PyPlanet Documentation

Release 0.7.4

PyPlanet (Tom Valk)

Mar 04, 2020

USER DOCUMENTATION

PyPlanet is a Maniaplanet Dedicated Server Controller that works on Python 3.5 and later. Because Maniaplanet is using a system that can be event based we use AsyncIO to provide an 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: Admin Toolbar, Providing mostly used admin functions within a few clicks.
- App: Karma, Let your players vote on your maps! Includes MX Karma integration.
- 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).
- App: Sector Times, Compare your checkpoint time against your local or dedimania record. (Trackmania).
- App: Dynamic Pointlimit, Royal point limit adjustment based on the number of players. (Shootmania Royal).
- App: CP Times, Show the best checkpoint times on top of your screen.
- App: Chat based voting, No more uncontrollable and unfair Call Votes. Use chat based voting.
- App: Vote to extend the TimeAttack limit instead of restarting the map! Extend-TA© command and voting is awesome!
- App: Waiting Queue, no more unfair and spamming of the join button, fairly queue spectators to join your full server.
- App: Add links to your PayPal donate page or Discord server.

Do you want to install PyPlanet, head towards our *Getting Started Manual*. Want to see PyPlanet in action, head to *Screenshots*.

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

ONE

GETTING STARTED (INSTALLATION)

1.1 Requirements

PyPlanet runs on Python 3.5 and later. Most linux distributions contain default packages or will come with Python pre-installed. If you don't have Python 3.5 you can still continue the installation, we will help you through the installation of Python 3.5 in our installation guides!

Summary of requirements:

- Python 3.5+ and pip 9.
- MySQL Server or PostgreSQL Server.
- Maniaplanet Dedicated (Maniaplanet 4 is minimum)

Installation manuals:

Please head to one of our installation manuals to continue:

Linux Guide or *Windows Guide*

1.2 Installation on Linux

Contents

1.2.1 1. Operating System needs

PyPlanet requires Python 3.5 and later. We also require to have some operating system libraries and build tools installed. We will guide you through the steps that are required to install those requirements in this subtopic.

Debian / Ubuntu

Install the operating system requirements by executing the following commands:

```
sudo apt-get update && sudo apt-get install build-essential libssl-dev
libffi-dev python3-dev zlib1g-dev liblzma-dev
```

Fedora / RHEL based

Install the operating system requirements by executing the following commands:

```
sudo yum install gcc libffi-devel python3-devel openssl-devel zlib zlib-devel
bzip2 bzip2-devel xz xz-devel sqlite sqlite-devel.
```

1.2.2 2. Install PyEnv and Python

To make things as easy as possible we are going to use *PyEnv*. It's a tool that will install Python for you with all the requirements and also manage to adjust the environment we are running in.

The following steps are the same for all distributions.

Note: Make sure you are logged in as the user that is going to run PyPlanet. (Mostly not root!).

Install PyEnv

```
curl -L https://raw.githubusercontent.com/pyenv/pyenv-installer/master/bin/pyenv-
↪installer | bash
printf '\nexport PATH="$HOME/.pyenv/bin:$PATH"\neval "$(pyenv init -)"\neval "$(pyenv␣
↪virtualenv-init -)"\n' >> ~/.bashrc
source ~/.bashrc
```

Install Python

```
pyenv install 3.7.6
pyenv global 3.7.6
```

Attention: The first set of commands makes adjustments to the `~/.bashrc` file. It can be that you don't have this file installed.

If that is the case, you can add those lines manually to any other script that is executed when you open your shell (`.profile`) or execute these commands manually at every start of a SSH session. Your SSH session might have to be restarted after this change!

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

1.2.3 3. Create environment for your installation

We recommend to separate multiple installations by creating a so called virtual environment. This will make sure you can run several PyPlanet and dependency versions on the same Python installation. You can skip this step if you don't want to use virtual environments, but we recommend to use it.

Create virtualenv:

```
pyenv virtualenv 3.7.6 my-env
# Where 'my-env' is your environment name, you need adjust this if you have multiple␣
↪installations.
```

Activate virtualenv:

Note: You have to activate your virtual environment **every time** you want to execute PyPlanet commands! That means that you have to activate before you update, start, develop and do anything with PyPlanet!

```
pyenv activate my-env
# Where 'my-env' is your environment name, you need adjust this if you have multiple␣
↪installations.
```

1.2.4 4. PyPlanet Installation

PyPlanet is published through the Python Package Index (PyPi) and is easy to install with `pip`.

```
pip install pyplanet --upgrade
```

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

1.2.5 5. 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`.

To upgrade your existing installation, see our *Upgrading Guide*.

Warning: If you use the virtual environment we installed in *3. Create environment for your installation*, make sure you activate it **before you install or update dependencies**!

Head to the next step

Configure your PyPlanet installation now by going to the next chapter: *Configuring PyPlanet*.

1.3 Installation on Windows

Contents

1.3.1 1. Installing Python

If you have not yet installed Python 3.5 or later on your Windows machine, do it now by going to the following link:

https://www.python.org/downloads/release/python-370/

Head towards the end of the page and click on the *Windows x86-64 executable installer* link. After starting the executable you will get an wizard.

Make sure it looks like this and click on the red area to continue.

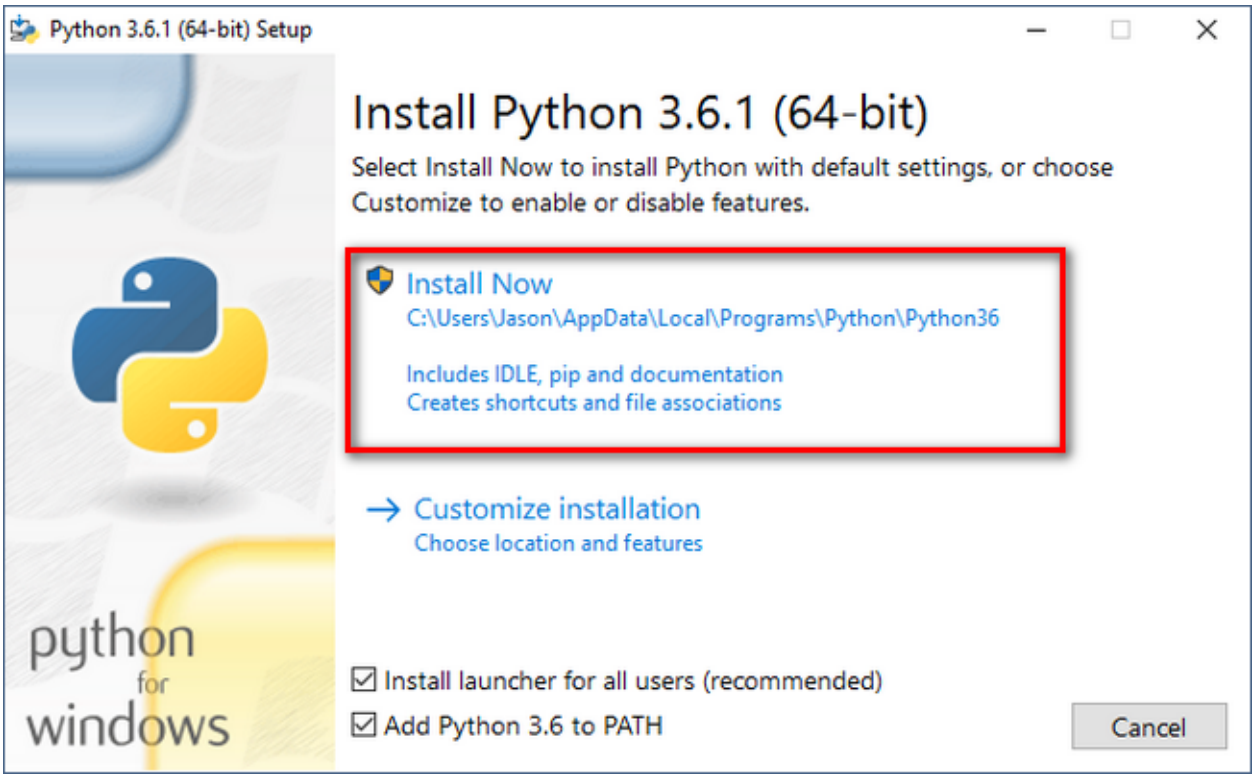


Fig. 1: Setup wizard with the checkboxes enabled.

Note: Make sure you checked the two checkboxes: *Install launcher for all users* and *Add Python to PATH*.

1.3.2 2. Creating Virtual Environment

To prevent the usage of the administration leverage and to benefit from multiple PyPlanet installations and a clean environment we recommend to setup a Virtual Environment.

First of all we need to install the `virtualenv` package. To do so, open a terminal screen by hitting start and write cmd. Open the command prompt.

```
pip install virtualenv
```

After this we will initiate the environment, you can do this by going to your directory where you want to setup the PyPlanet installation. Create a folder somewhere that is empty and ready for the PyPlanet settings and other files.

Open a terminal in this folder by holding `SHIFT` and `Right click` on an empty space in the folder. Then click `Open terminal here`.

In the terminal, type the following command to create the environment:

```
virtualenv 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:

```
# Windows, in your command prompt
env\Scripts\activate.bat
```

1.3.3 3. PyPlanet Installation

PyPlanet is published through the Python Package Index (PyPi) and is easy to install with the `pip` commands.

```
pip install pyplanet --upgrade
```

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

1.3.4 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.

Because you have created an Virtual Environment earlier you want to store your 'project' in the same folder. You can do this with the following command:

```
pyplanet init_project .
```

After setup your project, you have to install or update your dependencies from your local `requirements.txt`.

To upgrade your existing installation, see our *Upgrading Guide*.

> **Warning:** If you use the virtual environment we installed in *3. Create environment for your installation*, make sure you activate it **before you install or update dependencies**!

Head to the next step

Configure your PyPlanet installation now by going to the next chapter: *Configuring PyPlanet*.

CHAPTER

TWO

CONFIGURING PYPLANET

Settings method is the method to read out settings. This can be one of the following methods/backends:

- *python*: Default, the python loader uses the files `base.py` and `apps.py` in the `PYPLANET_SETTINGS_MODULE` provided.
- json: Read json files `base.json` and `apps.json` in the provided `PYPLANET_SETTINGS_DIRECTORY` directory.
- yaml: Read yaml files `base.yaml` and `apps.yaml` in the provided `PYPLANET_SETTINGS_DIRECTORY` directory.

Settings module (python only) is where the PyPlanet settings are stored for python backend. 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`.

Settings directory (json and yaml only) is where the two configuration files are located for the file based backends such as JSON or YAML.

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`). For both other backends its quite the same.

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.

Case sensitive: Only the keys are not case insensitive (with exception of the Python backend). The value and the subkeys are all case sensitive!

> **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.
>
> Also, the **JSON** examples always contain the opening and closing brackets in the examples. In a real file you would have these only once around the whole file.

2.1 Debug Mode (base)

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`. **This only works for the python config backend**

Listing 1: base.yaml

```yaml
DEBUG: false
```

Listing 2: base.json

```json
{
  "DEBUG": false
}
```

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

2.2 Pool defining (base)

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

Listing 3: base.py

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

Listing 4: base.yaml

```yaml
POOLS:
  - default
```

Listing 5: base.json

```
{
  "POOLS": [
    "default"
  ]
}
```

2.3 Owners (base)

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.

Listing 6: base.py

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

Listing 7: base.yaml

```
OWNERS:
  default:
    - your-maniaplanet-login
    - second-login
```

Listing 8: base.json

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

2.4 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 require for MySQL: `utf8mb4_unicode_ci` to work with the new symbols in Maniaplanet. Also, please make sure your MySQL installation uses InnoDB by default, more information can be found here: *MySQL Index Error*

Create MySQL Database by running this command:

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

Configuration

Configuration can follow the following examples:

Listing 9: base.py

```
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': 'utf8mb4',
    }
  }
}
```

Listing 10: base.yaml

```
DATABASES:
  default:
    ENGINE: 'peewee_async.MySQLDatabase'
    NAME: 'pyplanet'
    OPTIONS:
      host: 'localhost'
      user: 'pyplanet'
      password: 'pyplanet'
      charset: 'utf8mb4'
```

Listing 11: base.json

```
{
  "DATABASES": {
    "default": {
      "ENGINE": "peewee_async.MySQLDatabase",
      "NAME": "pyplanet",
      "OPTIONS": {
```

(continues on next page)

(continued from previous page)

```
        "host": "localhost",
        "user": "pyplanet",
        "password": "pyplanet",
        "charset": "utf8mb4"
      }
    }
  }
}
```

2.5 Dedicated Server (base)

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

Listing 12: base.py

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

Listing 13: base.yaml

```
DEDICATED:
  default:
    HOST: '127.0.0.1'
    PORT: '5000'
    USER: 'SuperAdmin'
    PASSWORD: 'SuperAdmin'
```

Listing 14: base.json

```
{
  "dedicated": {
    "default": {
      "HOST": "127.0.0.1",
      "PORT": "5000",
      "USER": "SuperAdmin",
      "PASSWORD": "SuperAdmin"
    }
  }
}
```

2.6 Server files settings (base)

Some of these settings are required to be able to save match settings and to save the blacklisted players for example.

Listing 15: base.py

```
# 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',
}

# Blacklist file is managed by the dedicated server and will be loaded and writen to␣
↪by PyPlanet once a
# player gets blacklisted. The default will be the filename Maniaplanet always uses␣
↪and is generic.
BLACKLIST_FILE = {
  'default': 'blacklist.txt'
}
```

Listing 16: base.yaml

```
MAP_MATCHSETTINGS:
  default: 'maplist.txt'

BLACKLIST_FILE:
  default: 'blacklist.txt'
```

Listing 17: base.json

```
{
  "MAP_MATCHSETTINGS": {
    "default": "maplist.txt"
  },
  "BLACKLIST_FILE": {
    "default": "blacklist.txt"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

2.7 Storage (base)

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:

Listing 18: base.py

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

Listing 19: base.yaml

```
STORAGE:
  default:
    DRIVER: 'pyplanet.core.storage.drivers.local.LocalDriver'
```

Listing 20: base.json

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

Using SFTP/SCP/SSH

> **Error:** The SFTP/SCP/SSH driver doesn't work for now! It's planned to be implemented later on if there are enough use-cases.

