (or use ._value, not recommended).

Parameters **prefetch_values** – Prefetch the values in this call. Defaults to True.

Returns List with setting objects.

get_app_manager(*app*)
Get the app manager for a specified app label or config instance.

Parameters **app** – App label in string or the app config instance.

Returns App manager instance.

Return type *pyplanet.contrib.setting.manager.AppSettingManager*

get_apps(*prefetch_values=True*)
Get all the app label + names for all the settings we can find in our registry. Returns a dict with label as key, and count + name as values.

Parameters **prefetch_values** – Prefetch the values in this call. Defaults to True.

Returns List with setting objects.

get_categories(*prefetch_values=True*)
Get all the categories we have registered. Returns a dict with label as key, and count + name as values.

Parameters **prefetch_values** – Prefetch the values in this call. Defaults to True.

Returns List with setting objects.

get_setting(*app_label*, *key*, *prefetch_values=True*)
Get setting by key and optionally fetch the value if not yet fetched.

Parameters

- **app_label** – Namespace (mostly app label).
- **key** – Key string
- **prefetch_values** – Prefetch the values if not yet fetched?

Returns Setting instance.

Raise SettingException

recursive_settings
Retrieve all settings, of all submanagers.

Exceptions for Setting Manager.

exception pyplanet.contrib.setting.exceptions.**SerializationException**
Setting value (un)serialization problems

exception pyplanet.contrib.setting.exceptions.**SettingException**
Abstract setting exception.

exception pyplanet.contrib.setting.exceptions.**TypeUnknownException**
The type is unknown.

pyplanet.contrib.mode

Mode contrib is managing mode settings and ui settings for the script mode.

class pyplanet.contrib.mode.**ModeManager**(*instance*)
Mode Manager manges the script, script settings and the mode UI settings of the current game mode.

> **Warning:** Don't initiate this class yourself. Use `instance.mode_manager` for an static instance.

get_current_script(*refresh=False*)
Get the current script name.

Parameters **refresh** – Refresh from server.

get_current_script_info()
Get the script info as a structure containing: Name, CompatibleTypes, Description, Version and the settings available.

get_next_script(*refresh=False*)
Get the next script name.

Parameters **refresh** – Refresh from server.

get_settings()
Get the current mode settings as a dictionary.

get_variables()
Get the mode script variables.

on_start()
Handle startup, just before the apps will start. We will make sure we are ready to get requests for permissions.

set_next_script(*name*)
Set the next played script name (after map restart/skip).

Parameters **name** – Name

update_next_settings(*update_dict*)
Queue setting changes for the next script (that will be active after restart).

Parameters **update_dict** – The dictionary with the partial updated keys and values.

update_next_variables(*update_dict*)
Queue variable changes for the next script (that will be active after restart).

Parameters **update_dict** – The dictionary with the partial updated keys and values.

update_settings(*update_dict*)
Update the current settings, merges current settings with the provided settings. Replaces by the keys you give if the data already exists.

Parameters **update_dict** – The dictionary with the partial updated keys and values.

update_variables(*update_dict*)
Update the current variables, merges current vars with the provided vars. Replaces by the keys you give if the data already exists.

Parameters **update_dict** – The dictionary with the partial updated keys and values.

Signals

This file contains the contrib mode signals, related to the current script/mode.

pyplanet.contrib.mode.signals.**script_mode_changed = <pyplanet.core.events.dispatcher.Signal object>**
Is called after a new script has been loaded and became active!. Reporting two parameters:

Parameters

- **unloaded_script** – Old script name.
- **loaded_script** – New and just loaded script.

pyplanet.contrib.converter

Converter contrib is managing migrating from another controller.

class pyplanet.contrib.converter.base.**BaseConverter**(*instance*, *db_type*, *db_host*, *db_name*, *db_user=None*, *db_password=None*, *db_port=None*, *prefix=None*, *charset='utf8'*)
Base Converter is the abstract converter class.

Please take a look at the other classes bellow.

class pyplanet.contrib.converter.xaseco2.**Xaseco2Converter**(**args*, ***kwargs*)
The XAseco2 Converter uses MySQL to convert the data to the new PyPlanet structure.

