Reference Manual

Generated by Doxygen 1.7.4

Thu Oct 20 2011 21:41:06

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Parser::SenseBodyData::armStruct Struct Reference

Public Attributes

- double **movable**
- double **expires**
- double **target** [2]
- double **count**

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

- Parser.hpp

3.2 Parser::AuralData Class Reference

```
#include <Parser.hpp>
```

Public Attributes

- int **timestamp**
- string **sender**
- double **direction**
- string **message**

3.2.1 Detailed Description

Holds data parsed from "(hear ...)" messages from the server.

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

- Parser.hpp

3.3 Parser::SenseBodyData::focusStruct Struct Reference

Public Attributes

- string **target**
- double **count**

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

- Parser.hpp

3.4 Parser::SenseBodyData::foulStruct Struct Reference

Public Attributes

- double **charged**
- string **card**

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

- Parser.hpp

3.5 Player Class Reference

```
#include <Player.hpp>
```

Public Member Functions

- Player ()
- bool parseBuffer (const string buffer)
- void printNewestVisualHash (ostream &os) const
- void printNewestVisiblePlayersList (ostream &os) const
- void printNewestAuralStruct (ostream &os) const
- void printNewestSenseBodyStruct (ostream &os) const
- void printServerHash (ostream &os) const
- void printPlayerTypesHash (ostream &os) const
- void printPlayerParamHash (ostream &os) const
- Vector2f getObjectPosition (string objName, int currentTimestamp) const
- void setTeamName (string teamname)

3.5.1 Detailed Description

A class which represents a player: memory, sensory processing, and thinking.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Player::Player ( )

Default constructor. Initializes invalid values for aural and sense body queues. Also initializes stationary flag positions.

Precondition

None.

Postcondition

This object is ready to receive server messages and parse/store their data.

3.5.3 Member Function Documentation

3.5.3.1 Vector2f Player::getObjectPosition (string *objName*, int *currentTimestamp*) const

Retrieves the position of the object from the visual data, estimating its position linearly based on past data if necessary.

Parameters

objName	Name of the object under consideration. E.g. "b" for ball.
current-Timestamp	Timestamp of latest data, used for linear estimation.

Precondition

objName should be a valid identifier for a game object.

Postcondition

None.

Returns

Returns the position of the object, and will return a vector with invalid float values if the object cannot be found and its position cannot be estimated.

3.5.3.2 bool Player::parseBuffer (const string *buffer*)

Parses a buffer of information from the soccer server.

Parameters

buffer	Any S-expression string sent from the soccer server.

Precondition

None.

Postcondition

The relevant data will be stored in the private members of this object based on the type of message parsed. If an error occured, this function will hit an assertion.

3.5.3.3 void Player::printNewestAuralStruct (ostream & *os*) const

Prints the most recently stored aural information struct to the specified output stream.

Parameters

os	The output stream to write to.

Precondition

None.

Postcondition

The contents of the aural struct will be printed to the output stream.

3.5.3.4 void Player::printNewestSenseBodyStruct (ostream & *os*) const

Prints the most recently stored sense body information struct to the specified output stream.

Parameters

os	The output stream to write to.

Precondition

None.

Postcondition

The contents of the sense body struct will be printed to the output stream.

3.5.3.5 void Player::printNewestVisiblePlayersList (ostream & *os*) const

Prints the most recently stored list of visible players to the specified output stream.

Parameters

os	The output stream to write to.

Precondition

None.

Postcondition

The contents of the visible player vector will be printed to the output stream.

3.5.3.6 void Player::printNewestVisualHash (ostream & *os*) const

Prints the most recently stored visual information hash to the specified output stream.

Parameters

os	The output stream to write to.

Precondition

None.

Postcondition

The contents of the hash will be printed to the output stream.

3.5.3.7 void Player::printPlayerParamHash (ostream & *os*) const

Prints the contents of the palyer parameter hash to the specified output stream.

Parameters

os	The output stream to write to.

Precondition

None.

Postcondition

The contents of the hash will be printed to the output stream.

3.5.3.8 void Player::printPlayerTypesHash (ostream & *os*) const

Prints the contents of the player types hash to the specified output stream.

Parameters

os	The output stream to write to.

Precondition

None.

Postcondition

The contents of the hash will be printed to the output stream.

3.5.3.9 void Player::printServerHash (ostream & *os*) const

Prints the contents of the server information hash to the specified output stream.

Parameters

os	The output stream to write to.

Precondition

None.

Postcondition

The contents of the hash will be printed to the output stream.