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',
```

(continues on next page)

(continued from previous page)

```
    '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

2.8 Cache (base)

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

2.9 Self Upgrade (base)

New since 0.6.0 is the self-upgrader where the master admins can self upgrade the PyPlanet installation from within the game. You don't want this to be enabled on shared servers (hosting environments) as it may break your installation.

Listing 21: base.py

```
SELF_UPGRADE = True
```

Listing 22: base.yaml

```
SELF_UPGRADE: true
```

Listing 23: base.json

```
{
  "SELF_UPGRADE": true
}
```

Warning: Using the self-upgrade (//upgrade and `pyplanet upgrade`) is very experimental. The method can break your installation. We don't guarantee the working of the method.

We advice to use the manual PIP method of upgrading over the in-game upgrading process!

2.10 Songs (base)

Note: This setting only works in combination with the `music_server` app. Enable the app by adding the app in your apps.py (or apps.json/apps.yaml).

You can add URL's of the music to the SONGS list.

Listing 24: base.py

```
SONGS = {
  'default': [
    'http://urltoogg'
  }
}
```

Listing 25: base.yaml

```
SONGS:
  default:
    - 'http://urltoogg'
```

Listing 26: base.json

```
{
  "SONGS": {
    "default": [
      "http://urltoogg"
    }
  }
}
```

2.11 Logging (base)

By default (from version 0.5.0) rotating logging is enabled by default but writing is disabled by default. The settings bellow can be adjusted to meet your requirements.

Listing 27: base.py

```
LOGGING_WRITE_LOGS = True
LOGGING_ROTATE_LOGS = True
LOGGING_DIRECTORY = 'logs'
```

Listing 28: base.yaml

```
LOGGING_WRITE_LOGS: true
LOGGING_ROTATE_LOGS: true
LOGGING_DIRECTORY: 'logs'
```

Listing 29: base.json

```
{
  "LOGGING_WRITE_LOGS": true,
  "LOGGING_ROTATE_LOGS": true,
  "LOGGING_DIRECTORY": "logs"
}
```

2.12 Enabling apps (apps)

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.

Listing 30: apps.py

```
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.info',
```

(continues on next page)

(continued from previous page)

```
    'pyplanet.apps.contrib.mx',
    'pyplanet.apps.contrib.transactions',

    # New since 0.4.0:
    'pyplanet.apps.contrib.sector_times',
    'pyplanet.apps.contrib.dynamic_points',

    # New since 0.5.0:
    'pyplanet.apps.contrib.clock',
    'pyplanet.apps.contrib.best_cps',
    'pyplanet.apps.contrib.voting',

    # New since 0.6.0:
    'pyplanet.apps.contrib.queue',
    'pyplanet.apps.contrib.ads',
    'pyplanet.apps.contrib.music_server',
  ],
}
```

Listing 31: apps.yaml

```
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.info'
    - 'pyplanet.apps.contrib.mx'
    - 'pyplanet.apps.contrib.transactions'

    # New since 0.4.0:
    - 'pyplanet.apps.contrib.sector_times'
    - 'pyplanet.apps.contrib.dynamic_points'

    # New since 0.5.0:
    - 'pyplanet.apps.contrib.clock'
    - 'pyplanet.apps.contrib.best_cps'
    - 'pyplanet.apps.contrib.voting'

    # New since 0.6.0:
    - 'pyplanet.apps.contrib.queue'
    - 'pyplanet.apps.contrib.ads'
    - 'pyplanet.apps.contrib.music_server'
```

Listing 32: apps.json

```
{
  "APPS": {
    "default": [
      "pyplanet.apps.contrib.admin",
      "pyplanet.apps.contrib.jukebox",
      "pyplanet.apps.contrib.karma",
      "pyplanet.apps.contrib.local_records",
```

(continues on next page)

(continued from previous page)

```
        "pyplanet.apps.contrib.dedimania",
        "pyplanet.apps.contrib.players",
        "pyplanet.apps.contrib.info",
        "pyplanet.apps.contrib.mx",
        "pyplanet.apps.contrib.transactions",

        "pyplanet.apps.contrib.live_rankings",
        "pyplanet.apps.contrib.sector_times",

        "pyplanet.apps.contrib.clock",
        "pyplanet.apps.contrib.best_cps",
        "pyplanet.apps.contrib.voting",

        "pyplanet.apps.contrib.queue",
        "pyplanet.apps.contrib.ads",
        "pyplanet.apps.contrib.music_server"
    ]
  }
}
```

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.

CHAPTER

THREE

STARTING PYPLANET

Contents

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

By default, PyPlanet will always run in the foreground. That's why we have several steps to make PyPlanet run in the background and as a service on your server. As a side-note we also have the screen method described. It's a matter of preference and support.

Hint: If you use an virtual environment, make sure it's activated. We will not show this in some instructions, but always activate before starting PyPlanet.

3.1 Start and fork to PID file (Linux)

This is available from PyPlanet 0.5.0. With this feature you can start PyPlanet and let it detach itself and write a so called PID file which contain the process ID of the detached process. This is only available on Linux systems.

3.1.1 1. Starting detached

Starting detached is as simple as it seems to be. Look at the starting command bellow and you will understand how to start PyPlanet detached.

```
./manage.py start --detach --pid-file=pyplanet.pid
```

This way you can create your own startup scripts. You can terminate PyPlanet by using the following command:

```
kill -SIGTERM `cat pyplanet.pid`
```

3.2 Start/stop with Screen (Linux)

Screen is a feature on Linux distributions that makes it possible to start a virtual terminal window, and keep the terminal open in the background for as long as required. You can watch or control the screen from multiple SSH sessions, making it ideal for platforms that require multiuser access to the servers while not require the root privileges required for the services.

3.2.1 1. Installation of screen

To use Screen for PyPlanet you have to install it for your OS.

Debian / Ubuntu::

```
sudo apt-get install screen
```

Fedora / RHEL:

```
sudo yum install screen
```

3.2.2 2. Start a new screen

You can start a new screen session with this command. Remember that you only have to do this once for starting a new session. After executing this command you will create and directly attach to this screen instance.

```
screen -S name-of-screen
```

3.2.3 3. Open a screen

If you have followed step 2, please skip this step, this step is meant for so called 'reattaching' to the screen.

To list the screens on this user account use: `screen -ls`.

To reattach to a deattached screen, use: `screen -r name-of-screen`. If you can't attach, you might have another session attached or need to use the numeric screen id's from the list command.

To reattach to an already attached screen, use: `screen -x name-of-screen`. Again, if this fails, try the numeric id from the list command.

From now you are in the virtual terminal session, when you accidentally disconnect your SSH tunnel, the process inside the screen will still be active!

3.2.4 4. Start PyPlanet

Make sure you activated your virtual environment first.

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 project environment(s).

3.2.5 5. Leaving the screen

To leave the screen the right way (deattach) you have to do the following keyboard combination:

`CTRL+A` then release, and press `D`.

If you want to exit and **destroy** the screen, just cancel all programs inside, and type `logout` or use `CTRL+D`.

3.3 Install SystemD Service (Linux)

SystemD is a pretty new init system that is included in the newest distributions. For example, Ubuntu 16.04 and higher, Debian 8 and higher make use of SystemD. SystemD will replace the old sysvinit system and make it easy to start/stop and automatically restart services (including during the OS boot)

> **Warning:** This method is slightly harder, and require you to have root rights al the time (even to (re)start).
>
> **This also requires you to use PyEnv.**

3.3.1 1. Installing the service

Head towards your systemd configuration folder by executing the following command(s):

Debian / Ubuntu / Fedora / RHEL / Most other Linux distros::

```
cd /etc/systemd/system
```

3.3.2 2. Determinate paths

First of all, we have to know the following paths:

1. Full path to the PyPlanet executable.
2. Full path to the project root.
3. The user and group you want to run PyPlanet under.
4. Your service name. (in our examples `pyplanet.service` and `pyplanet`)

2.1. Full PyPlanet path

You can check the full path to the pyplanet cli interface by executing this: `whereis pyplanet`. The outcome is the path, in our example it's `/home/toffe/.pyenv/shims/pyplanet`.

2.2. Full project path

Where is the root of the PyPlanet project located, this is the folder where the `settings` folder and the `manage.py` file exist. In our example it's `/path/to/your/pyplanet/project`.

2.3. Running user and group

It's important to not run as root! That's why you want to use a secondary user on your system.

Find out the current user and group name with the following command: `echo id` (don't execute with sudo!).

This will output something like this:

```
uid=1000(toffe) gid=1000(toffe) groups=1000(toffe),4(adm),24(cdrom),27(sudo),30(dip),
↪46(plugdev),113(lpadmin),128(sambashare),133(wireshark),140(kvm),141(libvirtd),
↪998(bumblebee),999(docker)
```

We only need two items in there, and its the value inside of the brackets of the first item (*uid=x*), in our case `toffe` which is the user.

And the second value is the group, just after the *gid=x*, and inside the brackets, in our case also `toffe`.

3.3.3 3. Create the service definition file

After going to the right location you have to create a new file called `pyplanet.service`. You can rename it as you want!

```
sudo nano pyplanet.service
# Or use your os editor, like vim or pico. Make sure you are still in the folder from␣
↪step 1!
```

After opening the editor, paste the contents bellow and change the contents according the steps above.

```
[Unit]
After=syslog.target network.target

[Service]
WorkingDirectory=/path/to/your/pyplanet/project
Environment="PYTHONPATH=/path/to/your/pyplanet/project"
ExecStart=/home/toffe/.pyenv/shims/pyplanet start --settings=settings
SyslogIdentifier=pyplanet

Restart=always
StandardOutput=syslog
StandardError=syslog
User=toffe
Group=toffe

[Install]
WantedBy=multi-user.target
```

After changing the contents, save the file and continue to the next step.

3.3.4 4. Reload systemd

After installing the new service file you have to let systemd know that you changed something. Do this with the following command:

```
sudo systemctl daemon-reload
```

3.3.5 5. Starting/stopping PyPlanet

From now you can start, stop and restart your controller with the following commands: (the pyplanet name is your service file name).

```
systemctl start pyplanet
systemctl stop pyplanet
systemctl restart pyplanet
```

To view the logs of the PyPlanet instance, type one of this commands:

```
journalctl --unit pyplanet.service -xe
journalctl --unit pyplanet.service -f
```

3.3.6 6. Starting at boot

Activate the service to have it started when your machine starts.

```
systemctl enable pyplanet
```

3.4 Start standalone and in foreground (Linux and Windows)

Warning: When you are using SSH to remotely access the server running PyPlanet, this starting option should only be used while testing your server. The moment you close the SSH terminal window, your PyPlanet instance will shutdown (crash). Use one of the methods above, if you want to run PyPlanet without having the terminal window open at all times.

3.4.1 1. Go to your project folder

Make sure you change directory to your project root (contains the `manage.py` file).

```
cd /my/project/location
```

3.4.2 2. Activate virtual environment

Make sure you activated your virtual environment.

```
# Linux / Mac OS
pyenv activate pyplanet

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

Tip: Don't know how to setup the environment exactly? Head to *Windows* or *Linux* guides.

3.4.3 3. Start PyPlanet

```
# Linux:
./manage.py start

# Windows
python manage.py start
```

This will start your PyPlanet setup.

CHAPTER

FOUR

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*.

Since 0.6.0 you have two methods of upgrading. The in-game method and the manual PIP method. **We strongly advice you to use the manual PIP method because the in-game upgrade can be unstable with big releases!**

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.

Warning: When using the executable method (downloaded from the GitHub releases page) you will have to redownload and replace the binary file instead of these steps! (Executable currently not released anymore).

4.1 In-game upgrade method

To use this method your current version needs to be 0.6.0 or higher. You can use the following command to execute the upgrade. You can also select a specific version (for example beta or rc) with the command.

```
//upgrade
-- or --
//upgrade 0.6.0-rc1
```

PyPlanet will reboot when the installation is complete. You might want to edit the apps.py to activate the new apps. On the *configuration page* you can always find the latest apps entries.

Warning: This method can be unstable. It's hard to fully adjust to your installation method and environment. We recommend making a backup of your installation, or have the knowledge of restoring or recreating the virtualenv or installation!

4.2 Manual PIP method

4.2.1 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

4.2.2 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
```

4.2.3 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.

4.2.4 4. Upgrade settings

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

4.2.5 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.

4.2.6 6. Start PyPlanet

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

CHAPTER

FIVE

MIGRATING FROM OLD CONTROLLER

5.1 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 to meet your needs:

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

5.2 Migrating from UAseco

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

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

As we don't have anything yet that can hold statistics except the times table (`uaseco_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 to meet your needs:

```
python manage.py db_convert --pool default --source-format uaseco --source-db-
↪username root --source-db-name uaseco
```

Warning: The UAseco converter is new since version 0.4.4.

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

5.3 Migrating from eXpansion

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

- Player basic information.
- Map basic information.
- Local records.
- Karma.

As we don't have anything yet that can hold statistics and the architecture of those statistics is very different in eXpansion, 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 to meet your needs:

```
python manage.py db_convert --pool default --source-format expansion --source-db-
↪username root --source-db-name expansion
```

Warning: The eXpansion converter is new since version 0.5.0. This has not yet been fully tested with several installations. **Make sure your source is using utf8 or utf8mb4_unicode collate**.

