ReadTrackingData

Generated by Doxygen 1.8.16

1 Tracking Data MsgPack API

1.1 Introduction

This is a header-only API for accessing Tracking Data stored using MessagePack files (see https://msgpack.org/index.html).

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

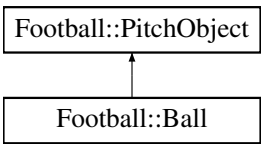
4.1 File List

Here is a list of all documented files with brief descriptions:

5 Data Structure Documentation

5.1 Football::Ball Class Reference

Inheritance diagram for Football::Ball:



Public Member Functions

- **MSGPACK_DEFINE** (FRAME_ID, OBJECT_POS_X, OBJECT_POS_Y, OBJECT_POS_Z, ALIVE, OWNING_TEAM, OWNING_PLAYER_ID)
- **Ball** (std::uint32_t frame_id)
- **Ball** (std::int16_t x, std::int16_t y, std::uint32_t frame_id=0)
- **Ball** (std::pair< std::int16_t, std::int16_t > p)
- **Ball** (const Ball &b, std::uint32_t frame_id)
- virtual void **print** (std::ostream os) const

- std::uint16_t **get_posZ** () const
- void **set_posZ** (const std::uint16_t _z)
- std::array< std::int16_t, 3 > **get_pos** () const
- void **set_pos** (const std::array< std::int16_t, 3 > &_pos)
- bool is_alive () const

  Check whether this Ball is marked alive.
- void **set_alive** (const bool _alive)
- char **get_owningTeam** () const
- void **set_owningTeam** (const char _team)
- std::uint32_t **get_owningPlayerId** () const
- void **set_owningPlayerid** (const std::uint32_t _player_id)
- std::uint32_t **get_frameId** () const
- void **set_frameId** (const std::uint32_t _frame_id)

Static Public Member Functions

- static Ball **createRandomBall** (std::uint32_t frame_id=0)

Protected Attributes

- std::int16_t OBJECT_POS_Z
- bool ALIVE
- char OWNING_TEAM
- std::uint32_t OWNING_PLAYER_ID
- std::uint32_t FRAME_ID

Friends

- class **Match**
- bool **operator==** (const Ball &lhs, const Ball &rhs)

5.1.1 Member Function Documentation

5.1.1.1 is_alive() `bool Football::Ball::is_alive ( ) const`

Check whether this Ball is marked alive.

Returns

The value of ALIVE.

5.1.2 Field Documentation

5.1.2.1 ALIVE `bool Football::Ball::ALIVE [protected]`

Whether the ball is in play (alive).

5.1.2.2 FRAME_ID `std::uint32_t Football::Ball::FRAME_ID [protected]`

The ID or index of the current Frame.

5.1.2.3 OBJECT_POS_Z `std::int16_t Football::Ball::OBJECT_POS_Z [protected]`

Cartesian Z coordinate in centimetres.

5.1.2.4 OWNING_PLAYER_ID `std::uint32_t Football::Ball::OWNING_PLAYER_ID [protected]`

Player::PLAYER_ID of the Player in possession of the ball.

5.1.2.5 OWNING_TEAM `char Football::Ball::OWNING_TEAM [protected]`

Character representation of which team owns the ball.

The documentation for this class was generated from the following file:

- include/FOOTBALL/Ball.hpp

5.2 Football::Frame Class Reference

An object to store Ball and Player objects for a given frame of a Match.

`#include <Match.hpp>`

Public Member Functions

- Frame (std::uint32_t _frame_id, const Ball &_b=Ball(), const Team &_ht=Team(), const Team &_at=Team())
  Parameterised constructor for convenience.
- bool isAlive () const
  Check whether Ball is marked alive in this Frame.

Data Fields

- std::uint32_t FRAME_ID
- Ball BALL
- Team HOMETEAM
- Team AWAYTEAM

5.2.1 Detailed Description

An object to store Ball and Player objects for a given frame of a Match.

Author

Lewis Higgins

Date

$MONTHLONGNAME$ $YEAR$

Examples

cpp_example.cpp.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Frame()

```
Football::Frame::Frame (
        std::uint32_t _frame_id,
        const Ball & _b = Ball(),
        const Team & _ht = Team(),
        const Team & _at = Team() )
```

Parameterised constructor for convenience.