3.5.3.10 void Player::setTeamName (string *teamname*)

Sets team name

Parameters

teamname	name of team

Precondition

None

Postcondition

Private member teamName is set to appropriate team name

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

- Player.hpp
- Player.cpp

3.6 Parser::PlayerParamStruct Class Reference

```
#include <Parser.hpp>
```

Public Attributes

- double **fValue**

3.6.1 Detailed Description

Holds a float; used for each parameter in the "(player_param ...)" message from the server. Struct wrapping float done for consistency with ServerStruct.

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

- Parser.hpp

3.7 Parser::PlayerTypeStruct Class Reference

```
#include <Parser.hpp>
```

Public Attributes

- double **fValue**

3.7.1 Detailed Description

Holds a float; used for each parameter in the "(player_type ...)" message from the server. Struct wrapping float done for consistency with ServerStruct.

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

- Parser.hpp

3.8 Parser::SenseBodyData Class Reference

```
#include <Parser.hpp>
```

Classes

- struct armStruct
- struct focusStruct
- struct foulStruct
- struct tackleStruct
- struct viewModeStruct

Public Attributes

- int **timestamp**
- bool **absCalculated**
- struct Parser::SenseBodyData::viewModeStruct **view_mode**
- Vector2f **absLocation**
- Vector2f **absVelocity**
- double **stamina** [3]
- double **speed** [2]
- double **head_angle**
- double **kick**
- double **dash**
- double **turn**
- double **say**
- double **turn_neck**
- double **catchCount**
- double **move**
- double **change_view**
- struct Parser::SenseBodyData::armStruct **arm**
- struct Parser::SenseBodyData::focusStruct **focus**
- struct Parser::SenseBodyData::tackleStruct **tackle**
- string **collision**
- struct Parser::SenseBodyData::foulStruct **foul**

3.8.1 Detailed Description

Holds information parsed from "(sense_body ...)" messages from the server. Also holds the absolute position and velocity of the player. This must be calculated from the information transmitted by the server.

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

- Parser.hpp

3.9 Parser::ServerStruct Struct Reference

Public Attributes

- string **sValue**
- double **fValue**

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

- Parser.hpp

3.10 SeverStruct Class Reference

```
#include <Parser.hpp>
```

3.10.1 Detailed Description

Holds a float and a string; one will be invalid while the other is valid, depending on the parameter of the "(server_param ...)" message that this corresponds to.

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

- Parser.hpp

3.11 Parser::SenseBodyData::tackleStruct Struct Reference

Public Attributes

- double **expires**
- double **count**

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

- Parser.hpp

3.12 UDP_client Class Reference

```
#include <udp_client.hpp>
```

Public Member Functions

- ~UDP_client (void)
- UDP_client (void)
- void UDP_close_socket (void)
- void UDP_dbg_log_dsbl (void)
- void UDP_dbg_log_enbl (string filename)
- void UDP_open_socket (string server_ip, unsigned int server_port, string team_-name, unsigned int hdl_idx)

Protected Attributes

- Player **m_player**
- udp_client_cb_t **m_client_cb**

Static Protected Attributes

- static const unsigned **udp_SERVER_PKT_SIZE**

3.12.1 Detailed Description

A class which performs the UDP client processing for each of the players

3.12.2 Constructor & Destructor Documentation

3.12.2.1 UDP_client::∼UDP_client (void)

Class destructor for UDP client

Parameters

None	

Precondition

None

Postcondition

If the UDP socket was opened the socket will be closed. The transmit, receive, and write threads signal termination and the debug log is closed. Finally the Winsock API DLL is closed

3.12.2.2 UDP_client::UDP_client (void)

Class constructor for UDP client

Parameters

None	

Precondition

None

Postcondition

The UDP control block is initialized to invalid data. This is helpful for determining if a process was successful

3.12.3 Member Function Documentation

3.12.3.1 void UDP_client::UDP_close_socket (void)

Close the UDP socket

Parameters

None	

Precondition

None

Postcondition

If the UDP socket was opened the socket will be closed. The transmit, receive, and write threads signal termination and the debug log is closed. Finally the Winsock API DLL is closed

3.12.3.2 void UDP_client::UDP_dbg_log_dsbl (void)

Disable write debug log to file

Parameters

None	

Precondition

None

Postcondition

The write thread is signalled to terminate and the debug log is closed

3.12.3.3 void UDP_client::UDP_dbg_log_enbl (string *filename*)