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

5.4 Migrating from ManiaControl

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

- Player basic information.
- Map basic information.
- Local records. (`uaseco_records` table).
- Karma.

As we don't have anything yet that can hold statistics, 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 to meet your needs:

```
python manage.py db_convert --pool default --source-format maniacontrol --source-db-
↪username root --source-db-name maniacontrol
```

Warning: The ManiaControl converter is new since version 0.4.5

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

CHAPTER

SIX

HOW TO'S AND TROUBLESHOOTING

6.1 Correct Database Collation (MySQL)

Because of the Emoji and other symbols used in MP4 and later you are required to have the `utf8mb4_unicode_ci` collation for databases, tables and columns in MySQL.

If you didn't set it right at the first start you will get a message when starting the controller. To correct this you can execute the following query. You have to change one part with the database name:

```
USE information_schema;

SELECT concat("ALTER DATABASE `",table_schema,
              "` CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci;") as _sql
  FROM `TABLES`
 WHERE table_schema like "pyplanet"
 GROUP BY table_schema;

SELECT concat("ALTER TABLE `",table_schema,"`.`",table_name,
              "` CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;") as _
↪sql
  FROM `TABLES`
 WHERE table_schema like "pyplanet"
 GROUP BY table_schema, table_name;
```

In this code-snippet, *pyplanet* is the database name. Make sure you change it to your database name.

The results you will get are queries that you need to execute one by one. Please make sure you create a backup before executing the queries.

6.2 MySQL Complaining about large indexes (1000 bytes)

Because we use `utf8mb4_unicode_ci` characters can take more bytes and will reach the limits of the MySQL database engine.

For PyPlanet it's required to have your database storage engine set to `InnoDB`! It's currently an issue that we can't provide the storage engine when creating tables. This makes it kinda frustrating and the workaround for now is to set your MySQL Servers default storage engine to `InnoDB`. To do this, find your my.ini in your MySQL installation, in most cases this is located in the installation directory on Windows, or somewhere in /etc/mysql or the file /etc/my.ini on Linux systems.

Please find the following text in the my.ini file `default-storage-engine`. When you can find the line, change it so it looks like the snippet given bellow. If you can't find the entry in the file, add it to the `[mysqld]` section, and make sure it looks like the snippet bellow.

```
default-storage-engine=InnoDB
```

Warning: We are looking for a better way to solve this issue, but we are limited to the Peewee library for creating the tables.

CHAPTER

SEVEN

ADMIN

7.1 Information

Name: `pyplanet.apps.contrib.admin`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

7.2 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

7.3 Commands

7.3.1 PyPlanet

Reboot PyPlanet Pool Process

Command: `//reboot`

Parameters: None.

Functionality: Reboot pyplanet pool process.

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

Toggle the admin toolbar personally

Command: `//toolbar`

Parameters: None.

Functionality: Toggle the visibility of the admin toolbar personally.

Required permission: at least admin level 1.

7.3.2 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.

Open Map browser

Command: `//localmaps`

Parameters: None.

Functionality: Opens a browser which can be used to add local maps to the server.

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

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.

Read Map list

Command: `//readmaplist` / `//rml`

Parameters:

- Match settings file.

Functionality: Read maplist from the match settings file.

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

Shuffle Map list

Command: `//shuffle`

Parameters:

-

Functionality: Shuffle and reload map list from disk!

Required permission: `admin:shuffle`, 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.

Extend TA limit

Command: `//extend`

Parameters:

- Time in seconds to extend the timer with, ignore this parameter to double the time.

Functionality: Extend the TA limit temporary with given seconds or double the current TA limit.

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

7.3.3 Players

Force player to spec

Command: `//forcespec`

Parameters:

- Player login.

Functionality: Force player into spectator.

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

Force player to player

Command: `//forceplayer`

Parameters:

- Player login.

Functionality: Force player into player slot.

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

Force player to team

Command: `//forceteam`

Parameters:

- Player login.
- Team identifier (0/blue or 1/red)

Functionality: Force player into a specific team.

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

Switch player to team

Command: `//switchteam`

Parameters:

- Player login.

Functionality: Switches the player into the other team.

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

Command: `//warn` / `//warning`

Parameters:

- Player login.

Functionality: Displays a warning message in chat for the player

Required permission: `admin:warn`, 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.

7.3.4 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.

Write Blacklist

Command: `//writeblacklist` / `//wbl`

Parameters:

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

Functionality: Write blacklist to file.

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

Read Blacklist

Command: `//readblacklist` / `//rbl`

Parameters:

- Blacklist file (optional).

Functionality: Read blacklist from the file given or the one in the settings file.

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

7.3.5 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.

Cancel CallVote

Command: `//cancelcallvote` / `//cancelcall`

Parameters: None

Functionality: Cancel a current started call vote.

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

7.4 Signal handlers

None.

CHAPTER

EIGHT

ADVERTISEMENTS

8.1 Information

Name: `pyplanet.apps.contrib.ads`

Depends on:

-

Game: All

8.2 Features

This app provides buttons, banners and other advertisements assets. For example it shows a Discord logo or a PayPal button. The app has the following features: - Show Discord join button. - Show how many users online in Discord. - Show PayPal donate button.

Setup Discord:

1. Get your discord join link and make sure it does not expire.
2. Get your discord server ID. (you might need to enable developer settings)
3. Enable the widget of your discord server in the server settings.
4. Start PyPlanet with this app enabled.
5. Type //settings and edit two discord related fields (join URL and ID)

Setup PayPal:

1. Create the PayPal donation link for you server account
2. Start PyPlanet with this app enabled.
3. Type //settings and fill the PayPal related field (Donation URL)

8.3 Commands

8.3.1 Display Discord Server Info

Command: `/discord`

Parameters: None.

Functionality: Displays the number of users and bots on the server.

Required permission: None.

8.3.2 Display PayPal Link

Command: `/paypal`

Parameters: None.

Functionality: Display the PayPal link in chat.

Required permission: None.

8.4 Signal handlers

8.4.1 Player connect

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

Displaying widgets

CHAPTER

NINE

BEST CPS

9.1 Information

Name: `pyplanet.apps.contrib.best_cps`

Depends on: `core.maniaplanet`

Game: TrackMania

Mode: TimeAttack

9.2 Features

This app shows the best driven time at each CP.

- Quick display on the top of the UI for the first 18 CPs (3 rows)
- Click on header to open up list view for all CPs

9.3 Installation

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

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

9.4 Commands

-

9.5 Signal handlers

9.5.1 Map begin

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

Functionality: Removes CP times from last round.

9.5.2 Player waypoint

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

Functionality: Process and update widget.

9.5.3 Player connect

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

Functionality: Display widget.

9.5.4 Map End

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

Functionality: Update the widget (for map restarts)

CHAPTER

TEN

CLOCK

10.1 Information

Name: `pyplanet.apps.contrib.clock`

Depends on: `core.maniaplanet`

Game: TrackMania, Shootmania

10.2 Features

This app shows a digital clock displaying the current time on the UI. This widget is using ManiaScript.

10.3 Signal handlers

10.3.1 Map begin

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

Functionality: Displays the clock.

10.3.2 Player connect

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

Functionality: Displays the clock widget for the connecting player.

CHAPTER

ELEVEN

DEDIMANIA RECORDS

11.1 Information

Name: `pyplanet.apps.contrib.dedimania`

Depends on: `core.maniaplanet`

Game: TrackMania

Mode: TimeAttack + Rounds

11.2 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.

11.3 Commands

11.3.1 Compare checkpoints

Command: `/dedicps [record nr to compare with]`

Parameters:

- Optional record number to compare with, will compare with record nr 1 if none is given.

Functionality: Displays a list with checkpoint times of the record and your dedimania record showing the exact differences per checkpoint.

Required permission: None.

11.4 Signal handlers

11.4.1 Map begin

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

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

11.4.2 Map start

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

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

11.4.3 Map end

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

Functionality: Used to save dedimania records.

11.4.4 Player connect

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

Functionality: Displaying widget + sending dedimania request.

11.4.5 Player disconnect

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

Functionality: Sending dedimania request.

11.4.6 Player finish

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

Functionality: Registers new records.

CHAPTER

TWELVE

DYNAMIC POINTS

12.1 Information

Name: `pyplanet.apps.contrib.dynamic_points`

Depends on: `core.maniaplanet`

Game: ShootMania

12.2 Features

This app enables the dynamic points limit in Shootmania Royal. Setup with the //settings command!

12.3 Signal handlers

12.3.1 Map begin

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

Functionality: Apply the new limit if settings allow us to do.

12.3.2 Player connect

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

Functionality: Adjust the limit

12.3.3 Player disconnect

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

Functionality: Adjust the limit

12.3.4 Player info change

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

Functionality: Adjust the limit

CHAPTER

THIRTEEN

JUKEBOX

13.1 Information

Name: `pyplanet.apps.contrib.jukebox`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

13.2 Features

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

13.3 Commands

13.3.1 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.

13.3.2 Display jukebox list

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

Parameters: None.

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

Required permission: None.

13.3.3 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.

13.3.4 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.

13.4 Signal handlers

13.4.1 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

FOURTEEN

KARMA

14.1 Information

Name: `pyplanet.apps.contrib.karma`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

14.2 Features

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

14.3 Commands

14.3.1 Display votes

Command: `/whokarma`

Parameters: None.

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

Required permission: None.

14.4 Signal handlers

14.4.1 Map begin

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

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

14.4.2 Player chat

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

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

14.4.3 Player connect

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

Functionality: Displays the karma widget for the connecting player.

CHAPTER

FIFTEEN

LIVE RANKINGS

15.1 Information

Name: `pyplanet.apps.contrib.live_rankings`

Depends on: `core.maniaplanet`

Game: TrackMania

15.2 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).

15.3 Installation

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

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

15.4 Commands

-

15.5 Signal handlers

15.5.1 Map begin

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

Functionality: Clears rankings and widget

15.5.2 Player finish

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

Functionality: Process and update widget.

15.5.3 Player waypoint

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

Functionality: Process and update widget.

15.5.4 Player give up

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

Functionality: Set the time to DNF in specific modes.

15.5.5 Player connect

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

Functionality: Display widget.

15.5.6 Scores

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

Functionality: Update the widget with the driven scores.

CHAPTER

SIXTEEN

LOCAL RECORDS

16.1 Information

Name: `pyplanet.apps.contrib.local_records`

Depends on: `core.maniaplanet`

Game: TrackMania

16.2 Features

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

16.3 Commands

16.3.1 Display local records

Command: `/records`

Parameters: None.

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

Required permission: None.

16.3.2 Compare checkpoints

Command: `/localcps [record nr to compare with]`

Parameters:

- Optional record number to compare with, will compare with record nr 1 if none is given.

Functionality: Displays a list with checkpoint times of the record and your local record showing the exact differences per checkpoint.

Required permission: None.

16.4 Signal handlers

16.4.1 Map begin

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

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

16.4.2 Player connect

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

Functionality: Displays the records widget for the connecting player.

16.4.3 Player finish

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

Functionality: Registers new records.

CHAPTER

SEVENTEEN

MAP INFO

17.1 Information

Name: `pyplanet.apps.contrib.mapinfo`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

17.2 Features

Displays basic map information in widget.

17.3 Commands

None.

17.4 Signal handlers

17.4.1 Map begin

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

Functionality: Updates widget with new map information.

17.4.2 Player connect

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

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

CHAPTER

EIGHTEEN

MUSIC SERVER

18.1 Information

Name: `pyplanet.apps.contrib.music_server`

Depends on:

-

Game: All

18.2 Features

This app provides the ability to play your own music for all the players in the server.

Setup:

Add URLs to the music files you want to play your settings module (base.py) or directory (base.json / base.yaml) in the `SONGS = []` section. The files must be in the .ogg format for maniaplanet to be able to play them.

18.3 Commands

18.3.1 Display music list

Command: `/songlist` or `/musiclist`

Parameters: None.

Functionality: Displays the list of all available songs. Click songs to put them into the playlist.

Required permission: None.

18.3.2 Display Playlist

Command: `/playlist`

Parameters: None.

Functionality: Display the playlist. Click songs to drop them from the playlist. Users can only drop the songs the juked themselves.

Required permission: None.

18.3.3 Current Song

Command: `/song`

Parameters: None.

Functionality: Prints the Title and Artist of the song currently playing to the chat.

Required permission: None.

18.3.4 Play Song

Command: `//play`

Parameters: `songname` URL to music file to be played next.

Functionality: Puts the song into the songlist. It will be gone from it on next restart of PyPlanet.

Required permission: requires admin level 1

18.4 Signal handlers

18.4.1 Map End

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

Queue the next song.

CHAPTER

NINETEEN

MANIAEXCHANGE

19.1 Information

Name: `pyplanet.apps.contrib.mx`

Depends on: `core.maniaplanet`

Game: Any

19.2 Features

Adding maps from Mania-Exchange.

19.3 Commands

19.3.1 Add map(s) from MX

Command: `//add mx` or `//mx add`

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.

19.3.2 Search maps on MX

Command: `//mx search`

Parameters:

-

Functionality: Search/browse for maps on MX.

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

19.3.3 Add mappack from MX

Command: `//mxpack add`

Parameters:

- ManiaExchange Pack ID.

Functionality: Adding maps form a specific mappack on ManiaExchange to the server.

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

19.3.4 Search mappacks on MX

Command: `//mxpack search`

Parameters:

-

Functionality: Search/browse for mappacks on MX.

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

19.3.5 Get current map info

Command: `/mx info`

Parameters:

-

Functionality: Get information about the current map from the MX database.

Required permission:

-

CHAPTER

TWENTY

PLAYERS

20.1 Information

Name: `pyplanet.apps.contrib.players`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

20.2 Features

This app provides the playerlist UI.

20.3 Commands

20.3.1 Display playerlist

Command: `/players`

Parameters: None.

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

Required permission: None.

