The RoboCup Client Parser
Reference Manual
1.2.2

April 24, 2003

By Tom Howard
tomhoward@users.sf.net

Contents

1 RoboCup Client Parser (RCCParser) 1

1.1 Introduction 1

1.2 Installation 1

1.3 Uninstalling 4

1.4 Using The Parser 5

1.5 Contacts 5

2 RoboCup Client Parser (rccparser) Namespace Index 7

2.1 RoboCup Client Parser (rccparser) Namespace List 7

3 RoboCup Client Parser (rccparser) Hierarchical Index 9

3.1 RoboCup Client Parser (rccparser) Class Hierarchy 9

4 RoboCup Client Parser (rccparser) Compound Index 11

4.1 RoboCup Client Parser (rccparser) Compound List 11

5 RoboCup Client Parser (rccparser) File Index 13

5.1 RoboCup Client Parser (rccparser) File List 13

6 RoboCup Client Parser (rccparser) Namespace Documentation 15

6.1 rcc Namespace Reference 15

7 RoboCup Client Parser (rccparser) Class Documentation 17

7.1 rcc::Parser Class Reference 17

8 RoboCup Client Parser (rccparser) File Documentation 167

8.1 rccparser.h File Reference 167

Chapter 1

RoboCup Client Parser (RCCParser)

1.1 Introduction

RCCParser is a parser library for RoboCup clients for the `RoboCup Soccer Simulator`. It can be used for field players and coaches using protocol versions 7 to 9.

The parser is Flex and Bison based so it should be fast compared to a hand written parser.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU GPL as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This product includes software (Flex) developed by the University of California, Berkeley and its contributors.

1.2 Installation

1.2.1 Basic Installation

The 'configure' shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a 'Makefile' in each directory of the package. It may also create one or more '.h' files containing system-dependent definitions. Finally, it creates a shell script 'config.status' that you can run in the future to recreate the current configuration, and a file 'config.log' containing compiler output (useful mainly for debugging 'configure').

It can also use an optional file (typically called 'config.cache' and enabled with '–cache-file=config.cache' or simply '-C') that saves the results of its tests to speed up reconfiguring (Caching is disabled by default to prevent problems with accidental use of stale cache files). If you are using the cache, and at some point 'config.cache' contains results you don't want to keep, you may remove or edit it.

If you need to do unusual things to compile the package, please try to figure out how 'configure' could check whether to do them, and mail diffs or instructions to rccparser-bugs@lists.sf.net so they can be considered for the next release.

The file 'configure.in' is used to create 'configure' by a program called 'autoconf'. You only need 'configure.in' if you want to change it or regenerate 'configure' using a newer version of 'autoconf'.

The simplest way to compile this package is:

1. 'cd' to the directory containing the package's source code and type './configure' to configure the package for your system. If you're using 'csh' on an old version of System V, you might need to type 'sh ./configure' instead to prevent 'csh' from trying to execute 'configure' itself.

   Running 'configure' takes awhile. While running, it prints some messages telling which features it is checking for.

2. Type 'make' to compile the package.

3. Optionally, type 'make check' to run any self-tests that come with the package.

4. Type 'make install' to install the programs and any data files and documentation.

5. You can remove the program binaries and object files from the source code directory by typing 'make clean'. To also remove the files that 'configure' created (so you can compile the package for a different kind of computer), type 'make distclean'. There is also a 'make maintainer-clean' target, but that is intended mainly for the package's developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

1.2.2 Compilers and Options

Some systems require unusual options for compilation or linking that the 'configure' script does not know about. Run './configure –help' for details on some of the pertinent environment variables.

You can give 'configure' initial values for configuration parameters by setting variables in the command line or in the environment. Here is an example:

./configure CC=c89 CFLAGS=-O2 LIBS=-lposix

See Defining Variables for more information.

1.2.3 Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you must use a version of 'make' that supports the 'VPATH' variable, such as GNU 'make'. 'cd' to the directory where you want the object files and executables to go and run the ‘configure’ script. ‘configure’ automatically checks for the source code in the directory that ‘configure’ is in and in ‘..’.