Enable write debug log to file

Parameters

filename	The filename of the debug log

Precondition

The filename given should not already be in use

Exceptions

If	the file cannot be opened, an assertion is thrown
If	the write thread cannot be created, an assertion is thrown

Postcondition

The debug log file is opened and the write thread is started

3.12.3.4 void UDP_client::UDP_open_socket (string *server_ip*, unsigned int *server_port*, string *team_name*, unsigned int *hdl_idx*)

Open a UDP socket

Parameters

server_ip	The RoboCup server IP
server_port	The RoboCup server port
team_name	The Team Name that is to be used
hdl_index	Will be removed next sprint

Precondition

The UDP control block should be initialized

Exceptions

If	a socket cannot be created, an assertion is thrown
If	the binding of the remote port is unsuccessful, an assertion is thrown
If	the receive thread cannot be created, an assertion is thrown
If	the transmit thread cannot be created, an assertion is thrown

Postcondition

The transmit and receive threads are started and processing begins for each client

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

- udp_client.hpp
- udp_client.cpp

3.13 udp_client_cb_t Class Reference

```
#include <udp_client.hpp>
```

Public Attributes

- char **buffer** [UDP_SRVR_PKT_SIZE]
- ofstream **dbg_log**
- boolean **dbg_log_enbl**
- ostringstream **dbg_log_ss**
- HANDLE **h_rx_thrd**

- HANDLE **h_tx_thrd**
- HANDLE **h_wt_thrd**
- unsigned int **hdl_idx**
- sockaddr_in **lcl_intfc**
- WSAOVERLAPPED **overlapped**
- CRITICAL_SECTION **rx_crit_sec**
- boolean **rx_thrd_alive**
- SOCKET **socket**
- boolean **socket_open**
- boolean **stop_tx_thrd**
- boolean **stop_rx_thrd**
- boolean **stop_wt_thrd**
- sockaddr_in **svr_intfc**
- CRITICAL_SECTION **tx_crit_sec**
- queue< string > **tx_data_q**
- boolean **tx_thrd_alive**
- boolean **wt_thrd_alive**

3.13.1 Detailed Description

A container for all the information related to a single UDP client including each of the required transmit, receive, and write threads

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

- udp_client.hpp

3.14 Vector2f Class Reference

`#include <Vector2f.hpp>`

Public Member Functions

- Vector2f ()
- Vector2f (double x, double y)
- double & operator[] (int index)
- double operator[] (int index) const
- Vector2f operator* (double scale) const
- Vector2f operator/ (double scale) const
- Vector2f operator+ (const Vector2f &other) const
- Vector2f operator- (const Vector2f &other) const
- Vector2f operator- () const
- const Vector2f & operator*= (double scale)
- const Vector2f & operator/= (double scale)
- const Vector2f & operator+= (const Vector2f &other)

- const Vector2f & operator-= (const Vector2f &other)
- double magnitude () const
- double magnitudeSquared () const
- Vector2f normalize () const

3.14.1 Detailed Description

A 2-dimensional vector class. Used for storing positions and velocities.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 Vector2f::Vector2f ( )

Default constructor. Initializes a zero vector.

Precondition

None.

Postcondition

The object will represent the vector (0, 0).

3.14.2.2 Vector2f::Vector2f (double *x*, double *y*)

Typical constructor. Initializes a vector from the input values.

Parameters

x	The magnitude of the vector in the x dimension.
y	The magnitude of the vector in the y dimension.

Precondition

None.

Postcondition

The object will represent the vector (x, y).

3.14.3 Member Function Documentation

3.14.3.1 double Vector2f::magnitude ( ) const

Returns the magnitude of the vector, equal to $\sqrt{x^2 + y^2}$.

Precondition

None.

Postcondition

None.

3.14.3.2 double Vector2f::magnitudeSquared ( ) const

Returns the square of the magnitude of the vector, equal to $x^2 + y^2$.

Precondition

None.

Postcondition

None.

3.14.3.3 Vector2f Vector2f::normalize ( ) const

Normalizes the vector by dividing it by its own magnitude.

Precondition

None.

Postcondition

None.

3.14.3.4 Vector2f Vector2f::operator* (double *scale*) const

Overloaded scalar multiplication.

Parameters

scale	The value to multiply the vector by.

Precondition

None.

Postcondition

This object will be unchanged, and return a vector equal to (scale $*$ x, scale $*$ y).

3.14.3.5 const Vector2f & Vector2f::operator*= (double *scale*)

Overloaded scalar multiplication with assignment.

Parameters

scale	The value to multiply the vector by.