20.3.2 Show last online date of player

Command: `/laston` / `/lastseen`

Parameters:

- Login of the player.

Functionality: Display the last date and time the user has been seen on the server.

Required permission: None.

20.4 Signal handlers

None.

CHAPTER

TWENTYONE

WAITING QUEUE

21.1 Information

Name: `pyplanet.apps.contrib.queue`

Depends on: `core.maniaplanet`

Game: TrackMania or ShootMania

Mode: Any

21.2 Features

This app enables the waiting queue for crowded servers. Players should use the waiting queue on full servers and will be in a queue where the waiting is fair for all players.

> **Warning:** This app is new in 0.6.0 and is still in BETA. Unexpected behaviour can be expected, please post any issues to our GitHub project.

21.3 Commands

21.3.1 Show queue list

Command: `/queue`

Parameters:

-

Functionality: Get the list of the current queue.

Required permission:

-

21.3.2 Clear queue

Command: `//queue clear`

Parameters:

-

Functionality: Clear the queue (unqueue all spectators).

Required permission:

- queue:manage_queue (level 2 by default)

21.3.3 Shuffle queue

Command: `//queue shuffle`

Parameters:

-

Functionality: Shuffle the queue (randomly)

Required permission:

- queue:manage_queue (level 2 by default)

21.4 Signal handlers

21.4.1 Player Info Change

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

Functionality: Used to force the release of the player slot when going to spectator

21.4.2 Player enters player slot

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

Functionality: Update all views

21.4.3 Player enters spectator slot

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

Functionality: Update all views

21.4.4 Player connect

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

Functionality: When server is full or queue is filled, force to spectator and show message in the chat.

21.4.5 Player disconnect

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

Functionality: Remove player from queue if in, clear the data.

CHAPTER

TWENTYTWO

SECTOR TIMES

22.1 Information

Name: `pyplanet.apps.contrib.sector_times`

Depends on: `core.maniaplanet`

Game: TrackMania

22.2 Features

This app enables comparing the sector times against your best time driven ever (local or dedi record, or the current session best record). This widget is instant updating and using ManiaScript.

22.3 Signal handlers

22.3.1 Map begin

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

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

22.3.2 Player connect

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

Functionality: Displays the records widget for the connecting player.

CHAPTER

TWENTYTHREE

TRANSACTIONS

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

23.1 Information

Name: `pyplanet.apps.contrib.transactions`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

23.2 Features

Donate, show planets on server and payout players.

23.3 Commands

23.3.1 Donate

Command: `/donate`

Parameters:

- Amount of planets.

Functionality: Donate planets to the server.

Required permission:

-

23.3.2 Get amount of planets on server

Command: `//planets`

Parameters: None

Functionality: Get planet

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

23.3.3 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.

23.4 Signal handlers

23.4.1 Map begin

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

Functionality: Update bill signal

CHAPTER

TWENTYFOUR

VOTING

24.1 Information

Name: `pyplanet.apps.contrib.voting`

Depends on: `core.maniaplanet`

Game: TrackMania, ShootMania

24.2 Features

This app provides chat-based voting for your players.

24.3 Commands

24.3.1 Replay Vote

Command: `/replay`

Parameters: None.

Functionality: Initiate replay vote.

Required permission: None.

24.3.2 Skip Vote

Command: `/skip`

Parameters: None.

Functionality: Initiate skip vote.

Required permission: None.

24.3.3 Restart Vote

Command: `/restart`

Parameters: None.

Functionality: Initiate instant-restart vote.

Required permission: None.

24.3.4 Extend TimeAttack Time

Command: `/extend`

Parameters: None.

Functionality: Initiate time extend vote.

Required permission: None.

24.3.5 Vote Yes

Command: `/y`

Parameters: None.

Functionality: Vote yes, you can also use F5 to vote yes.

Required permission: None.

24.3.6 Vote No

Command: `/n`

Parameters: None.

Functionality: Vote no, you can also use F6 to vote no.

Required permission: None.

24.3.7 Cancel Vote

Command: `//cancel`

Parameters: None.

Functionality: Cancel current chat-based vote.

Required permission: `voting:cancel`, requires admin level 1.

24.4 Signal handlers

None.

CHAPTER

TWENTYFIVE

STATISTICS

25.1 Information

Name: `pyplanet.apps.core.statistics`

Depends on: `core.maniaplanet`

Game: TrackMania & ShootMania

25.2 Features

This app keeps track of the general statistics across the games.

25.3 Commands

25.3.1 Display Top Donators

Command: `/topdons`

Parameters: None.

Functionality: Display a list of the top donating players on the server.

Required permission: None.

25.3.2 Display Top Active

Command: `/topactive`

Parameters: None.

Functionality: Display a list of the top active players on the server.

Required permission: None.

25.3.3 Display Top Players (based on records) (TM only)

Command: `/topsums`

Parameters: None.

Functionality: Display a list of best players according to the top 3 records on maps.

Required permission: None.

25.3.4 Display personal score progression on map (TM only)

Command: `/scoreprogression`

Parameters: None.

Functionality: Display a list with past scores on the current map.

Required permission: None.

CHAPTER

TWENTYSIX

ARCHITECTURE & DESIGN

Contents

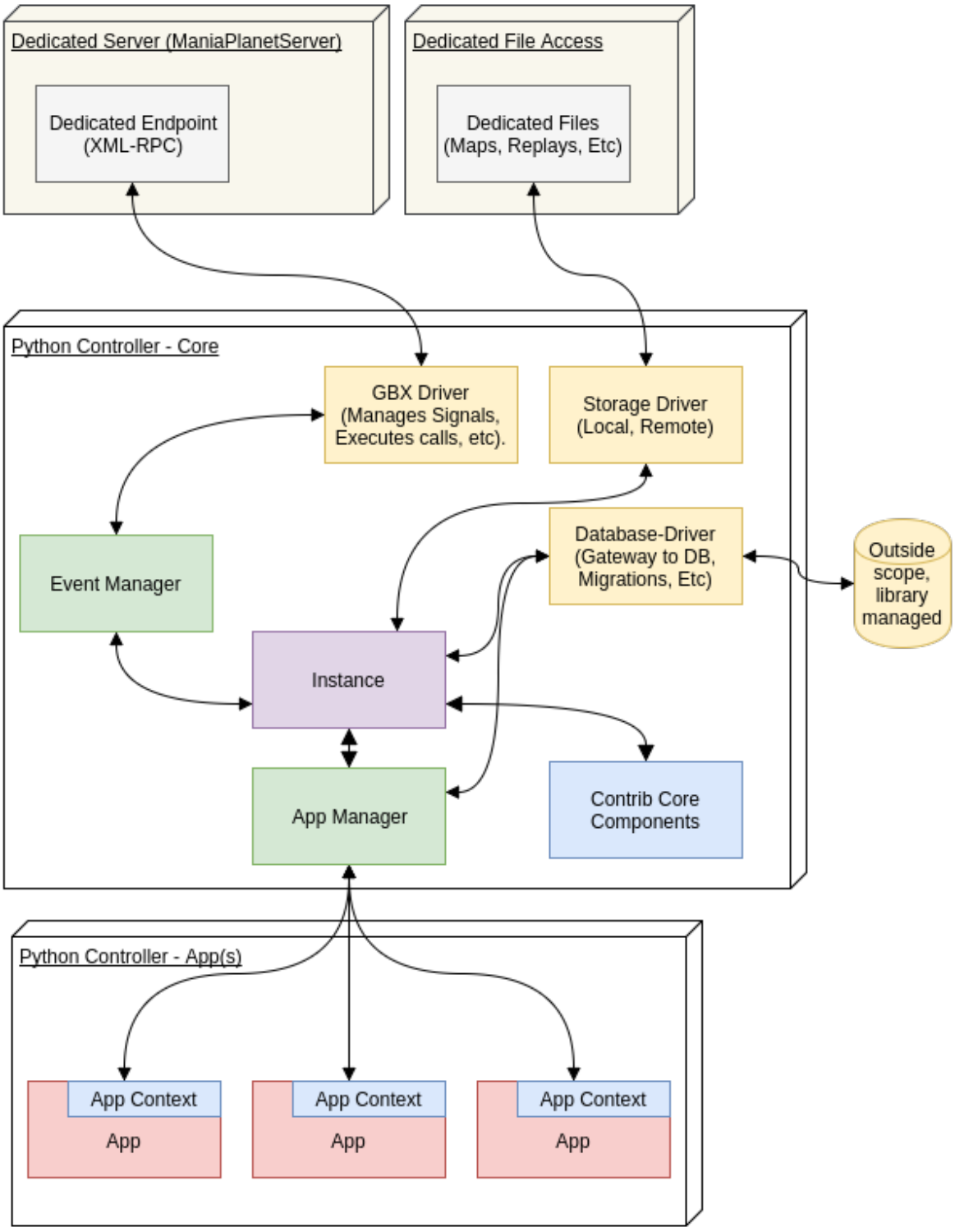
26.1 Core Architecture

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.

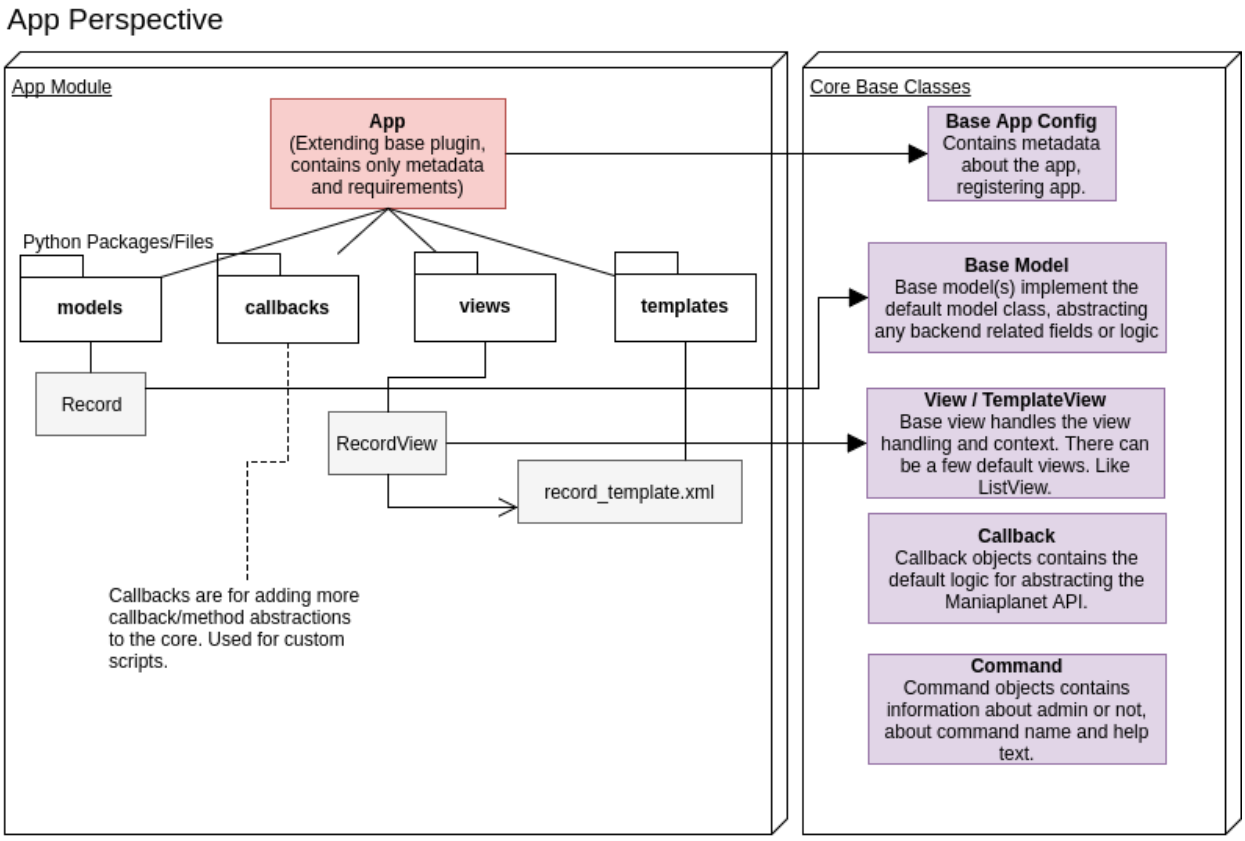
Global Overview

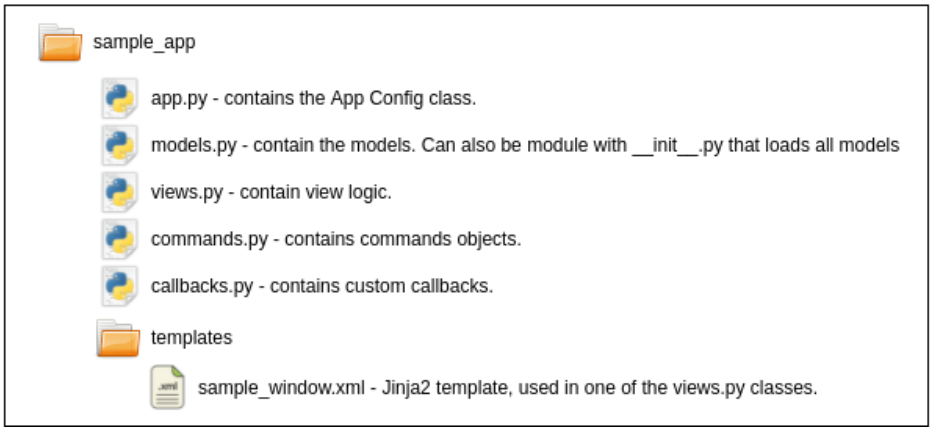


Note: This image is only describing the most important core components, some components are not shown here.