Parameters

_frame_id	Frame ID or index to assign to FRAME_ID.
_b	Ball to be stored in BALL.
_ht	Team object containing Home players to be stored in HOMETEAM.
_at	Team object containing Away players to be stored in AWAYTEAM.

Note

Stores default-initialised Ball and Team objects if none provided.

5.2.3 Member Function Documentation

5.2.3.1 isAlive()

```
bool Football::Frame::isAlive ( ) const
```

Check whether Ball is marked alive in this Frame.

Returns

The value of Ball::is_alive()

Examples

cpp_example.cpp.

5.2.4 Field Documentation

5.2.4.1 AWAYTEAM `Team Football::Frame::AWAYTEAM`

The Team object for the Away Team in this Frame

5.2.4.2 BALL `Ball Football::Frame::BALL`

The Ball object for this Frame.

Examples

cpp_example.cpp.

5.2.4.3 FRAME_ID `std::uint32_t Football::Frame::FRAME_ID`

The ID or index of this Frame.

5.2.4.4 HOMETEAM `Team Football::Frame::HOMETEAM`

The Team object for the Home Team in this Frame

Examples

cpp_example.cpp.

The documentation for this class was generated from the following file:

- include/FOOTBALL/Match.hpp

5.3 Football::Match Struct Reference

`#include <Match.hpp>`

Public Member Functions

- std::uint32_t number_of_frames () const

  Returns the number of frames in the match. Counted from the length of the vectors.
- Frame **get_frame** (std::uint32_t idx) const
- void reduce_to_5fps ()

  Reduce the match to effective 5fps by removing all frames where the frame ID is not a multiple of 5. Skips if match is already 5fps.
- void remove_dead_frames (bool verbose=false)
- void mirror_alternate_periods ()

  Rotate the pitch coordinates for periods 2,4 to stop teams swapping halves after each period.
- void resetFrameIDs ()

  Translates the value of Frame::FRAME_ID for each Frame in this Match such that the first Frame has Frame::FRAME_ID = 0.
- bool loadFromFile (std::string _data_dir, std::uint32_t _match_id, bool fps5=true)

  Loads a full match from a given path into this match object. If fps5 option is true then the '5fps/' subdir is used to load data.

Static Public Member Functions

- template<typename T >
  static bool load_subfile (std::string path, T &store, bool required=true)

  I'm leaving this exposed rather than having it as a protected member. It loads the msgpk file from.
- static bool getMatchFromFile (Match &storage_match, std::string _data_dir, std::uint32_t _match_id, bool fps5=true)

Data Fields

- std::vector< Ball > **BALL_FRAMES**
- std::vector< Team > **HOMETEAM_FRAMES**
- std::vector< Team > **AWAYTEAM_FRAMES**
- std::vector< Team > **OFFICIALS_FRAMES**
- Metadata **METADATA**

5.3.1 Detailed Description

Match structure. Stores a Ball object and two Team objects for each frame in a std::vector. Unlike most of the objects defined in this library, this is a struct and all members are publicly accessible for shortcuts. Be careful with this.

Examples

cpp_example.cpp.

5.3.2 Member Function Documentation

5.3.2.1 getMatchFromFile() `static bool Football::Match::getMatchFromFile (`

```
Match & storage_match,
std::string _data_dir,
std::uint32_t _match_id,
bool fps5 = true )  [static]
```

Loads a full match from a given path. If fps5 option is true then the '5fps/' subdir is used to load data. Match is stored in a provided container.

Parameters

storage_match	- Match object which is wiped then used to store loaded data
_data_dir	- the path to the directory where data folders are stored.
_match_id	- the optaId of the desired match (used to locate the gamePack folder)
fps5	[true] - use the 5fps version of the match

Returns

bool success - whether the file was loaded or not

5.3.2.2 load_subfile()

```
template<typename T >
static bool Football::Match::load_subfile (
            std::string path,
            T & store,
            bool required = true )  [static]
```

I'm leaving this exposed rather than having it as a protected member. It loads the msgpk file from.

Parameters

path	and stores the data in
T&	store. While T is a template it will only work with structures that have proper MsgPack definitions. If you wish to use this function to load a subfile, create an empty std::vector<Football::Team> (or std::vector<Football::Ball> for Ball subfile) and use that vector as the store for this method.
path	- path to subfile
T&	store - object to store data in

Returns

bool success - whether the file was loaded or not

5.3.2.3 loadFromFile()

```
bool Football::Match::loadFromFile (
            std::string _data_dir,
            std::uint32_t _match_id,
            bool fps5 = true )
```

Loads a full match from a given path into this match object. If fps5 option is true then the '5fps/' subdir is used to load data.