Precondition

None.

Postcondition

This object will now equal (scale ∗ x, scale ∗ y).

3.14.3.6 Vector2f Vector2f::operator+ (const Vector2f & *other*) const

Overloaded vector addition.

Parameters

other	The vector to add to this one.

Precondition

None.

Postcondition

This object will be unchanged, and return a vector equal the sum of the vectors.

3.14.3.7 const Vector2f & Vector2f::operator+= (const Vector2f & *other*)

Overloaded vector addition with assignment.

Parameters

other	The vector to add to this one.

Precondition

None.

Postcondition

This object will now equal the sum of this and the other vector.

3.14.3.8 Vector2f Vector2f::operator- (const Vector2f & *other*) const

Overloaded vector subtraction.

Parameters

other	The vector to subtract from this one.

Precondition

None.

Postcondition

This object will be unchanged, and return a vector equal to this vector minus the other.

3.14.3.9 Vector2f Vector2f::operator- ( ) const

Overloaded unary minus sign.

Precondition

None.

Postcondition

This object will be unchanged, and return a vector equal to (-1 ∗ x, -1 ∗ y).

3.14.3.10 const Vector2f & Vector2f::operator-= (const Vector2f & *other*)

Overloaded vector subtraction with assignment.

Parameters

other	The vector to subtract from this one.

Precondition

None.

Postcondition

This object will now equal the defference of this minus the other vector.

3.14.3.11 Vector2f Vector2f::operator/ (double *scale*) const

Overloaded scalar division.

Parameters

scale	The value to divide the vector by.

Precondition

The value of scale must not be 0..

Postcondition

This object will be unchanged, and return a vector equal to (x / scale, y / scale).

3.14.3.12 const Vector2f & Vector2f::operator/= (double *scale*)

Overloaded scalar division with assignment.

Parameters

scale	The value to divide the vector by.

Precondition

The value of scale must not be 0.

Postcondition

This object will now equal (x / scale, y / scale).

3.14.3.13 double Vector2f::operator[] (int *index*) const

Overloaded square bracket accessor. Used for getting the values of the vector.

Parameters

index	Inicates which value to get, the x value or the y value.

Precondition

The value of index must be 0 (for x) or 1 (for y).

Postcondition

A copy of the data at the specified index will be returned, with no change to the vector.

3.14.3.14 double & Vector2f::operator[] (int *index*)

Overloaded square bracket accessor. Used for setting the values of the vector.

Parameters

index	Indicates which value to set, the x value or the y value.

Precondition

The value of index must be 0 (for x) or 1 (for y).

Postcondition

A reference to the specified value will be returned, so that it may be changed.

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

- Vector2f.hpp
- Vector2f.cpp

3.15 Parser::SenseBodyData::viewModeStruct Struct Reference

Public Attributes

- string **viewQuality**
- string **viewWidth**

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

- Parser.hpp

3.16 Parser::VisiblePlayer Class Reference

```
#include <Parser.hpp>
```

Public Attributes

- std::string **teamName**
- int **uniformNumber**
- bool **isGoalie**
- VisualData **visualData**
- double **bodyDirection**
- double **headDirection**

3.16.1 Detailed Description

Holds information parsed from "(see ...)" messages from the server, for player objects in particular. Note that this also stores a VisualData struct with information about its position, velocity, etc.

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

- Parser.hpp

3.17 Parser::VisualData Class Reference

```
#include <Parser.hpp>
```

Public Attributes

- int **timestamp**
- double **distance**
- double **direction**
- double **distanceChange**
- double **directionChange**
- Vector2f **absLocation**
- Vector2f **absVelocity**

3.17.1 Detailed Description

Holds information parsed from "(see ...)" messages from the server, non-player objects in particular. Player object information is stored elsewhere. Also holds the absolute position and velocity of the objects, such as the ball. This must be calculated from the information transmitted by the server.