26.2 Apps Architecture

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





CHAPTER

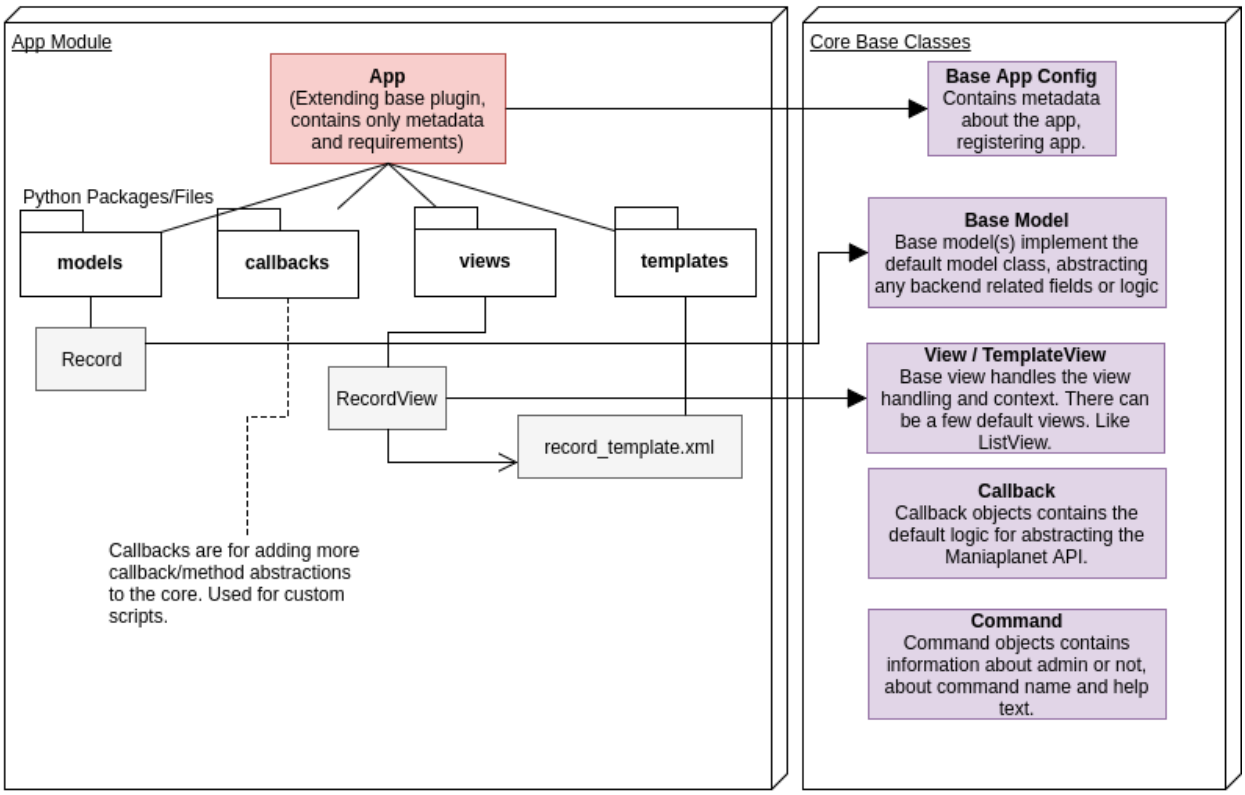
TWENTYSEVEN

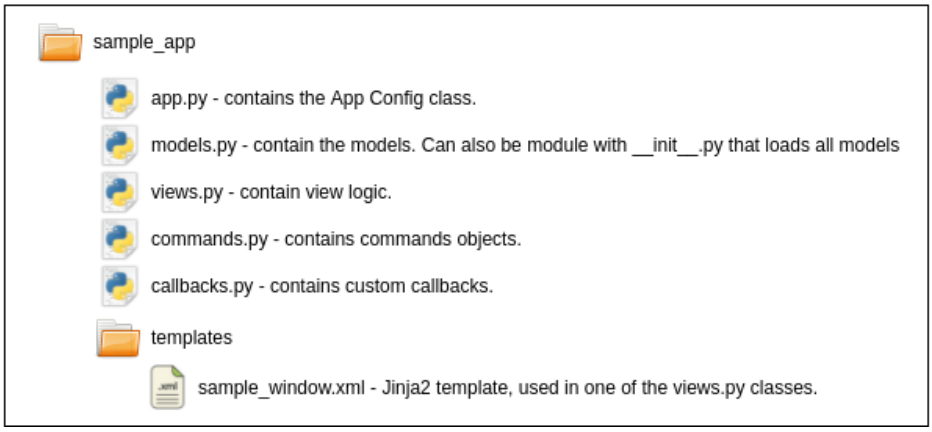
APP DEVELOPMENT

Contents

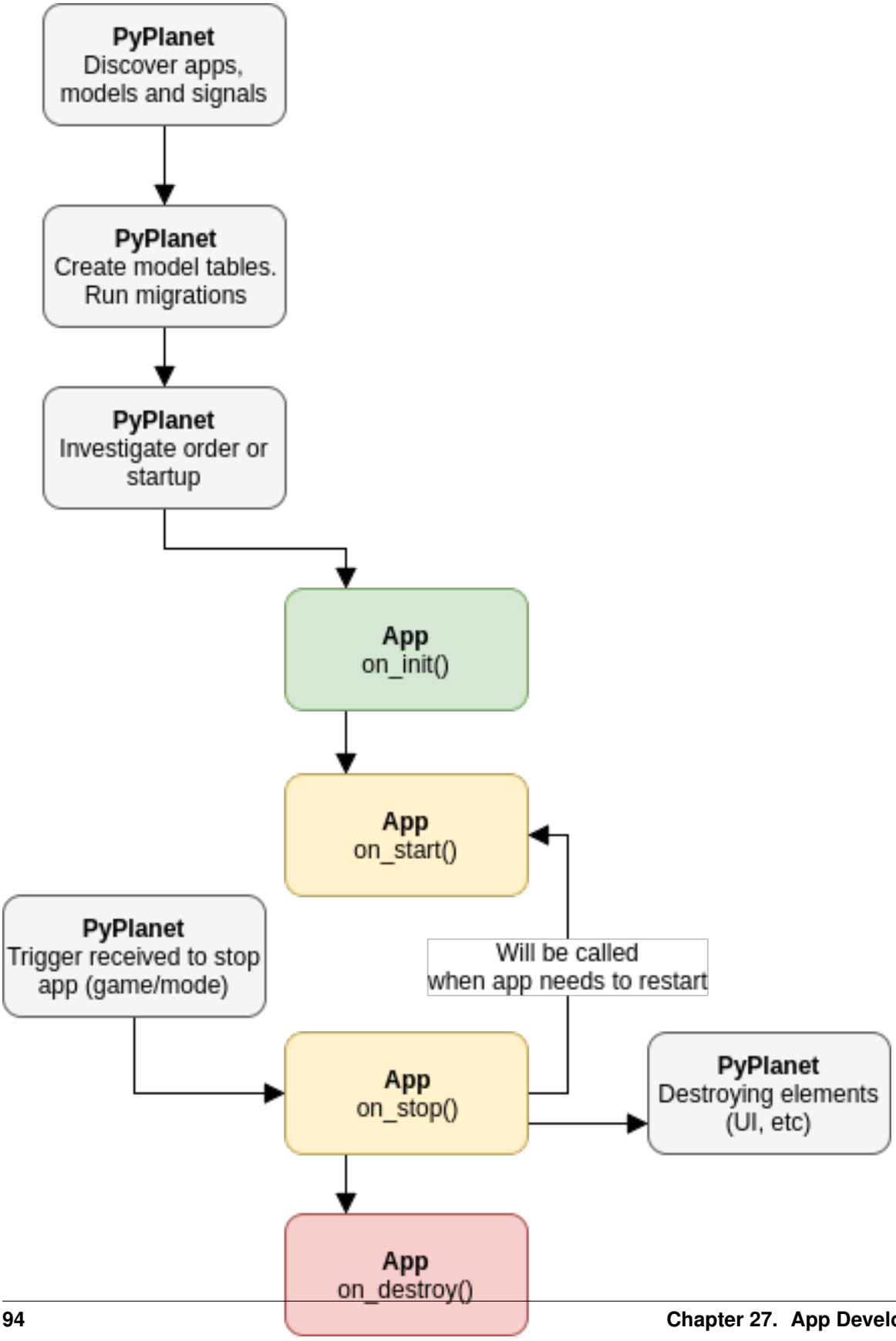
27.1 Apps Architecture

App Perspective





27.2 Life Cycle



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.

To support the life cycle in the future, use the `self.context.signals` instead of the `self.instance.signal_manager`

27.2.1 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.

27.2.2 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.

27.2.3 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.

27.2.4 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.

27.3 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`

27.3.1 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!
```

27.3.2 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.

27.3.3 3. Add to configuration

Make sure you add your new App to your configuration.

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

27.3.4 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
```

27.3.5 5. Start PyPlanet

Your ready to get started. Start PyPlanet!

27.4 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*.

27.5 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*

27.6 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.

27.6.1 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

27.6.2 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!

27.7 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.

27.7.1 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.

27.8 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'),
)
```

(continues on next page)

(continued from previous page)

```
# 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'),
)
```

27.9 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'),
)
```

27.10 User Interface

You are free to implement any User Interface features in your app yourself. You can use the template engine Jinja2 for getting values from the Python code inside of your XML that will be displayed to the client.

On this page you will find out how to implement a simple template and maniascript integration. As well as the useful manialink classes for hiding or showing for specific view styles.

27.10.1 Using templates

To use templates, use the *pyplanet.views.template.TemplateView* class (click on the class for the API docs). You can provide the class property *template_name* which should contain the exact template filename and path.

Example for the example_app:

```
class SampleView(TemplateView):
        template_name = 'example_app/test.xml'  # template should be in: ./example_
↪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``.
```

Providing data to the template can be done with several overriden methods in the class itself.

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

Async Method get_all_player_data(logins): Retrieve the player specific dictionary. Return dict with player as key and value should contain the data dict.

Async Method get_per_player_data(login): Retrieve the player specific dictionary per player. Return dict with the data dict for the specific login (player).

Make sure you visit the class documentation for all the methods on the TemplateView: *pyplanet.views.template.TemplateView*

Template Content

The actual XML you include with the *template_name* property is the file that get's loaded on rendering. The file can contain anything and can be enriched with the Jinja2 Template Language.

For the Jinja2 documentation we refer to the following page: http://jinja.pocoo.org/docs/2.10/

Example of a XML template with Jinja2 statements:

```
<frame pos="0 -40" id="sample_frame">
  {% if variable == 'value' %}
    <label pos="0 0" size="30 5" text="Variable contains value!" textsize="1.2"␣
↪valign="top" />
  {% else %}
    <label pos="0 0" size="30 5" text="Variable does not contain value!" textsize="1.2
↪" valign="top" />
  {% endif %}
</frame>
```

27.10.2 ManiaScript

Including ManiaScript to your ManiaLink template is pretty simple actually. Even including global libraries provided by the PyPlanet team is pretty easy. We will explain how you include ManiaScript in your ManiaLink template.

To include ManiaScript in your ManiaLink template, make sure you create a new file besides your ManiaLink template ending with *.Script.Txt* and add the following line to your ManiaLink (XML) template:

```
<script><!-- {% include 'my_app/sample.Script.Txt' %} --></script>
```

That's it! Now you can start with writing ManiaScript in the *sample.Script.Txt*. You can use Jinja2 inside your ManiaScript to add dynamic content as well.

To include libraries from PyPlanet inside of your ManiaScript, use the following in your *.Script.Txt* file:

```
// Includes
{% include 'core.views/libs/TimeUtils.Script.Txt' %}
```

> **Warning:** Remember, the core script utils can change behaviour at any time!

TimeUtils Lib

The TimeUtils contains several useful utils for working with times. The full path: `core.views/libs/TimeUtils.Script.Txt`.

Text LeftPad(Integer number, Integer pad)

This method will make sure the number is left-padded with the number of pads given.

`Text TimeToText(Integer inTime)`

This method will format time to text to show local or dedi records for example.

27.10.3 ManiaLink

Useful information about ManiaLink changes or additions made by PyPlanet. ManiaLink docs can be found here: https://doc.maniaplanet.com/manialink

27.11 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

TWENTYEIGHT

SIGNALS (CALLBACKS)

Contents

28.1 Maniaplanet

28.1.1 Flow

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

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.ca**

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.Ca**

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.callba**

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**

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.cal**

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.callba**

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.callb**

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.C**

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.callbacl**

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.Ca**

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.callba**

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**

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.cal**

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.**

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.callb**

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.callbac**

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.ca**

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.callba**

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.Ca**

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.callbac**

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.C**

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.callb**

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.ca**

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.**

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.

28.1.2 Map

pyplanet.apps.core.maniaplanet.callbacks.map.**map_begin = <pyplanet.core.events.callback.Ca**

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.Callb**

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.Ca**

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.callba**

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.cal**

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.

28.1.3 Player

pyplanet.apps.core.maniaplanet.callbacks.player.**player_chat = <pyplanet.core.events.callbac**

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.cal**

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.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_enter_player_slot = <pyplanet.core.e**

Signal Player enters a player slot.

Code `maniaplanet:player_enter_player_slot`

Description Player change into a player, is using a player slot.

Original Callback *None*

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

pyplanet.apps.core.maniaplanet.callbacks.player.**player_enter_spectator_slot = <pyplanet.co**

Signal Player enters a spectator slot (not temporary).

Code `maniaplanet:player_enter_spectator_slot`

Description Player change into a spectator, is using a spectator slot.

Original Callback *None*

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

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

Signal Player has changed status.

Code `maniaplanet:player_info_changed`

Description Callback sent when a player changes from state or information. The callback has been updated in 0.6.0 to include the information retrieved from extracting the flags parameter.