Please take a look at *Migrating from other controllers* pages for information on how to use this.

pyplanet.contrib.chat

Sending chat messages

We implemented an abstraction that will provide auto multicall and auto prefixing for you. You can use the following statements for example:

```
# Send chat message to all players.
await self.instance.chat('Test')

# Send chat message to specific player or multiple players.
await self.instance.chat('Test', 'player_login')  # Sends to single player.
await self.instance.chat('Test', 'player_login', player_instance)  # Sends to both␣
↪players.
```

```
# Execute in chain (Multicall).
await self.instance.chat.execute(
  'global_message',
  self.instance.chat('Test', 'player_login'),
  self.instance.chat('Test2', 'player_login2'),
)

# You can combine this with other calls in a GBX multicall:
await self.instance.gbx.multicall(
  self.instance.gbx.prepare('SetServerName', 'Test'),
  self.instance.chat('Test2', 'player_login2'),
)
```

API Documentation

The chat contrib makes it possible to send chat messages way more easy and faster. It also maintains some other features related to the chat.

class pyplanet.contrib.chat.**ChatManager**(*instance*)

The Chat manager is available with: `instance.chat` shortcut.

execute(**queries*)

Execute and send one or multiple chat messages (prepared queries or raw strings) with a multicall.

Parameters **queries** – One or more query instances or one or multiple strings that gets send as global messages.

Returns The results of the multicall.

prepare(*message=None*, *raw=False*)

Prepare a Chat Query by returning a Chat Query object.

Parameters

- **message** – Messsage predefined or build later.
- **raw** – Don't append prefixes or add any automatic message parts.

Returns Query instance

Return type pyplanet.contrib.chat.query.ChatQuery

prepare_raw(*message=None*)

Prepare raw message query without prefixes!

Parameters **message** – Predefined message.

Returns Query instance

Return type pyplanet.contrib.chat.query.ChatQuery

pyplanet.utils

pyplanet.utils.gbxparser

exception pyplanet.utils.gbxparser.**GbxException**

Exception with parsing the Gbx file.

class pyplanet.utils.gbxparser.**GbxParser**(*file=None*, *buffer=None*)
Async GBX Map Information Parser.

Author: Toffe.

seek(*offset*)
We need to override the second param to move from the current position.

Parameters **offset** (*int*) – offset to move away.

pyplanet.utils.style

pyplanet.utils.style.**STRIP_ALL = {'part': '\\$[lh]\\[.+\\]|\\$[lh]|\\$[0-9a-f]{3}', 'letters': 'wnoitsgz<>'}**
Strip all custom maniaplanet styles + formatting.

pyplanet.utils.style.**STRIP_CAPITALS = {'letters': 't'}**
Strip capital style ($t).

pyplanet.utils.style.**STRIP_COLORS = {'part': '\\$[0-9a-f]{3}', 'letters': 'g'}**
Strip colors from your input (including $g, color reset).

pyplanet.utils.style.**STRIP_LINKS = {'part': '\\$[lh]\\[.+\\]|\\$[lh]'}**
Strip links ($h and $l).

pyplanet.utils.style.**STRIP_SHADOWS = {'letters': 's'}**
Strip shadow style ($s).

pyplanet.utils.style.**STRIP_SIZES = {'letters': 'wnoiz'}**
Strip all size and adjustments styles ($w $n $o $i $z).

pyplanet.utils.style.**style_strip**(*text*, **strip_methods*, *strip_styling_blocks=True*, *keep_reset=False*, *keep_color_reset=False*)
Strip styles from the Maniaplanet universe.