Parameters

_data_dir	- the path to the directory where data folders are stored.
_match_id	- the optaId of the desired match (used to locate the gamePack folder)
fps5	[true] - use the 5fps version of the match

Returns

bool success - whether the file was loaded or not

Examples

cpp_example.cpp.

5.3.2.4 remove_dead_frames() `void Football::Match::remove_dead_frames ( bool verbose = false )`

Removes frames where the ball is dead.

Parameters

verbose	- print before and after statistics to std::cout

5.3.2.5 resetFrameIDs() `void Football::Match::resetFrameIDs ( )`

Translates the value of Frame::FRAME_ID for each Frame in this Match such that the first Frame has Frame::FRAME_ID = 0.

Takes the value of Frame::FRAME_ID in the

The documentation for this struct was generated from the following file:

- include/FOOTBALL/Match.hpp

5.4 Football::Metadata Class Reference

Public Member Functions

- void **load_from_file** (const std::string &filepath, const bool verbose=false)
- void adjust_frames (std::int32_t d_frame)

  Translate the frameIds in all period objects by an amount. Used if frameIds are adjusted in the match to ensure that the frameId references in the period objects are correct. E.G if frameIds are translated to make the first frameId = 0.
- std::uint32_t get_matchId () const

  Get optaMatchId.
- void set_matchId (const std::uint32_t _match_id)

  set optaMatchId
- std::string get_date () const

  get string containing match date
- void set_date (const std::string _date)

  set match date string
- std::float_t get_FPS () const

  get the value of Frames Per Seconds (FPS) which the data is recorded at

- void set_FPS (const std::float_t _fps)

  set the value of Frames Per Second (FPS) which the data is recorded at
- std::float_t get_pitchX () const

  get the x dimension of the pitch
- std::float_t get_pitchY () const

  get the y dimension of the pitch
- std::array< std::float_t, 2 > get_pitchDims () const

  get the dimensions of the pitch as a 2D array
- void set_pitchX (const std::float_t _x)

  Set the x dimension of the pitch.
- void set_pitchY (const std::float_t _y)

  Set the y dimension of the pitch.
- void set_pitchDims (const std::float_t _x, const std::float_t _y)

  Set the pitch dimensions.
- void set_pitchDims (const std::pair< std::float_t, std::float_t > &_dims)

  Set the pitch dimensions.
- void set_pitchDims (const std::array< std::float_t, 2 > &_dims)

  Set the pitch dimensions.
- std::uint16_t get_numberOfPeriods () const
- Period & get_period (const std::uint16_t _period_idx)
- void set_period (const std::uint16_t _period_idx, const Period &_period)
- std::vector< Period > & get_periodsVector ()
- void set_periodsVector (const std::vector< Period > &_periods)
- bool get_optaF7 () const

  Check whether an accompanying Opta F7 feed is available.
- void set_optaF7 (const bool _f7)

  Set whether an accompanying Opta F7 feed is available.
- bool get_optaF24 () const

  Check whether an accompanying Opta F24 feed is available.
- void set_optaF24 (const bool _f24)

  Set whether an accompanying Opta F24 feed is available.

Static Public Member Functions

- static Metadata & load_metadata_from_file (Metadata &storage_metadata, const std::string &filepath, const bool verbose=false)

  Load a metadata file from a given path and returns an object representing that data.

Protected Attributes

- std::uint32_t **MATCHID**
- std::string **DATE**
- std::float_t **FPS**
- std::array< std::float_t, 2 > **PITCH_DIMS**
- std::vector< Period > **PERIODS**
- bool **OPTA_F7**
- bool **OPTA_F24**
- std::string **TRACKING_PROVIDER**

Friends

- class **Match**
- bool **operator==** (const Metadata &lhs, const Metadata &rhs)

5.4.1 Member Function Documentation

5.4.1.1 adjust_frames() `void Football::Metadata::adjust_frames ( std::int32_t d_frame )`

Translate the frameIds in all period objects by an amount. Used if frameIds are adjusted in the match to ensure that the frameId references in the period objects are correct. E.G if frameIds are translated to make the first frameId = 0.

All values for start/end frame are subtracted by parameter d_frame.