If you have to use a ‘make’ that does not support the ‘VPATH’ variable, you have to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use ‘make distclean’ before reconfiguring for another architecture.

1.2.4 Installation Names

By default, ‘make install’ will install the package’s files in ‘/usr/local/bin’, ‘/usr/local/man’, etc. You can specify an installation prefix other than ‘/usr/local’ by giving ‘configure’ the option ‘–prefix=PATH’.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give ‘configure’ the option ‘–exec-prefix=PATH’, the package will use PATH as the prefix for installing programs and libraries. Documentation and other data files will still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like ‘–bindir=PATH’ to specify different values for particular kinds of files. Run ‘configure –help’ for a list of the directories you can set and what kinds of files go in them.

1.2.5 Optional Features

‘configure’ also allows you to select and deselect optional features when building RCCParser, the most relevant of these being ‘–enable-rccptest=ARG’, ‘–enable-shared=ARG’ and ‘–enable-static=ARG’.

‘–enable-rccptest=ARG’ can be used to enable of disable the building of the rccptest executable, which is used to test the parser by reading data to be parsed from stdin. By default, building rccptest is enabled.

‘–enable-shared=ARG’ and ‘–enable-static=ARG’ respectively enable and disable the building of shared and static versions of the library. Both are enabled by default (that is, when the system supports shared libraries, otherwise only static libraries are built). There are advantages to using shared libraries (that I wont go into here), but they can be a pain when trying to debug a program. For that reason, it is recommended that you disable shared libraries when debugging.

1.2.6 Specifying the System Type

There may be some features ‘configure’ cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the _same_ architectures, ‘configure’ can figure that out, but if it prints a message saying it cannot guess the machine type, give it the ‘–build=TYPE’ option. TYPE can either be a short name for the system type, such as ‘sun4’, or a canonical name which has the form:

CPU-COMPANY-SYSTEM

where SYSTEM can have one of these forms:

OS KERNEL-OS

See the file ‘config.sub’ for the possible values of each field. If ‘config.sub’ isn’t included in this package, then this package doesn’t need to know the machine type.

If you want to _use_ a cross compiler, that generates code for a platform different from the build platform, you should specify the ”host” platform (i.e., that on which the generated programs will eventually be run) with ‘–host=TYPE’.

1.2.7 Defining Variables

Variables not defined in a site shell script can be set in the environment passed to ‘configure’. However, it is best to set them in the ‘configure’ command line, using ‘VAR=value’. For example:

./configure CC=/usr/local2/bin/gcc

will cause the specified gcc to be used as the C compiler.

1.2.8 ‘configure’ Invocation

‘configure’ recognizes the following options to control how it operates.

‘–help’ ‘-h’ Print a summary of the options to ‘configure’, and exit.

‘–version’ ‘-V’ Print the version of Autoconf used to generate the ‘configure’ script, and exit.

‘–cache-file=FILE’ Enable the cache: use and save the results of the tests in FILE, traditionally ‘config.cache’. FILE defaults to ‘/dev/null’ to disable caching.

‘–config-cache’ ‘-C’ Alias for ‘–cache-file=config.cache’.

‘–quiet’ ‘–silent’ ‘-q’ Do not print messages saying which checks are being made. To suppress all normal output, redirect it to ‘/dev/null’ (any error messages will still be shown).

‘–srcdir=DIR’ Look for the package’s source code in directory DIR. Usually ‘configure’ can determine that directory automatically.

‘configure’ also accepts some other, not widely useful, options. Run ‘configure –help’ for more details.

1.3 Uninstalling

RCCParser can be easily removed by entering the src directory and running ‘make uninstall’. This will remove all the files that where installed, but not any directories that were created during the installation process.

1.4 Using The Parser

RCCParser takes care of parsing data from the simulator as a std::ostream, what it doesn't take care of is what to do with the parsed data. That is the job of your client. The default behaviour of the parser is to do nothing with the parsed data.

To overide the default behaviour, you need to subclass rcc::Parser and overide the virtual functions that the parser calls.