Examples:

```
print("--- Strip: colours ---")
print(style_strip("$i$fffMax$06fSmurf$f00.$fffes$$l$09f.$fffm$08f$a5x$n$w$o",␣
↪STRIP_COLORS))
print(style_strip("$l[some link]$i$FFFMax$06fSmurf$f00.$fffesl$09f.$fffm$08fx$l",␣
↪STRIP_COLORS))
print(style_strip("$l[some link]$i$fffMax$06fSmurf$f00.$fffesl$09f.$fffm$08fx",␣
↪STRIP_COLORS))
print("--- Strip: links ---")
print(style_strip("$l$i$fffMax$06fSmurf$f00.$fffesl$09f.$fffm$08f$a5x$l", STRIP_
↪LINKS))
print(style_strip("$i$fffMax$06fSmurf$f00.$fffesl$09f.$fffm$08f$a5x", STRIP_
↪LINKS))
print(style_strip("$l[some link]$i$fffMax$06fSmurf$f00.$fffes$$l$09f.$fffm$08fx$l
↪", STRIP_LINKS))
print(style_strip("$l[some link]$i$fffMax$06fSmurf$f00.$fffesl$09f.$fffm$08fx",␣
↪STRIP_LINKS))
print("--- Strip: sizes ---")
print(style_strip("$i$n$fffMax$06fSmurf$f00.$w$o$fffe$$nsl$09f.$w$fffm$08f$a5$ox",
↪ STRIP_SIZES))
print("--- Strip: everything ---")
print(style_strip("$h$i$fffMax$06fSmurf$f00.$fffesl$09f.$fffm$08f$a5x$h", STRIP_
↪ALL))
print(style_strip("$l[some link]$i$fffMax$06fSmur$$f$f00.$fffesl$09f.$fffm$08fx$l
↪"))
```

```
print(style_strip("$l[some link]$i$fffMax$06fSmu$nrf$f00.$fffesl$09f.$fffm$08fx"))
# Other stuff.:
print(style_strip("$l[some link]$i$fffMax$06fSmu$nrf$f00.$fffesl$09f.$fffm$08fx",␣
↪STRIP_CAPITALS, STRIP_SHADOWS))
```

.

Parameters

- **text** (*str*) – The input string text.
- **strip_methods** – Methods for stripping, use one of the STRIP_* constants or leave undefined to strip everything.
- **strip_styling_blocks** (*bool*) – Strip all styling blocks ($> and $<)
- **keep_reset** (*bool*) – Keep full resets ($z).
- **keep_color_reset** (*bool*) – Keep color resets ($g).

Returns Stripped style string.

Return type str

pyplanet.utils.times

pyplanet.utils.times.**format_time**(*time*)

Format time from integer milliseconds to string format that could be displayed to the end-user.

Parameters **time** (*int*) – Integer time in milliseconds.

Returns String output

Return type str

CHAPTER 18

Support, Bug report & Contact

If you have any problems with starting PyPlanet, please report it on GitHub: https://github.com/PyPlanet/PyPlanet

If you have any problems that are maybe not that PyPlanet related, please referer to the Maniaplanet Forum: https://forum.maniaplanet.com/

Demo Servers

There are several demo servers available. You need to search for servers on the following logins:

- toffestaging1
- toffestaging2

Who is behind PyPlanet

Core:

- Toffe: Lead developer of the PyPlanet project.

Apps:

- TheM: Bundled Application developer.

Want to help us? Contact Toffe on Discord or Forum: `Toffe#8999` or Forum Profile.

Tip: The pages on this site will be updated

CHAPTER 19

Privacy

Error reports

We have an automated error reporting system in place that tells the developers or PyPlanet when there are instabilities in the core code or in one of the contributed and bundled apps.

What are we catching and reporting

We will catch so called uncaught exceptions, these are mostly not handled by one of the functions inside of the code. When it's not handled, the whole function or call to that function is halt and stopped. When this happends, we send an report with the full traceback (all the touched lines of code in order) with the data of the exception.

We also have some specific messages we forward towards the error reporting service. For this kind of messages it's really important we get to know them. For example a memory leak in one of the apps or contributed apps. Or a captured exception but it's not known or handled right.

What data are we sending with the report

Depending on the setting we send minimum of full data about your installation and server. By default you will contribute with the full data option.

Full (level 2 or 3):

- Server dedicated login, paths to maps, scripts, modes, all kind of dedicated configuration.
- Variable data in local method, inside of the exception call.
- Exception with trace or error message including module and line.
- Share information (filtered to only the basic information, *no user specific data!!*) to app developers (must be level 3)

Minimum (level 1):

- Exception with basic trace or error message.

Where will the data be stored

All the data will be stored in an analyse and reporting tool that is fully self hosted by Toffe. We protect the installation with HTTPS and don't allow unauthorized or non-pyplanet team members to access the data.

What about sensitive information

We will replace any sensitive information that seems to be either a key, serial or password by asterisk. This is done in the reporting process.