Parameters

d_frame	- value to subtract from frameIds

5.4.1.2 get_numberOfPeriods() `std::uint16_t Football::Metadata::get_numberOfPeriods ( ) const`

Get the number of periods stored in this metadata file. Counted using std::vector<>.size() with the PERIODS vector.

Returns

integer counting the number of periods in the PERIODS vector

5.4.1.3 get_period() `Period& Football::Metadata::get_period ( const std::uint16_t _period_idx )`

Get period object (by reference) from PERIODS vector.

Parameters

_period_idx	- the index selecting which period in the vector.

Returns

reference to Period object located at _period_idx in PERIODS

Exceptions

std::out_of_range()	if _period_idx >= get_numberOfPeriods()

5.4.1.4 get_periodsVector()

```
std::vector<Period>& Football::Metadata::get_periodsVector ( )
```

Get the full vector (by reference) which all period objects are stored in.

Returns

std::vector<Football::Period>&.

5.4.1.5 load_metadata_from_file()

```
static Metadata& Football::Metadata::load_metadata_from_file (
        Metadata & storage_metadata,
        const std::string & filepath,
        const bool verbose = false )  [static]
```

Load a metadata file from a given path and returns an object representing that data.

Parameters

storage_metadata	- Football::Metadata object to store the data in
filepath	- exact path (i.e with extentions etc.) to metadata file.

Returns

Football::Metadata - Object which contains all the data stored in the metadata file.

Exceptions

std::runtime_error	if std::ifstream fails.

5.4.1.6 set_period()

```
void Football::Metadata::set_period (
        const std::uint16_t _period_idx,
        const Period & _period )
```

Get period object (by reference) from PERIODS vector.

Parameters

_period_idx	- the index indicating location in the vector.
_period	- the period object to store.

Exceptions

std::out_of_range()	if _period_idx >= get_numberOfPeriods().

5.4.1.7 set_periodsVector()

```
void Football::Metadata::set_periodsVector (
        const std::vector< Period > & _periods )
```

Set the full vector which all period objects are stored in.

Parameters

_periods	- vector of periods to store

The documentation for this class was generated from the following file:

- include/FOOTBALL/Metadata.hpp

5.5 Football::Period Class Reference

Public Member Functions

- **Period** (std::uint8_t period_id, std::uint32_t start_frame, std::uint32_t end_frame)
- void adjust_frames (std::uint32_t d_frame)
- std::uint8_t get_periodId () const
- void set_periodId (const std::uint8_t _period_id)
- std::uint32_t get_startFrame () const

  Get the frameId when this period starts.
- void set_startFrame (const std::uint32_t _start_frame)

  Set the frameId when this period starts.
- std::uint32_t get_endFrame () const

  Get the frameId when this period ends.
- void set_endFrame (const std::uint32_t _end_frame)

  Set the frameId when this period ends.

Protected Attributes

- std::uint8_t **PERIOD_ID**
- std::uint32_t **START_FRAME**
- std::uint32_t **END_FRAME**

Friends

- class **Metadata**
- class **Match**
- bool **operator==** (const Period &lhs, const Period &rhs)

5.5.1 Member Function Documentation

5.5.1.1 adjust_frames() `void Football::Period::adjust_frames ( std::uint32_t d_frame )`

Translate the frameIds by an amount. Used if frameIds are adjusted in the match to ensure that the frameId references in the period objects are correct. E.G if frameIds are translated to make the first frameId = 0. All values for start/end frame are subtracted by parameter d_frame.

Parameters

d_frame	- value to subtract from frameIds.

5.5.1.2 get_periodId() `std::uint8_t Football::Period::get_periodId ( ) const`

Get the ID of this period. KEY: 1 - first half, 2 - second half, 3 - first half of ET, 4 - second half of ET, 5 - penalties.

Returns

integer value of this Period's ID.

5.5.1.3 set_periodId() `void Football::Period::set_periodId ( const std::uint8_t _period_id )`

Set the ID of this period. KEY: 1 - first half, 2 - second half, 3 - first half of ET, 4 - second half of ET, 5 - penalties.

Parameters

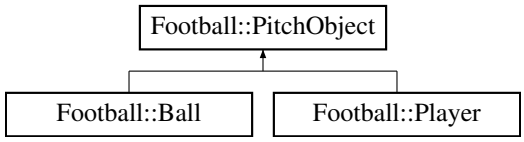
_period_id	- integer value of this Period's ID.