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).
- **force_spectator** (*int*) – 1, 2 or 3. Force spectator state
- **is_referee** – Is the player a referee.
- **is_podium_ready** – Is the player podium ready.
- **is_using_stereoscopy** – Is the player using stereoscopy
- **is_managed_by_other_server** – Is the player managed by another server (relaying).
- **is_server** – Is the player one of the servers.
- **has_player_slot** – Has the player a reserved player slot.

- **is_broadcasting** – Is the player broadcasting (steaming) via the in-game stream functionality.
- **has_joined_game** – Is the player ready and has it joined the game as player.

28.1.4 User Interface

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

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!

28.1.5 Other

pyplanet.apps.core.maniaplanet.callbacks.other.**bill_updated = <pyplanet.core.events.callbac**

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.eve**

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.e**

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.dispatch**

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.**server_password = <pyplanet.core.events.disp**

Signal Server player or spectator password changed

Code `maniaplanet:server_password`

Description Custom signal called when the password has been changed with PyPlanet.

Origin Callback None.

Parameters

- **password** (*str*) – String with the new password.
- **kind** (*str*) – Kind of password, could be 'player' or 'spectator'.

pyplanet.apps.core.maniaplanet.callbacks.other.**vote_updated = <pyplanet.core.events.callbac**

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.

28.2 Shootmania

28.2.1 Base

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

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

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**

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.callbac**

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.Ca**

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.Ca**

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_default = <pyplanet.core.events.callback.Ca**

Signal On Default Event

Code `shootmania:on_default`

Description Callback sent when a old event or default event has been fired.

Original Callback *Script* Shootmania.Event.Default

Parameters

- **time** – Time of server when callback is sent.
- **type** – Name of the command

pyplanet.apps.core.shootmania.callbacks.base.**on_fall_damage = <pyplanet.core.events.callbac**

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.Callba**

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**

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.Cal**

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**

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**

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.callbac**

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**

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**

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.c**

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**

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**

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.Callba**

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!!

28.2.2 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.Cal**

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.C**

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.

28.2.3 Joust

pyplanet.apps.core.shootmania.callbacks.joust.**player_reload = <pyplanet.core.events.callbac**

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.Cal**

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.cal**

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).

28.2.4 Royal

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

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.callbac**

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.Cal**

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.

28.3 Trackmania

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

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 o**

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 o**

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 ob**

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.Ca**

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.Callbac**

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 ob**

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.Callbac**

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.C**

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.Callb**

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**

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.Call**

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**

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

TWENTYNINE

API DOCUMENTATION

Modules:

29.1 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.

async check()

Check and remove unsupported apps based on the current game and script mode. Also loads unloaded apps and try if the mode and game does support it again.

async discover()

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

async init()

This method will initiate all apps in order and in series.

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?

async start()

This method will start all apps that are previously initiated.

async stop()

This method is executed when the instance is shutting down (will stop all the apps).

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
You can provide a list of dependencies to other apps (each entry needs to be a string of the app label!)

game_dependencies = None
You can provide a list of game dependencies that needs to meet when the app is started. For example you can provide:

```
game_dependencies = ['trackmania']
```

You can override this behaviour by defining the following method in your config class

```
def is_game_supported(self, game):
        return game != 'questmania'
```

human_name = None

static import_app(*entry*, *instance*)

is_game_supported(*game*)

is_mode_supported(*mode*)

label = None

mode_dependencies = None
You can provide a list of gamemodes that are required to activate the app. Gamemodes needs to be declared as script names. You can override this behaviour by defining the following method in your config class

```
def is_mode_supported(self, mode):
        return mode.lower().startswith('TimeAttack')
```

name = None

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

async 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.

async 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.

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

path = None

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*.

signals = None
Signal manager. See *Signal Manager*.

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

29.2 pyplanet.views

29.2.1 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*, *relaxed_updating=False*)

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

async 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!

async 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.

async 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** –

  –

async 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.

async 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*, *relaxed_updating=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_all_player_data(logins) Retrieve the player specific dictionary. Return dict with player as key and value should contain the data dict.

Method get_per_player_data(login) Retrieve the player specific dictionary per player. Return dict with the data dict for the specific login (player).

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
```

async 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!

async 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.

async get_all_player_data(*logins*)
Get all player data, should return dictionary with login as key, and dict as value.

Parameters **logins** – Login list of players. String list.

Returns Dictionary with data.

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

async get_per_player_data(*login*)
Get data for specific player. Will be called for all players that will render the xml for.

Parameters **login** (*str*) – Player login string.

Returns Dictionary or None to ignore.

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

Deprecated since version 0.4.0: Use *get_per_player_data()* and *get_all_player_data()* instead. Will be removed in 0.8.0!

async 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** –

  –

async 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.

async 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

29.2.2 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.

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

async wait_for_reaction()
Wait for reaction or input and return it.

Returns Returns the button clicked or the input value string of the user.

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)
```

__init__(*message*, *size='md'*, *buttons=None*, *manager=None*, *default=''*, *validator=None*)
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.

async wait_for_input()
Wait for input and return it.

Returns Returns the string value of the user.

async pyplanet.views.generics.alert.**ask_confirmation**(*player*, *message*, *size='md'*, *buttons=None*)
Ask the player for confirmation and return the button number (0 is first button).

Parameters

- **player** – Player login or instance.
- **message** – Message to display.
- **size** – Size, could be 'sm', 'md', or 'lg'.
- **buttons** – Buttons, optional, default is yes and no.

Returns Number of button that is clicked.

async pyplanet.views.generics.alert.**ask_input**(*player*, *message*, *size='md'*, *buttons=None*, *default=None*, *validator=None*)
Ask the player a question and prompt for input.

Parameters

- **player** – Player login or instance.
- **message** – Message to display.

- **size** – Size, could be 'sm', 'md', or 'lg'
- **buttons** – Buttons, optional, default is ok.
- **default** – The default and pre-filled value. Default empty.
- **validator** – Validator method, default is only checking if the input isn't empty.

Returns Input given by the user.

async pyplanet.views.generics.alert.**show_alert**(*player*, *message*, *size='md'*, *buttons=None*)

Show an alert to the player with given details. This is a shortcut method for the class itself.

Parameters

- **player** – Player login or instance.
- **message** – Message in string.
- **size** – Size, could be 'sm', 'md', or 'lg'.
- **buttons** – Buttons, optional, default is 'OK'.

Returns Number of the clicked button. (in Future).

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'
                },
        ]
```

(continues on next page)

(continued from previous page)

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

__init__(**args*, ***kwargs*)
Create manialink (USE THE MANAGER CREATE, DONT INIT DIRECTLY!

Parameters

- **manager** – Manager instance. use your app manager.
- **id** – Unique manialink id. Could be set later, must be set before displaying.
- **version** – Version of manialink.
- **body** – Body of manialink, not including manialink tags!!
- **template** – Template instance.
- **timeout** – Timeout to display, hide after the timeout is reached. Seconds.
- **hide_click** – Hide manialink when click is fired on button.
- **data** – Data to render. Could also be set later on or controlled separate from this instance.
- **player_data** – Dict with player login and for value the player specific variables. Dont fill this to have

a global manialink instead of per person. :param throw_exceptions: Throw exceptions during handling and executing of action handlers. :param relaxed_updating: Relaxed updating will rate limit the amount of updates send to clients. :type manager: pyplanet.core.ui.AppUIManager :type template: pyplanet.core.ui.template.Template :type id: str :type version: str :type timeout: int

async 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.

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

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

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

async 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** –

  –

async 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.

__init__(*data=None*, **args*, ***kwargs*)
Create manialink (USE THE MANAGER CREATE, DONT INIT DIRECTLY!

Parameters

- **manager** – Manager instance. use your app manager.
- **id** – Unique manialink id. Could be set later, must be set before displaying.
- **version** – Version of manialink.
- **body** – Body of manialink, not including manialink tags!!
- **template** – Template instance.
- **timeout** – Timeout to display, hide after the timeout is reached. Seconds.
- **hide_click** – Hide manialink when click is fired on button.
- **data** – Data to render. Could also be set later on or controlled separate from this instance.
- **player_data** – Dict with player login and for value the player specific variables. Dont fill this to have

a global manialink instead of per person. :param throw_exceptions: Throw exceptions during handling and executing of action handlers. :param relaxed_updating: Relaxed updating will rate limit the amount of updates send to clients. :type manager: pyplanet.core.ui.AppUIManager :type template: pyplanet.core.ui.template.Template :type id: str :type version: str :type timeout: int

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

29.3 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.

29.4 pyplanet.core.instance

PyPlanet Instance Module

This module holds the main instance class of the PyPlanet system.

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.
- ***signals*** – Signal Manager (global). Please use the APP context Signal Manager instead!
- **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.

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

Returns Performance mode boolean.

property signal_manager
Deprecated!

Deprecated since version 0.5.0: Use `self.context.signals()` instead in your apps.

Returns Signal manager (global).

Return type *pyplanet.core.events.manager._SignalManager*

start(*run_forever=True*)
Start wrapper.

stop()
Stop all the instance apps and managers.

29.5 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`.

class pyplanet.core.ui.**AppUIManager**(*instance*, *app*)
The App UI manager is here to maintain the context of the app and have it destroy all the listeners when the app is unloaded.

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).

async reset()
Reset the UI Properties to the default ManiaPlanet ones. :return:

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*, *relaxed_updating=False*)
The StaticManiaLink is mostly used in PyPlanet for general views. Please use the `View` classes instead of this core ui component!

async 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!

async 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.

async 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** –

  –

async 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.

async 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.

async 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!

async 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.

async 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** –

  –

async 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.

async 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.

29.6 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!

property 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.

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

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

29.6.1 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

29.7 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 in the app via `self.context.signals`.

create_app_manager(*app*)

This method will create the manager instance for the app context.

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

Returns SignalManager instance for the app.

Return type pyplanet.core.events.manager.AppSignalManager

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

async 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).

29.7.1 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!

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

async 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.

29.7.2 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

async 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.

async 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), ...].

async 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

29.8 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*, *options=None*)
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*, *options=None*)
The InstanceProcess is the encapsulation around the real controller instance.

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

property did_die
Boolean determinating if the process did die.

property exitcode
Exit code of process.

Returns Exit code.

graceful()
Graceful shutdown the process.

is_alive()
Call process method is_alive()

shutdown()
Shutdown (terminate) process.

start()
Start the process.

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

29.9 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.

async add_map(*filename*, *insert=True*, *save_matchsettings=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.
- **save_matchsettings** (*bool*) – Save match settings as well.

Raise pyplanet.contrib.map.exceptions.MapIncompatible

Raise pyplanet.contrib.map.exceptions.MapException

property current_map
The current map, database model instance.

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

async extend_ta(*extend_with=None*)
Extend time limit of the current map. Extend with given seconds, or double the original TA timer if None is given.

Parameters **extend_with** (*int*) – Extend with the given seconds, or None for adding the original TA limit to the current limit(double)

Returns

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

Parameters **uid** – By uid (pk).

Returns Player or exception if not found

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

Parameters **index** – Primary key index.

Returns Map instance or raise exception.

async 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

async load_matchsettings(*filename*)
Load Match Settings file and insert it into the current map playlist.

Parameters **filename** – File to load, relative to Maps folder.

Returns Boolean if loaded.

property 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

property 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).

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

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

async 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

async 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

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

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

async 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.

async 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

exception pyplanet.contrib.map.exceptions.**ModeIncompatible**
The current mode doesn't support the given action.

29.10 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.

You can access this class in your app with:

```
self.instance.player_manager
```

With the manager you can get several useful information about the players on the server. See all the properties and methods below for more information.

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

property count_all
Get all player counts (players + spectators).

property count_players
Get number of playing players.

property count_spectators
Get number of spectating players.

async 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

async get_player_by_id(*identifier*)
Get player object by ID.

Parameters **identifier** – Identifier.

Returns Player object or None

async 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

async 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

async load_blacklist(*filename=None*)
Load blacklist file.

Parameters **filename** – File to load or will get from settings.

Raise pyplanet.core.exceptions.ImproperlyConfigured

Raise pyplanet.core.storage.exceptions.StorageException

Returns Boolean if loaded.

async map_loaded(**args*, ***kwargs*)
Reindex the current number of players and spectators.

Parameters

- **args** –
- **kwargs** –

Returns

property max_players
Get maximum number of players.

property max_spectators
Get maximum number of spectators.

async 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!

property online
Online player list.

property online_logins
Online player logins list.

async save_blacklist(*filename=None*)
Save the current blacklisted players to file given or fetch from config.

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

Raise pyplanet.core.exceptions.ImproperlyConfigured

Raise pyplanet.core.storage.exceptions.StorageException

Exceptions for Map Manager.

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

29.11 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.

You should access this class within your app like this:

```
self.instance.command_manager
```

You can register your commands like this:

```
await self.instance.command_manager.register(
        Command(command='reboot', target=self.reboot_pool, perms='admin:reboot',␣
↪admin=True),
)
```

More information of the command and the options of it, see the *pyplanet.contrib.command.Command* class.

Warning: Don't initiate this class yourself. Access this class from the `self.instance.command_manager` instance.

async execute(*player*, *command*, **args*)
Execute a command for the given player with the given args.

Parameters

- **player** (*pyplanet.apps.core.maniaplanet.models.player.Player*) – Player instance.
- **command** (*pyplanet.contrib.command.command.Command*) – Command instance.
- **args** – Args for the command, will be concat into a string with spaces.

Returns

async 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.

Some examples of some commands:

```
# Admin command with permission on it.
Command(command='reboot', target=self.reboot_pool, perms='admin:reboot',␣
↪admin=True)

# Normal user command with optional argument.
Command(command='list', target=self.show_map_list)                    .add_
↪param(name='search', required=False)
```

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

async 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.

property 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*

property 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!

exception pyplanet.contrib.command.exceptions.**ParamException**

exception pyplanet.contrib.command.exceptions.**ParamParseException**

exception pyplanet.contrib.command.exceptions.**ParamValidateException**

29.12 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.

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

Parameters

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

async 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.

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

async 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.

29.13 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.

async 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.

async 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

async 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*

async 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*

async 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.

async 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.

async 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

property 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.

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

async 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

async 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

async 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

async 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

async 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

property 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.

__init__(*instance*)
Initiate, should only be done from the core instance.

Parameters **instance** (pyplanet.core.instance.Instance) – Instance.

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*

async 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*

async 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.

async 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.

async 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

property 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.

29.14 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.

async get_current_full_script(*refresh=False*)
Get the current full script name.

Parameters **refresh** – Refresh from server.

async get_current_script(*refresh=False*)
Get the current script name.

Parameters **refresh** – Refresh from server.

async get_current_script_info()
Get the script info as a structure containing: Name, CompatibleTypes, Description, Version and the settings available.

async get_next_full_script(*refresh=False*)
Get the next full script name.

Parameters **refresh** – Refresh from server.

async get_next_script(*refresh=False*)
Get the next script name.

Parameters **refresh** – Refresh from server.

async get_settings()
Get the current mode settings as a dictionary.

async get_variables()
Get the mode script variables.

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

async set_next_script(*name*)
Set the next played script name (after map restart/skip).

Parameters **name** – Name

async 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.

async 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.

async 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.

async 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.

29.14.1 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**
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.

29.15 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.

class pyplanet.contrib.converter.uaseco.**UasecoConverter**(**args*, ***kwargs*)
The UAseco 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.

29.16 pyplanet.contrib.chat

29.16.1 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'),
)
```

29.16.2 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.

async 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

29.17 pyplanet.utils

29.17.1 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.

async seek(*offset*)
We need to override the second param to move from the current position.