How to change the behaviour of reporting

You can add this line to your *base.py* file to change the behaviour.

```
# Error reporting
# See documentation for the options, (docs => privacy).
# Options:
# 0 = Don't report any errors or messages.
# 1 = Report errors with only traces.
# 2 = Report errors with traces and server data.
# 3 = Report errors with traces and server data, provide data to contributed apps␣
↪(only pyplanet team has access).
LOGGING_REPORTING = 3
```

Warning: **We really like to improve the stability of PyPlanet**, therefor we kindly ask you to keep the setting on, or at least at level `2`.

CHAPTER 20

Changelog

0.3.3

Core

- Bugfix: Ignore erros with unknown login for ui updates. (means the player left).

Apps

- Bugfix: Fixing issues with dedimania and unsupported maps.
- Bugfix: Fixing issues with dedimania and replays.
- Bugfix: Fixing issues with local records widget showing the wrong offset.
- Bugfix: Fixing issues with local records and double records.
- Improvement: Some not visible improvements to the local record widget logic.

0.3.2

Core

- Bugfix: Not properly sending and handling mode changes.
- Bugfix: Several errors in handling the callbacks in shootmania

Apps

- Bugfix: Fixing issue with removing or erasing maps.

- Improvement: Dedimania now also works in cup mode.
- Feature: Add //replay command for admins, make it able to juke the current map for non-players (ops and admins)

0.3.1

Core

- Improvement: Multiple namespaces per command + improve help.
- Improvement: Hide the alt menu in shootmania when having a window in the middle.
- Improvement: Add method to retrieve map by index.
- Bugfix: Save boolean in the //settings
- Bugfix: Fixing issue with writing the map list.
- Bugfix: Handling of fetching player in a callback for shootmania.
- Bugfix: Several fixes for shootmania modes.

Apps

- Improvement: Make dedimania record message shorter.
- Bugfix: Double prefix in leave messages.
- Bugfix: Dedimania nickname fetching gave error. Sometimes the widget didn't work properly.
- Bugfix: Improve error handling in Dedimania.
- Bugfix: Fixing issue with write map list (admin part of it).
- Bugfix: Don't display the time of the author when in shootmania

0.3.0

Core

- Feature: Refactor the app config class so you can define apps in __init__.py and use shorter configuration, (backward compatible for current contrib apps).
- Feature: Signals runs with gather mode (parallel) now. Makes this way more faster!
- Feature: Add save hook to setting object.
- Feature: Chat contrib component, for shorter syntax at sending and preparing chat messages.
- Feature: Refactor the GBX component, for shorter syntax at sending and preparing Gbx Methods.
- Feature: Make it able to change the UI Properties from the games
- Feature: Add 'suggestion or bug' report button.
- Improvement: Unknown command message.
- Improvement: Makes it faster to display local records.

- Improvement: Refactor the local record code.

Apps

- Feature: Add Live Rankings app (beta). Add it to your apps.py!
- Feature: Add chat announce limit in local and dedi records.
- Improvement: Autosave matchsettings on insertion of map.
- Improvement: Hide dedimania widget on downtime.
- Improvement: Better error handling in dedimania app.
- Bugfix: Fixing issue with displaying WhoKarma list.
- Bugfix: Fixing path issues in MX app.

0.2.0

Core

- Feature: Improved performance with the all new Performance Mode. This will improve performance for a player threshold that is freely configurable.
- Feature: Technical: Add option to strip styles/colors from searchable column in listviews.
- Feature: Technical: Add shortcut to get an app setting from global setting manager.
- Improvement: Improve log color for readability.
- Bugfix: Fixing issue with integer or other numeric values and the value 0 in the //settings values.
- Bugfix: Replace invalid UTF-8 from the dedicated response to hotfix (dirty fix) the bug in client with dedicated communication.

Apps

- Feature: New app: Transactions: Features donations and payments to players as the actual planets stats. Activate the app now in your apps.py!!
- Feature: Map info shows nickname of author if the author nickname is known.
- Feature: /list [search] directly searching in map list.
- Feature: Implement //modesettings to show and change settings of the current mode script.
- Feature: Restrict karma voting to count after the player finishes the map for X times (optional).
- Feature: Apply the performance mode improvements to the local and dedimania records widgets.
- Feature: Add command to restart PyPlanet pool process. //reboot
- Improvement: Changed dedimania record text chat color.
- Improvement: Changed welcome player nickname default color (white).
- Improvement: Reduced length of record chat messages.
- Improvement: Add player level name to the join/leave messages.