The documentation for this class was generated from the following file:

- include/FOOTBALL/Metadata.hpp

5.6 Football::PitchObject Class Reference

Inheritance diagram for Football::PitchObject:



Public Member Functions

- **PitchObject** (std::int16_t x=0, std::int16_t y=0)
- **PitchObject** (std::pair< std::int16_t, std::int16_t > p)
- std::int16_t get_posX () const

  get the x position
- std::int16_t get_posY () const

  get the y position
- void set_posX (const std::int16_t x)

  set the x position
- void set_posY (const std::int16_t y)

  set the y position
- void **printPosition** () const
- virtual void **print** (std::ostream &where) const

Protected Attributes

- std::int16_t **OBJECT_POS_X**
- std::int16_t **OBJECT_POS_Y**

Friends

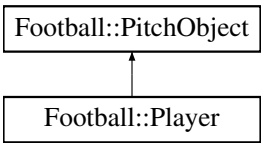
- std::ostream & **operator**<< (std::ostream &, const PitchObject &)

The documentation for this class was generated from the following file:

- include/FOOTBALL/PitchObject.hpp

5.7 Football::Player Class Reference

Inheritance diagram for Football::Player:



Public Member Functions

- **MSGPACK_DEFINE** (PLAYER_ID, PLAYER_SHIRT_NUM, OBJECT_POS_X, OBJECT_POS_Y, BALL_↩ OWNED)
- **Player** (std::int16_t x=0.0, std::int16_t y=0.0, std::uint8_t sn=1)
- **Player** (std::pair< std::int16_t, std::int16_t > p, std::uint8_t sn=1)
- std::array< std::int16_t, 2 > get_pos () const

  get player position as a 2D array
- void set_pos (const std::array< std::int16_t, 2 > &_pos)

  set player position using 2D array
- std::uint8_t get_shirtNumber () const

  get the player's shirt number
- void set_shirtNumber (const std::uint8_t _sn)

  set the player's shirt number
- char get_team () const

  get character representing player's current team. Key: 'H' - Home, 'A' - Away, 'O' - Official, 'U' - Undefined
- void set_team (const char _team)

  set character representing player's current team. Key: 'H' - Home, 'A' - Away, 'O' - Official, 'U' - Undefined
- std::uint32_t get_playerId () const

  get optaPlayerId
- void set_playerId (const std::uint32_t _player_id)

  set optaPlayerId
- bool get_ballOwned () const

  check if this player is specifically in possession of the ball
- bool ownsBall () const

  check if this player is specifically in possession of the ball
- void set_ballOwned (const bool _ball_owned)

  set if this player is specifically in possession of the ball
- std::string get_summaryString () const

  get string with player summary
- virtual void print (std::ostream &os) const

  sends some player details to the provided ostream

Static Public Member Functions

- static Player **createRandomPlayer** (const std::uint16_t sn=1)

Protected Attributes

- std::uint8_t **PLAYER_SHIRT_NUM**
- char **TEAM**
- std::uint32_t **PLAYER_ID**
- bool **BALL_OWNED** = false

Friends

- class **Match**
- class **Team**
- bool **operator==** (const Player &lhs, const Player &rhs)

The documentation for this class was generated from the following file:

- include/FOOTBALL/Player.hpp

5.8 Football::Team Class Reference

Public Member Functions

- std::uint16_t number_of_players () const

  Returns the number of players on the team determined by the size of the PLAYERS_IN_TEAM vector.
- **MSGPACK_DEFINE** (FRAME_ID, TEAM, BALL_OWNED, PLAYERS_IN_TEAM)
- **Team** (std::uint32_t frame_id)
- **Team** (std::vector< Player > plyrs, std::uint32_t frame_id, bool ball_owned=false)
- void setPlayerTeamChar (char team_char)
- std::uint32_t get_frameId () const

  get current frameId
- void set_frameId (const std::uint32_t _frame_id)

  set current frameId
- bool get_ballOwned () const

  Check if team is in possession of the ball.
- bool ownsBalled () const

  Check if team is in possession of the ball.
- void set_ballOwned (bool _ball_owned)

  Set if team is in possession of the ball.
- std::vector< Player > & get_playersInTeam ()

  Returns (by reference) a std::vector containing all the players in the team.
- void set_playersInTeam (const std::vector< Player > &_players_in_team)

  Set the vector of players for this team.
- Player & get_player (const std::uint16_t _player_array_index)
- void set_player (const std::uint16_t _player_array_index, const Player &_player)
- void add_player (const Player &_player)
- void **set_teamChar** (const char _team_char)
- char **get_teamChar** () const

Protected Attributes

- std::uint32_t **FRAME_ID**
- bool **BALL_OWNED** = false
- char **TEAM**
- std::vector< Player > **PLAYERS_IN_TEAM**

Friends

- class **Match**

5.8.1 Member Function Documentation

5.8.1.1 add_player() `void Football::Team::add_player ( const Player & _player )`

Adds a player to the PLAYERS_IN_TEAM vector via the push_back() method.