Parameters **offset** (*int*) – offset to move away.

29.17.2 pyplanet.utils.style

pyplanet.utils.style.**STRIP_ALL = {'letters': 'wnoitsgz<>', 'part': '\\$[lh]\\[.+\\]|\\$[**
Strip all custom maniaplanet styles + formatting.

pyplanet.utils.style.**STRIP_CAPITALS = {'letters': 't'}**
Strip capital style ($t).

pyplanet.utils.style.**STRIP_COLORS = {'letters': 'g', 'part': '\\$[0-9a-f]{3}'}**
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))
```

(continues on next page)

(continued from previous page)

```
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

29.17.3 pyplanet.utils.times

pyplanet.utils.times.**format_time**(*time*, *hide_hours_when_zero=True*, *hide_milliseconds=False*)

Format time from integer milliseconds to string format that could be displayed to the end-user.

Parameters

- **time** (*int*) – Integer time in milliseconds.
- **hide_hours_when_zero** (*bool*) – Hide the hours when there are zero hours.
- **hide_milliseconds** (*bool*) – Hide the milliseconds.

Returns String output

Return type str

CHAPTER

THIRTY

SUPPORT & 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/

30.1 Demo Servers

There are several demo servers available. You need to search for servers on the following logins:

- toffestaging1
- toffestaging2

30.2 Who is behind PyPlanet

Organisation:

- Toffe: Project and organisation lead.

Core:

- Toffe: Lead developer of the PyPlanet project.

Apps:

- Toffe: Bundled Application developer.
- TheM: Bundled Application developer.

Want to help us? Contact Toffe on Discord or Forum: `Toffe#8999` or Forum Profile.

CHAPTER

THIRTYONE

DONATE

You can support PyPlanet and other projects of Toffe (such as ManiaCDN and the Toolkit application) with the following donation options:

CHAPTER

THIRTYTWO

PRIVACY

32.1 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`.

32.2 Analytics & Telemetry

For future improvements and look into the usage of PyPlanet and it's apps we collect the following information:

- PyPlanet version
- Python version
- Server version
- Operating system
- Server login
- Server titlepack
- Active apps
- Total number of players

We do this by sending so called ping-updates every hour with up-to-date status about the server. By collecting this we gain information on how to improve with targeting specific titles or apps for updates. And to improve the operating system support if required.

You can turn this off by adding this line to your `settings/base.py`:

```
ANALYTICS = False
```

CHAPTER

THIRTYTHREE

CHANGELOG

33.1 0.7.4 (04 March 2020)

33.1.1 Apps

- Bugfix: Fixing issue with the MX update dialog and it's internal logic.

33.2 0.7.3 (02 March 2020)

33.2.1 Core

- Bugfix: Make sure the libraries also work for older Python versions (3.5.x).

33.3 0.7.2 (02 March 2020)

33.3.1 Core

- Improvement: Python 3.8.x support!
- Improvement: Update libraries used.
- Improvement: Better error handling for loading configuration/settings files.
- Bugfix: Make sure the MX-id is properly extracted and inserted into the database.

33.3.2 Apps

- Feature: Add MX map update window. Access it with //mx status. You can update your maps when there are any available updates.
- Improvement: Add dedimania link to the dedimania page in the chat message and the record list.
- Improvement: Add alias for the command /mapfolders: /mf.
- Improvement: Add alias for the MX search: //mx list and //mxpack list.
- Improvement: Improve the error messages from a failing Dedimania service.
- Bugfix: Make sure the queue app is inactive when the server is password protected.

- Bugfix: Make sure admins can't kick/ban/blacklist admins at the same level or higher.

33.4 0.7.1 (23 October 2019)

33.4.1 Core

- Bugfix: External map changes are detected wrongly resulting in performance impact in map change on large servers. This issue has been resolved.

33.5 0.7.0 (05 October 2019)

33.5.1 Core

- **Breaking**: Removed the deprecated `app.mapinfo`.
- Feature: Keeping track of the MX-id in the database (Database Migration is executed at first startup, no action required for this).
- Feature: Keep track of the total donations and total playtime of the players. Show it with `/topactive` and `/topdons`.
- Improvement: Upgrade several external libraries.
- Improvement: Support for the latest XMLRPC Scripted version and latest dedicated version. (Min. dedicated is now set to 2018-02-09_16_00).
- Improvement: Improve the cleanup and initial reset of the UI Properties.
- Improvement: Changed the key to show/hide some widgets from F7 to F8.
- Improvement: Added one missing scripted event handler for Shootmania.
- Improvement: Update the maplist when a change is detected by the server (useful when adding/removing maps in another tool).
- Security: Update some libraries to fix some security issues (none of which were critical).
- Bugfix: When a map is removed it previously didn't always got removed from the /list view, this has been fixed.

33.5.2 Apps

- New App: Integrated the Current CPS App from Teemann into the bundled apps (will get a refactor later on).
- Feature: Add MX Info command `/mx info`.
- Feature: Add command to show/hide the admin toolbar `//toolbar`.
- Feature: Add a setting to disable/enable juking maps by players.
- Feature: Add voting widget (displaying buttons when a vote is ongoing).
- Feature: Add support for MX MapPacks. `//mxpack search` and `//mxpack add [id]`.
- Feature: Add a setting to decide how many days a map should be classified as 'new' and be included in the mapfolder 'new maps'.
- Feature: Added a warn button to the manage players view (`//players`).

- Feature: Add a timeout to the chatvotes, the timeout is an adjustable setting. (default 120 seconds).
- Improvement: The dedimania welcome message also contains the limits of the player and server according to their donation status. (This is a setting and can be turned on, off by default!)
- Improvement: Small improvements in the map karma app related to usability and chat feedback.

33.6 0.6.4 (17 February 2019)

33.6.1 Core

- Improvement: Upgrade several external libraries.
- Improvement: Fix English grammar mistake.
- Security: Make sure that the Yaml files are loaded with the safe method.
- Bugfix: Fixing the integer overflow when extending the time limit too much (for TA modes).
- Bugfix: Make sure to await the coroutine in the royal points callback.

33.6.2 Apps

- Improvement: Make sure the user can use the localcps and dedicps when not having an record (just to view the checkpoint times).

33.7 0.6.3 (17 November 2018)

33.7.1 Core

- Bugfix: Fixing loading of settings on some setups.

33.8 0.6.2 (17 November 2018)

33.8.1 Core

- Security: Upgraded library to solve security issues (requests library).
- Bugfix: Fixing issues with the command line interface and showing settings error, preventing executing commands outside project

33.8.2 Apps

- Bugfix: Fix issue with clearing the jukebox and locking up the whole jukebox app.

33.9 0.6.1 (7 October 2018)

33.9.1 Core

- Improvement: Added compatibility with Python 3.7.x.
- Improvement: Upgraded external libraries.
- Improvement: Giant performance improvement when indexing maps, karma and local-records data after writing maplist and booting for large servers.
- Bugfix: Fixing issue with invalid JSON files (settings). Will show a correct error message.
- Bugfix: Fixing readmaplist.

33.9.2 Apps

- Bugfix: Fix issue in Local Records. Trying to initiate widget before the widget is created in the context.
- Bugfix: Fixing incorrect differences on the live cp times (live rankings) in laps mode.
- Bugfix: Fixing issues with Dedimania in Laps mode.
- Bugfix: Fixing issues with cleaning the Dedimania replays.
- Bugfix: Fixing issue with Dedimania and first driven record (global while it should be only to the person).
- Bugfix: Fixing issue with recording of normal and expanded karma scores in karma app.

33.10 0.6.0 (5 May 2018)

33.10.1 Core

- **Breaking**: Removed the deprecated `app.ui`.
- Feature: Add in-game and command line upgrade commands (//upgrade and ./manage.py upgrade) (CAUTION: Can be unstable!).
- Improvement: Slightly improved the performance when booting PyPlanet on large servers (indexing of local and karma)
- Improvement: Increased the retry count for connecting to a dedicated server from 5 to 10 retries.
- Improvement: Added bumpversion to project (technical and only for development).
- Improvement: Unpack the flags of the `PlayerInfoChange` callback and expand the flow variables (technical).
- Improvement: Updated external libraries.
- Improvement: Extract the zone information for players (technical).
- Improvement: Add nation to join and leave messages.

- Improvement: Activated the shutdown handlers to safely exit PyPlanet. The stop callbacks are now called at shutdown of PyPlanet.
- Improvement: Show pre-release as update when running on a pre-release version. (We now release pre-releases for public testing).
- Bugfix: Fix issue when trying to //reboot on Windows.

33.10.2 Apps

- **NEW: Add Music Server App: Queue music on your server. Add `pyplanet.apps.contrib.music_server` to your**
  More information: http://www.pypla.net/en/latest/apps/contrib/music_server.html
- **NEW: Add Advertisement App: Show Discord and PayPal logos in-game. Add `pyplanet.apps.contrib.ads` to you**
  More information: http://www.pypla.net/en/latest/apps/contrib/ads.html
- **NEW: Add Queue App: Add a queue for your spectators to fairly join on busy servers. Add `pyplanet.apps.contrib`**
  More information: http://www.pypla.net/en/latest/apps/contrib/queue.html
- Feature: Add settings to change vote ratio for the chat voting app.
- Feature: Add advanced voting (++, +, +-, -, –).
- Feature: Add MX Karma integration. You can configure this in-game with //settings and retrieve a key from: https://karma.mania-exchange.com/
- Feature: Add Admin Toolbar to manage your server a bit faster. (you can disable this in //settings)
- Feature: Add new vote to extend the time limit on TA modes (better than /replay or /restart, try it!).
- Feature: Add admin command to extend the time limit on TA modes temporary (//extend [time to extend with] or empty for double the current limit).
- Feature: Add dedimania checkpoint comparison (/dedicps and /dedicps [record number]) to compare your checkpoint times with the record given (or first when none given).
- Feature: Add local record checkpoint comparison (/localcps and /localcps [record number]) to compare your checkpoint times with the record given (or first when none given).
- Feature: Add F7 to hide most of the widgets (concentration mode).
- Feature: Add /topsums statistics to see the top local record players.
- Feature: Add buttons to delete local records by an admin.
- Feature: Add checkpoint difference in the middle of the screen when passing checkpoints (in the sector_times app).
- Feature: Cleanup the dedimania ghost files after reading and sending to dedimania API.
- Feature: Add advanced /list for searching and sorting with your personal local record, the time difference and karma. (can take long on big servers).
- Improvement: Add caching to the /list view per player and per view.
- Bugfix: Fix issue with incorrect link in the dedimania settings entry.
- Bugfix: Fix the type inconsistency of the dedimania API and driven records
- Bugfix: Fix when trying to vote after restarting the map in the podium sequence.
- Bugfix: Fix the retry logic of Dedimania when losing connection.

33.11 0.5.4

33.11.1 Core

- Improvement: Add unit testing on Windows platform (Technically, using AppVeyor).
- Bugfix: Make sure script names with folders are cleaned and stripped from folder names in most cases.

33.11.2 Apps

- Feature: Add button and window to change a folder's name.
- Improvement: Juke maps that are just added the correct order.
- Improvement: Allow the best CP widget for all modes.
- Improvement: Add blacklist write and read commands, now writes when adding player to blacklist and reads when PyPlanet starts.
- Bugfix: Fix the scoreprogression command and window.
- Bugfix: Fix issue when map list was saved to disk and all auto-folders where empty afterwards.
- Bugfix: Fix issue where the dedimania records where not reloaded when game mode changed and map has been restarted.
- Bugfix: Fix message when 2 players rapidly vote and the vote has passed.

33.12 0.5.3

33.12.1 Apps

- Bugfix: Fixing issue with spamming chat vote reminder.
- Bugfix: Fixing admin pass message when forcing pass a vote.

33.13 0.5.2

33.13.1 Core

- Improvement: Disable writing log files by default from 0.5.2.
- Improvement: Move logo and clock down so it doesn't interfere with the spectator icon.
- Bugfix: Logging on windows should be fixed now.
- Bugfix: Issue with multiple users editting modesettings or PyPlanet settings at the same time.