- Bugfix: Jukebox current map gives errors and exceptions.
- Bugfix: Ignore color and style codes inside /list searching.
- Bugfix: Some small improvements on widgets (black window behind local/dedi removed and more transparent)

0.1.5

Core

- Bugfix: Fixing several issues related to the connection and parsing of the payload.
- Bugfix: Fixing issue with the BeginMatch callback.
- Bugfix: Change issues related to the utf8mb4 unicode collate (max index lengths).

Apps

- Bugfix: Fixing several issues with the dedimania app.
- Bugfix: Fixing issue with local and dedimania records being saved double (2 records for 1 player). (#157).
- Bugfix: Fixing several exception handling in dedimania app.

0.1.4

Core

- Bugfix: Undo locking, causing freeze.

0.1.3

Apps

- Bugfix: Fixing issue in dedimania causing crash.

0.1.2

Core

- Bugfix: Filter out XML parse error of Dedicated Server (#121).
- Bugfix: Give copy of connected players instead of a reference to prevent change of list when looping (#117).
- Bugfix: Fixing issue when player rapidly connects and disconnects, giving error (#126 & #116).

Apps

- Bugfix Karma: Fixing whokarma list not displaying due to error (#122 & #118).
- Bugfix Dedimania: Reconnection issues (#130).
- Improvement Local Records: Improve performance on sending information (chat message) on large servers. (#139).
- Improvement Dedimania Records: Improve performance on sending information (chat message) on large servers. (#139).
- Improvement Dedimania Records: Improve the error reporting and implement shorter timeout + retry procedure (#139).

0.1.1

Core

- Fixing issue with creating migrations folder when no permission.

0.1.0

Core

- Add new fields to the `game` state class.
- Refresh the `game` infos every minute.

Contrib Apps

- NEW: Dedimania App: Adding dedimania integration and widget.

0.0.3

Contrib Apps

- Bugfix Local Records: Widget showing wrong offset of records. (Not showing own record if just in the first part of >5 recs) (#107).

0.0.2

Contrib Apps

- Bugfix Local Records: Widget not updating when map changed. Login not found exception. (#106).

0.0.1

Core

- First implementation of the core.
- First implementation of the CLI tool.

Contrib Apps

Admin *pyplanet.apps.contrib.admin*

- Feature: Basic map functions: skip / restart / add local / remove / erase / writemaplist
- Feature: Basic player functions: ignore / kick / ban / blacklist
- Feature: Basic server functions: set passwords (play / spectator)

Map list + jukebox *pyplanet.apps.contrib.jukebox*

- Feature: Display maplist with maps currently on the server
- Feature: Basic jukebox functions: list / drop / add / clear (admin-only)

Map karma *pyplanet.apps.contrib.karma*

- Feature: Basic map karma (++ / –)
- Feature: Display who voted what (whokarma)

Local records *pyplanet.apps.contrib.local_records*

- Feature: Saving local records
- Feature: Display current first/personal record on map begin (in chat)
- Feature: Display list of records

Playerlist *pyplanet.apps.contrib.players*

- Feature: Add join/leave messages.

MX *pyplanet.apps.contrib.mx*

- Feature: Add MX maps (//add mx [id(s]).
- Feature: Implement MX API Client.

CHAPTER 21

Todo (docs)

Todo

Write introduction.

original entry

Todo

Write introduction + examples

original entry

Todo

Write introduction + examples.

original entry

Todo

Write introduction.

original entry

Todo

Write introduction.

original entry

Todo

Write introduction.

original entry

CHAPTER 22

Some thoughts from experts

CHAPTER 23

Indices and tables

- genindex
- modindex
- search

CHAPTER 24

Links

Documentation: http://pypla.net/
GitHub: https://github.com/PyPlanet/PyPlanet
PyPi: https://pypi.python.org/pypi/pyplanet

Python Module Index

p

Index

Symbols

A

B

C

D

H

I

L

M

N

O

P

R

S

T

U

V

W

X