Parameters

_player	- the player to add.

5.8.1.2 get_player() `Player& Football::Team::get_player ( const std::uint16_t _player_array_index )`

Returns (by reference) a specific Football::Player located by index.

Parameters

_player_array_index	- location of the player in the vector.

Returns

Football::Player located at position _player_array_index in PLAYERS_IN_TEAM vector.

Exceptions

std::out_of_range	if _player_array_index >= number_of_players()

5.8.1.3 set_player() `void Football::Team::set_player ( const std::uint16_t _player_array_index, const Player & _player )`

Assign the Football::Player provided to the PLAYERS_IN_TEAM vector at the given position.

Parameters

_player_array_index	- location in the std::vector to store the player.
_player	- the player to store

Exceptions

std::out_of_range	if _player_array_index >= number_of_players()

5.8.1.4 setPlayerTeamChar() `void Football::Team::setPlayerTeamChar ( char team_char )`

Change the value of TEAM for all players in the team.

Parameters

team_char	- single character representing the players' team. Valid values are 'H' - Home, 'A' - Away, 'O' - Officials, 'U' - Undefined (in case of error)

Exceptions

std::invalid_argument	if team_char not in "HAOU"

The documentation for this class was generated from the following file:

- include/FOOTBALL/Team.hpp

6 File Documentation

6.1 include/FOOTBALL/Football.h File Reference

The main header to include the classes.

```
#include "Ball.hpp"
#include "Match.hpp"
#include "Team.hpp"
#include "Metadata.hpp"
```

6.1.1 Detailed Description

The main header to include the classes.

Author

Lewis Higgins

Date

October 2019

This header provides includes for Ball.hpp, Team.hpp, Metadata.hpp, and Match.hpp. This provides access to Football::Ball, Football::Player, Football::Team, Football::Frame, Football::Metadata, and Football::Match.

6.2 include/FOOTBALL/Match.hpp File Reference

```
#include <cstdlib>
#include <memory>
#include <fstream>
#include <string>
#include <cmath>
#include <msgpack.hpp>
#include "Ball.hpp"
#include "Team.hpp"
#include "Metadata.hpp"
```

Data Structures

- class Football::Frame

  An object to store Ball and Player objects for a given frame of a Match.
- struct Football::Match

Functions

- bool **Football::operator!=** (const Metadata &lhs, const Metadata &rhs)

6.2.1 Detailed Description

Author

Lewis Higgins

Date

$MONTHLONGNAME$ $YEAR$

Defines Match and Frame which are the main methods for accessing tracking data.

Example here:

6.3 include/FOOTBALL/PitchObject.hpp File Reference

Defines the Base Class used for representing anything with a position on the pitch.

```
#include <cstdlib>
#include <utility>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <iostream>
```

Data Structures

- class Football::PitchObject

Functions

- std::ostream & **Football::operator**<< (std::ostream &os, const PitchObject &c)

6.3.1 Detailed Description

Defines the Base Class used for representing anything with a position on the pitch.

Author

Lewis Higgins

Date

October 2019

6.4 include/FOOTBALL/Player.hpp File Reference

Defines a Class used to represent a player.

```
#include <cstdint>
#include <iostream>
#include <msgpack.hpp>
#include "PitchObject.hpp"
```

Data Structures

- class Football::Player

Functions

- bool **Football::operator==** (const Player &lhs, const Player &rhs)
- bool **Football::operator!=** (const Player &lhs, const Player &rhs)

6.4.1 Detailed Description

Defines a Class used to represent a player.