33.13.2 Apps

- Feature: Add zero karma folder (auto-folder)
- Feature: Added settings to enable or disable specific chat votes.
- Feature: Add //cancelcall (//cancelcallvote) for cancelling a call vote as an admin.
- Feature: Add //pass to pass a chat vote with your admin powers.
- Feature: Add button to add current map to folder on the folder list.
- Improvement: Change chat color of the chat vote lines.
- Improvement: Disable callvotes when chatvotes is turned on (made setting for this as well).
- Bugfix: Only show the folders of the user when adding maps to a folder.
- Bugfix: Fix error when player has not been online and users trying to get the last on date of the player.
- Bugfix: Remove unique index on the folder name so folders can have the same name over all. (auto-migration made).
- Bugfix: Fix bug that prevented added maps to be auto-juked.

33.14 0.5.1

33.14.1 Core

- Bugfix: Fix for Windows users and import error.

33.15 0.5.0

33.15.1 Core

- **Breaking**: App context aware signal manager.

  This is a *deprecation* for the property `signal_manager` of the `instance`. This means that `self.instance.signal_manager` needs to be replaced by `self.context.signals` to work with the life cycle changes in 0.8.0. More info: https://github.com/PyPlanet/PyPlanet/issues/392

  The old way will break your app from version 0.8.0

- Feature: Add multiple configuration backends. You can now use JSON or YAML as configuration as well. This is in a beta stage and can still change in upcoming versions. See the documentation for usage.
- Feature: Add logging to file option for starting PyPlanet. You can set this up inside of your settings *base.py*. More information can be found in the documentation for configuring PyPlanet.
- Feature: Add detach switch to the PyPlanet starter so it can fork itself to the background and write a PID file. More information can be found in the documentation for starting PyPlanet.
- Feature: Add player attributes that can be set by apps for caching or maintaining user settings or data during the session. (Technical)
- Feature: Add migration script for eXpansion database. Look at the manual on http://www.pypla.net/en/stable/convert/index.html for more information.

- Improvement: Retry 5 times when connecting to the dedicated server, making it possible to start both at the same time.
- Improvement: Update library versions.
- Improvement: Add minimum required version of the dedicated server to prevent starting PyPlanet for non-supported dedicated versions.
- Improvement: Only check for stable new versions. Now check for releases instead of tags on Github.
- Improvement: Let the list view skip 10 pages buttons skip to end or begin when less than 10 pages difference. (Thanks @froznsm)
- Improvement: Add online players login list in the player_manager. (Technical)
- Bugfix: Fixing issue with the release checker.
- Bugfix: Fixing the link to the upgrade documentation page (Thanks to @thefifthisa).
- Bugfix: Only handle player info change event when this player is still on the server to prevent errors.
- Bugfix: Handle exception when the server initiated a callvote (Thanks to @teemann).
- Bugfix: Correctly handle None column values when searching and/or sorting generic lists.
- Bugfix: Correctly handle non-string column values when searching and/or sorting generic lists.
- Bugfix: Refresh and fixed the player and spectator counters.

33.15.2 Apps

- NEW: Best CPS Widget for Trackmania, shows the best times per checkpoint above the screen. Add the new app to your apps.py: *'pyplanet.apps.contrib.best_cps'*. More info on the documentation pages of the app. (Big thanks to @froznsm)
- NEW: Clock Widget, shows the local time of the players computer on the PyPlanet logo. Add the new app to your apps.py: *'pyplanet.apps.contrib.clock'*. More info on the documentation pages of the app. (Big thanks to @froznsm)
- NEW: Chat-based Vote App, want to have votes in the chat instead of the callvotes? Enable this app now! Add the new app to your apps.py: *'pyplanet.apps.contrib.voting'*. More info on the documentation pages of the app.
- Feature: Add folders to the /list interface. There are two types of folders, automatic folders based on facts and manual per player/admin folders.
- Feature: Add folders for karma related information when karma app is enabled.
- Feature: Add folder for newest maps (added within 14 days).
- Feature: Add spectator status in the /players list.
- Feature: Add /scoreprogression command to see your current score progressions statistics on the current track.
- Feature: Add team switch commands (//forceteam and //switchteam) to the admin app.
- Feature: Add warning command (//warn) and alert to the admin app to warn players.
- Feature: Add the MX link of the current map to the logo left from the map name.
- Feature: Add setting to directly juke after adding map from MX or local (defaults to on).
- Feature: Add //blacklist and //unblacklist to the admin app.
- Improvement: Applied context aware signal manager everywhere.
- Improvement: Moving logic to view in dedimania app.

- Improvement: Adding setting to juke map after //add (mx and local) the map. Enabled by default!
- Improvement: Adding help text to jukebox app command.
- Improvement: Remove workaround for the fixed dedicated issue caused problems with the dedimania app.
- Improvement: Only show login in /list for now as it was causing inconsistency.
- Improvement: Check if the player is online before taking admin actions like kicking the player.
- Improvement: Refactor logic of viewing dedimania records to the desired view class. (Technical)
- Improvement: Further investigate dedimania problems for some specific players. Internal cause is known, exact reason not yet, we will further investigate this issue.
- Bugfix: Make sure to skip jukeboxed map when it's deleted from the server.
- Bugfix: Fix the double live rankings entry when changing nickname.
- Bugfix: Check if we have data to compare before calculating CP difference in the live rankings widget.
- Bugfix: Local record widget display fix when player joined during a very specific time that causes it to not display to the user.

33.16 0.4.5

33.16.1 Core

- Feature: Add ManiaControl convert script. See documentation on converting from old controller for instructions.
- Improved: Add documentation on how to convert to the right database collation.

33.16.2 Apps

- Bugfix: Fixing issue in the Dymanic Pointlimit app that results in 3 settings having the same key name.

33.17 0.4.4

- Feature: Add UAseco convert script. See documentation on converting from old controller for instructions.
- Improved: Updated libraries and dependencies.
- Bugfix: Catch error when server initiated callvote, thanks to @teemann.
- Bugfix: Fix the release/update checker.

33.18 0.4.3

33.18.1 Apps

- Bugfix: Fix issue with switching to custom script (lower case not found), specially teams mode.

33.19 0.4.2

33.19.1 Core

- Improvement: Bump XML-RPC Script API to version 2.2.0.
- Improvement: Show the Round Score build-in ui (nadeo widget) and move it a bit.
- Improvement: Move the build-in warmup ui (nadeo widget) a bit.

33.19.2 Apps

- Feature: Add //shuffle and //readmaplist. Both are unsure to work.
- Improvement: Further investigate and report issues related to Dedimania.
- Bugfix: Fixing negative count issue on the info widgets.
- Bugfix: Remove faulty and debug line from dedimania api catch block.
- Bugfix: Properly handle the dedimania response when player is not correct.
- Bugfix: Fixing issues with boolean values and the //modesettings GUI.

33.20 0.4.1

33.20.1 Core

- Improvement: Add command ignore and /version improvements.
- Improvement: Disable the live infos in the left upper corner (player join/leave, 1st finish).
- Bugfix: Issue with database collate and utf8mb4, nickname parsing issue has been solved.
- Bugfix: Don't auto reload and use different environments for the template engine. Should improve performance very much.
- Bugfix: Ignore unknown login at the chat and UI managers.
- Bugfix: Ignore key interrupt exception trace when stopping PyPlanet while it has got a reboot in the mean time.
- Bugfix: Hide the ALT menu in shootmania, just as it should do since before 0.4.0.
- Bugfix: Fixing issue with checking for updates could result in a exception trace in the console for some installations with older setuptools.
- Bugfix: Fixing an issue that results in fetching data for widget several times while it's not needed (thinking it's per player data when it isn't). (Thanks to Chris92)

33.20.2 Apps

- Improvement: Make it able to drive dedimania records on short maps made by Nadeo.
- Improvement: Make the improvement time blue like Nadeo also does in the sector times widget.
- Improvement: Always show nickname of the map author and make it switchable by clicking on it.
- Bugfix: Don't set the time of the spectator as your best time in the sector times widget.
- Bugfix: Problems that could lead to dedimania not being init currently on the map if the map was replayed.
- Bugfix: Hide dedimania if map is not supported.
- Bugfix: Fix the offset issue for the live rankings widget (in TA mode).
- Bugfix: Fix the incorrect number of spec/player count on the top left info widget.

33.21 0.4.0

33.21.1 Core

- **Breaking**: Refactored the TemplateView to make it able to use player data way more efficient.

  This is a *deprecation* for the method `get_player_data`. From now on, use the `get_all_player_data` or the better `get_per_player_data`. More info: *pyplanet.views*.

  The old method will not be called from 0.7.0

- Feature: UI Overhaul is done! We replaced the whole GUI for a nicer, simple and modern one! With large inspiration of LongLife's posted image (https://github.com/PyPlanet/PyPlanet/issues/223).
- Feature: UI Update queue, Don't make the dedicated hot by sending UI updates in realtime, but queue up and sent every 0,25 seconds. (Performance)
- Improvement: Removing the fix for symbols in nicknames/chat (fix for the maniaplanet dedicated/client issue earlier).
- Improvement: Add analytics.
- Improvement: Don't report several exceptions to Sentry.
- Improvement: Remove SQlite references in code and project skeleton.
- Improvement: Give error message when loaded script is using old style scripted callbacks.
- Improvement: Dynamic future timeouts for script/gbx queries.
- Improvement: Add ManiaScript libs includes in core. Will be expanded, open pull requests if needed!
- Improvement: Adding two new signals for players when entering spec/player slot.
- Bugfix: Adding several investigation points to send more data about problems that occur for some users.

33.21.2 Apps

- **Breaking**: Refactor the MapInfo app to Info app. Adding new features: Server and general info on top left corner.

  This requires a config change: Change `pyplanet.apps.contrib.mapinfo` into `pyplanet.apps.contrib.info` and you are done!

  The old app will be removed in 0.7.0

- Feature: **New App**: Shootmania Royal Dynamic Point Limit is here! Add it with `pyplanet.apps.contrib.dynamic_points`.
- Feature: **New App**: Trackmania Checkpoint/Sector time widget is here! Add it with `pyplanet.apps.contrib.sector_times`.
- Feature: Change modesettings directly from the GUI (//modesettings).
- Improvement: Apply the new UI Overhaul to all apps.
- Improvement: Add message when dedimania records are sent.
- Improvement: Improve the dedimania error handling even better.
- Improvement: Notice when map is not suited for dedimania records.
- Improvement: Several performance improvements on the dedimania and localrecords apps.
- Improvement: Add dynamic actions to map list, such as deletion of maps.
- Improvement: Modesettings list is ordered by name by default now.
- Bugfix: Adding several investigation points to send more data about problems that occur for some users.
- Bugfix: Trying to sent dedi records when dedimania isn't initialized bug is solved.
- Bugfix: Prevent double message of dedimania record when switching game modes.
- Bugfix: Fixing double local records (or investigate more if it still occurs).

33.22 0.3.3

33.22.1 Core

- Bugfix: Ignore errors with unknown login for ui updates. (means the player left).

33.22.2 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.

33.23 0.3.2

33.23.1 Core

- Bugfix: Not properly sending and handling mode changes.
- Bugfix: Several errors in handling the callbacks in shootmania

33.23.2 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)

33.24 0.3.1

33.24.1 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.

33.24.2 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

33.25 0.3.0

33.25.1 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.

33.25.2 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.

33.26 0.2.0

33.26.1 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.

33.26.2 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)

33.27 0.1.5

33.27.1 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).

33.27.2 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.

33.28 0.1.4

33.28.1 Core

- Bugfix: Undo locking, causing freeze.

33.29 0.1.3

33.29.1 Apps

- Bugfix: Fixing issue in dedimania causing crash.

33.30 0.1.2

33.30.1 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).

33.30.2 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).

33.31 0.1.1

33.31.1 Core

- Fixing issue with creating migrations folder when no permission.

33.32 0.1.0

33.32.1 Core

- Add new fields to the `game` state class.
- Refresh the `game` infos every minute.

33.32.2 Contrib Apps

- NEW: Dedimania App: Adding dedimania integration and widget.

33.33 0.0.3

33.33.1 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).

33.34 0.0.2

33.34.1 Contrib Apps

- Bugfix Local Records: Widget not updating when map changed. Login not found exception. (#106).

33.35 0.0.1

33.35.1 Core

- First implementation of the core.
- First implementation of the CLI tool.

33.35.2 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

THIRTYFOUR

TODO (DOCS)

Todo: Write introduction + examples.

original entry

Todo: Write introduction.

original entry

Todo: Write introduction.

original entry

CHAPTER

THIRTYFIVE

SOME THOUGHTS FROM EXPERTS

CHAPTER

THIRTYSIX

SCREENSHOTS


18/128
0/32
0 - 50k
PyPlanet 0.4.0
Live Rankings
Dedimania
Local Records
5th 0:13.861
+0:00.144
0:54.150
2
83.3%
10
2:16
5/18
202
KM/H
0.653 KM
0:14.17
REPORT ISSUE / SUGGESTION
Checkpoint 3/8
Round: 00:13.717
00:00.144
mi a neved zsidó 0:12.716
Genry_Galante +0:00.533
+0:00.635
+0:00.770
+0:01.145
Genry_Galante drove the 25. Local Record: 1:04.722.
onyx2000 drove the 29. Local Record: 1:16.113.
onyx2000 drove the 29. Dedimania Record: 1:16.113.
mi a neved zsidó drove the 17. Local Record: 0:56.412.
mi a neved zsidó drove the 17. Dedimania Record: 0:56.412.
drove the 30. Dedimania Record: 1:12.712.
drove the 30. Local Record: 1:12.712.

CHAPTER

THIRTYSEVEN

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

THIRTYEIGHT

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

E

F

G

H

I

L

M

N

O

P

R

S