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

- Parser.hpp

Chapter 4

File Documentation

4.1 ai_processing.cpp File Reference

```
#include <iostream>
#include <sstream>
#include <string>
#include "ai_processing.hpp"
#include "debug.hpp"
```

Defines

- #define **MIN_MOMENT** (-180)
- #define **MAX_MOMENT** (180)
- #define **MIN_POWER** (-100)
- #define **MAX_POWER** (100)
- #define **MIN_NECK_MOMENT** (-180)
- #define **MAX_NECK_MOMENT** (180)
- #define **MIN_X_COORDINATE** (-52.5)
- #define **MAX_X_COORDINATE** (52.5)
- #define **MIN_Y_COORDINATE** (-34)
- #define **MAX_Y_COORDINATE** (-34)

4.1.1 Detailed Description

Artificial Intelligence Processing

The Artificial Intelligence Processing is the primary module that will handle the decision processing and server string formatting for each of the clients

Author

Joseph Wachtel

Date

Oct 19, 2011

4.2 ai_processing.hpp File Reference

```
#include "udp_client.hpp"
```

Enumerations

- enum **change_view_width_t32** { **CV_WIDTH_NARROW**, **CV_WIDTH_NORMAL**, **CV_WIDTH_WIDE** }
- enum **change_view_quality_t32** { **CV_QUALITY_LOW**, **CV_QUALITY_HIGH** }

Functions

- void AI_Processing::Decision_Processing (void)
- string AI_Processing::Catch_Cmd (double direction)
- string AI_Processing::Change_View_Cmd (change_view_width_t32 width, change_view_quality_t32 quality)
- string AI_Processing::Dash_Cmd (double power)
- string AI_Processing::Kick_Cmd (double power)
- string AI_Processing::Move_Cmd (double x_val, double y_val)
- string AI_Processing::Say_Cmd (string msg_str)
- string AI_Processing::Score_Cmd (void)
- string AI_Processing::Sense_Body_Cmd (void)
- string AI_Processing::Turn_Cmd (double direction)
- string AI_Processing::Turn_Neck_Cmd (double direction)

4.2.1 Detailed Description

Artificial Intelligence Processing Declarations

Declarations for the Artificial Intelligence namespace

Author

Joseph Wachtel

Date

Oct 19, 2011

4.3 Debug.hpp File Reference

```
#include <iostream>
#include <cstdlib>
#include <string>
```

Defines

- #define **DEBUG_MESSAGES_ON**
- #define **ASSERTIONS_ON**
- #define **fatalAssert**(X) if(!(X)) { std::cout << "Fatal assertion failed: " << #X << ", File: " << __FILE__ << ", Line: " << __LINE__ << std::endl; exit(-1); }
- #define **softAssert**(X) if(!(X)) { std::cout << "Soft assertion failed: " << #X << ", File: " << __FILE__ << ", Line: " << __LINE__ << std::endl; }
- #define **alwaysAssert**() { std::cout << "Always assert here: " << "File: " << __FILE__ << ", Line: " << __LINE__ << std::endl; exit(-1); }

Functions

- void printDebugMessage (const std::string &message)
- void fatalError (const std::string &message)

4.3.1 Detailed Description

Contains functions useful for debugging, like asserts.

Author

Keeler Russell

Date

Oct 13, 2011

4.3.2 Function Documentation

4.3.2.1 void fatalError (const std::string & *message*)

Prints a message to standard output before terminating program execution.

Parameters

message	The message to print before exiting.

Precondition

None.

Postcondition

The message will be printed to standard output, then the program will terminate.

4.3.2.2 void printDebugMessage (const std::string & *message*)

Prints a message to standard output.

Parameters

message	The message to print.

Precondition

DEBUG_MESSAGES_ON must be defined.

Postcondition

The message will be printed to standard output.

4.4 main.cpp File Reference

```
#include <iostream>
#include <conio.h>
#include "udp_client.hpp"
```

Defines

- #define **CLIENT_CNT** (11)
- #define **TEAM_NAME** ("team1")

Functions

- int main (int argc, char ∗argv[ ])

4.4.1 Detailed Description

Main Program Processing

Serves as the main entry point for the program. Starts and initializes each client

Author

Joseph Wachtel

Date

Oct 19, 2011

4.4.2 Function Documentation

4.4.2.1 int main (int *argc*, char ∗ *argv[]*)

Main program processing

Parameters

argc	Number of input arguments
∗*argv[]*	An array of the inputs arguments

Precondition

None

Postcondition

Each client is initialized and started in its own thread while the function loops until a keyboard stroke is pressed