Author

Lewis Higgins

Date

October 2019

7 Example Documentation

7.1 cpp_example.cpp

```
/*
    Lewis Higgins,
    City Football Group & The University of Manchester,
    September 2019

    E: lewis.higgins@postgrad.manchester.ac.uk

    W: https://github.com/hidgjens/ReadTrackingData

    Example for loading and analysing a GamePack.

    To build, please add "include/" to your include-dirs via the -I flag:
        g++ cpp_example.cpp -I"include"

    Might also want to consider building a 64-bit binary using the -m64 flag.

*/
// Football.h will include the whole folder
#include "FOOTBALL/Football.h"
int main (int argc, char * argv[])
{
    // variables to locate game
    std::string     DATA_DIR    = "./";
    uint            MATCH_ID    = 919268;
    bool            mode_5fps   = false;         // true for loading the 5fps version
    // Note that everything from the FOOTBALL folder is stored in namespace Football
    // Create match object
    Football::Match ex_match;
    // load game from file
    ex_match.loadFromFile(DATA_DIR, MATCH_ID, mode_5fps);
    // count the number of frames in possession
    uint        home_possession = 0;
    uint        away_possession = 0;
    uint        total_frames    = 0;       // only counting alive frames
    uint        dead_frames     = 0;
    {
    // create frame object as temporary storage
    Football::Frame _frame;
    // iterate through match frames
    for (uint i = 0 ; i < ex_match.number_of_frames() ; i++ )
    {
        // store current frame in temporary storage
        _frame = ex_match.get_frame(i);
        // std::cout << "frame get " << _frame.FRAME_ID << " (" << i << ")" << std::endl;
        /*
            Analysis for this frame.
        */
        // check if ball is alive in this frame
        if (_frame.isAlive())
        {
            // std::cout << "frame get " << _frame.FRAME_ID << " (" << i << ")" << std::endl;
            // increment alive frames counter
            total_frames ++;
            // check who is in possession
            switch (_frame.BALL.get_owningTeam())
            {
            // home team
            case 'H':
                // increment home counter
                home_possession ++;
                break;
            // away team
            case 'A':
                // increment away counter
                away_possession ++;
                break;
            // officials
            case 'O':
                // officials in possession of the ball?
                std::cerr << "Frame " << i << " official possession?" << std::endl;
                // discount this frame
                total_frames --;
                break;
            // undefined
            case 'U':
                // undefined possession - unlikely to occur, but not necessarily an error
                std::cout << "Frame " << i << " undefined possesion" << std::endl;
                // discount this frame
                total_frames --;
            default:
            // default case is none of the above
```

```
                std::cerr « "Frame " « i « " Default case on switch" « std::endl;
                // discount this frame
                total_frames --;
                break;
            } /* end of switch */
        } /* endif ball alive */
        else
        {
            dead_frames ++;
            // dead frame
            // std::cout « "DEAD" « std::endl;
        }
    } /* for loop ends */
    } // the extra set of curly braces is limiting the scope of _frame, beyond here it is no longer in
       scope. Good practice as _frame was temporary storage for the loop and no longer needed

    // compute fraction of possession from extracted data
    float home_pos_frac = home_possession / ((float) total_frames); // explicitely casting one of these
       numbers to float to avoid integer result i.e. 1/2 = 0 vs 1/2.0 = 0.5
    float away_pos_frac = away_possession / ((float) total_frames);
    // print result to console
    printf ("\nHome team possession %4.1f%%, Away team possession %4.1f%%\n", home_pos_frac * 100.0,
       away_pos_frac * 100.0);

    std::cout « "Alive frames: " « total_frames « " Dead frames: " « dead_frames « std::endl;
    // print starting player line-up
    printf ("\nInitial team line-ups:\n");
    // get the first frame
    auto first_frame = ex_match.get_frame(120255);
    // get the Football::Team objects stored in the frame
    auto& initial_home_team = first_frame.HOMETEAM;
    auto& initial_away_team = first_frame.AWAYTEAM;
    printf ("\tHome Team\n");
    // iterate through the players in team
    for (const auto& player : initial_home_team.get_playersInTeam()) // currently, Football::Team is not
       iterable, but the std::vector\<Football::Player> contained within is
    {
        printf("\t\t%s\n", player.get_summaryString().c_str());
    }
    printf ("\tAway Team\n");
    // iterate through the away players
    for (const auto& player : initial_away_team.get_playersInTeam())
    {
        printf("\t\t%s\n", player.get_summaryString().c_str());
    }
    return EXIT_SUCCESS;
}
/*
    Goal:
    I want to implement the Football::Match object to be iterable, i.e.

    for (auto frame_ : match)
    {
        //  analyse frame_
    }

    likewise for Football::Team:

    for (auto& player : team)
    {
        //  analyse player
    }
*/
```

Index