Returns

Returns a value of 0

4.5 Parser.hpp File Reference

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>
#include <string.h>
#include <cstdlib>
#include <cmath>
#include "Vector2f.hpp"
```

Classes

- class Parser::AuralData
- class Parser::PlayerParamStruct
- class Parser::PlayerTypeStruct
- class Parser::SenseBodyData
- struct Parser::SenseBodyData::viewModeStruct

- struct Parser::SenseBodyData::armStruct
- struct Parser::SenseBodyData::focusStruct
- struct Parser::SenseBodyData::tackleStruct
- struct Parser::SenseBodyData::foulStruct
- struct Parser::ServerStruct
- class Parser::VisualData
- class Parser::VisiblePlayer

Defines

- #define **PI** 3.14159265
- #define **INVALID_FLOAT_VALUE** -50000.0
- #define **INVALID_UNIFORM_NUMBER** -1
- #define **INVALID_STRING_VALUE** "INVALID STRING"
- #define **INVALID_TEAM_NAME** INVALID_STRING_VALUE
- #define **INVALID_SENDER_NAME** INVALID_STRING_VALUE
- #define **INVALID_DIRECTION** -50000.0

Functions

- bool Parser::isBufferComplete (const string buffer)
- void Parser::parseAuralPacket (const string auralString, AuralData &auralData)
- void Parser::parseInitPacket (const string initString, int &uniformNumber, char &side)
- void Parser::parsePlayerParamPacket (const string buffer, unordered_map< string, PlayerParamStruct > &playerParams)
- void Parser::parsePlayerTypePacket (const string buffer, unordered_map< string, PlayerTypeStruct > playerTypes[ ])
- void Parser::parseSenseBodyPacket (const string inData, SenseBodyData &sbd)
- void Parser::parseServerPacket (const string buffer, unordered_map< string, ServerStruct > &serverInfo)
- void Parser::parseVisualPacket (const string visualString, unordered_map< string, VisualData > &visualHash, vector< VisiblePlayer > &visiblePlayers)
- void Parser::convertToAbsoluteCoordsAndVelocity (unordered_map< string, VisualData > &visualHash, vector< VisiblePlayer > &visiblePlayers, SenseBodyData &senseBodyData, unordered_map< string, Vector2f > &stationaryFlags)
- double Parser::getAbsoluteAngle (double absAngle, double refAngle)
- vector< VisiblePlayer > Parser::getTeammateIdentities (string teamName, const vector< VisiblePlayer > &visiblePlayers)
- vector< VisiblePlayer > Parser::getOpponentIdentities (string teamName, const vector< VisiblePlayer > &visiblePlayers)
- vector< VisiblePlayer > Parser::getUnidentifiedIdentities (string teamName, const vector< VisiblePlayer > &visiblePlayers)

4.5.1 Detailed Description

Module used for parsing S-expressions from the server.

Author

Keeler Russell, Jared Mar, Corbin Charpentier

Date

Oct 15, 2011

4.6 Player.hpp File Reference

```
#include "Parser.hpp"
#include "Vector2f.hpp"
#include "Brain.h"
#include <iostream>
#include <string>
#include <cstdlib>
#include <unordered_map>
#include <deque>
```

Classes

- class Player

Defines

- #define **NUM_PLAYER_TYPES** 17
- #define **MAX_QUEUE_SIZE** 10

4.6.1 Detailed Description

Represents a player on the field.

Author

Keeler Russell, Jared Mar, Corbin Charpentier

Date

Oct 13, 2011

4.7 udp_client.cpp File Reference

```
#include <iostream>
#include <sstream>
#include <string>
#include "debug.hpp"
#include "udp_client.hpp"
```

Defines

- #define **WSA_VER_H** (2)
- #define **WSA_VER_L** (2)

4.7.1 Detailed Description

UDP Client Processing

Handles the threading of both transmit and receive for each client which communicates via the UDP protocol with the soccer server

Author

Joseph Wachtel

Date

Oct 19, 2011

4.8 udp_client.hpp File Reference

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <ws2tcpip.h>
#include <queue>
#include "demo.hpp"
#include "Player.hpp"
```

Classes

- class udp_client_cb_t

- class UDP_client

Defines

- #define **UDP_SRVR_PKT_SIZE** (8192)

4.8.1 Detailed Description

UDP Client Processing Declarations

Declarations for the UDP Client class

Author

Joseph Wachtel

Date

Oct 19, 2011

4.9 Vector2f.hpp File Reference

```
#include <iostream>
```

Classes

- class Vector2f

Functions

- std::ostream & operator<< (std::ostream &os, const Vector2f &vector)

4.9.1 Detailed Description

Represents a 2-dimensional vector.

Author

Keeler Russell

Date

Oct 13, 2011

4.9.2 Function Documentation

4.9.2.1 std::ostream& operator<< (std::ostream & *os*, const Vector2f & *vector*)

Overloaded stream output operator.

Parameters

os	The output stream to write the vector to.
vector	The vector to write to the output stream.

Precondition

None.

Postcondition

The vector will be printed out with no trailing newline.

Index