As a RoboCup message is parsed, the virtual functions will be called to from the bottom up. For instance for the message

(init l 2 before_kick_off)

The follwing functions will be called:

1. rcc::Parser::doBuildLeftSide()
2. rcc::Parser::doBuildBeforeKickOffPlayMode()
3. rcc::Parser::doBuildInit()

You will also need to pass a rcss::Parser object (note the different namespace) to rcc::Parser when it is initialized. This so the rcssclangparser library that comes with rcsserver can be used to parse CLang messages (thus making sure the CLang parsing is always upto date).

Assuming the data to be parsed is in a char array, the following code would cause parsing to occur.

```
rcss::clang::MsgBuilder clang_builder;
rcss::clang::Parser clang_parser( clang_builder );
YourParser parser( clang_parser );

[...snip...]


istrstream strm( buffer ); // buffer is the char array with the data
if( !parser.parse( strm ) )
{
   // handle error
}
```

Instances of rcc::Parser derived classes can be used in a multi-threaded environment, but you will either need prevent simultaneous access by using a mutex before the call the parse, or by using a different instance in each thread. Normally clients only need one instance of the parser, which is only accessed from a single thread.

1.5 Contacts

The RCCParser home page should be your main point of contact for all matters concerning RCCParser, like getting the latest versions, lodging support requests, bug reports and contacting other users of RCCParser library.

At the home page you will also find access the the RCCParser mailing lists, `rccparser-main@lists.sf.net` and `rccpaser-bugs@lists.sf.net.`

1.5.1 rccpaser-bugs@lists.sf.net

This secondary list is for reporting bugs or potential bugs in the parser. This could be anything like parse errors, building and installation problems, or even defects in the documentation. I've tried to make building, installing and using the parser as simple as possible, so if you run into any problems whatsoever, you should post it to this list.

1.5.2 rccpaser-main@lists.sf.net

The `rccparser-main@lists.sf.net` list is the main list for the RCCParser library, where you will mostly find announcements for the latest releases. If you have any questions or suggestions feel free to post them here.

I hope your team development goes well, that this tool makes it that little bit easier and I hope to see you at the next RoboCup World Cup.

Chapter 2

RoboCup Client Parser (rccparser) Namespace Index

2.1 RoboCup Client Parser (rccparser) Namespace List

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

rcc (The vendor namespace used by RCCParser) 15

Chapter 3

RoboCup Client Parser (rccparser) Hierarchical Index

3.1 RoboCup Client Parser (rccparser) Class Hierarchy

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

- rcc::Holder
- Parser
  - CLangParser
  - rcc::Parser 17
    - TestParser
- yyFlexLexer
  - RCCLexer
    - rcc::Lexer

Chapter 4

RoboCup Client Parser (rccparser) Compound Index

4.1 RoboCup Client Parser (rccparser) Compound List

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

rcc::Parser (The parsing base class) 17

Chapter 5

RoboCup Client Parser (rccparser) File Index

5.1 RoboCup Client Parser (rccparser) File List

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

rccparser.h 167

Chapter 6

RoboCup Client Parser (rccparser) Namespace Documentation

6.1 rcc Namespace Reference

The vendor namespace used by RCCParser.

Compounds

- class **rcc::Holder**
- class **rcc::Lexer**
- class rcc::Parser

    The parsing base class.

- class **rcc::Parser::Param**

6.1.1 Detailed Description

The vendor namespace used by RCCParser.

Chapter 7

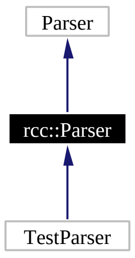
RoboCup Client Parser (rccparser) Class Documentation

7.1 rcc::Parser Class Reference

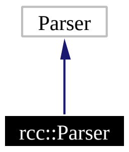
The parsing base class.

`#include <rccparser.h>`

Inheritance diagram for rcc::Parser:



Collaboration diagram for rcc::Parser:



Public Types

- typedef rcc::Lexer **Lexer**

Public Methods

- Parser (rcss::Parser &clang_parser)

  Parser constructor.

- virtual ∼**Parser** ()

Parsing Functions

- bool parse (std::istream &strm)

  Main parsing function.

- bool parse (const std::string &file)

  Secondary parsing function.

Protected Methods

- virtual void doMsgParsed ()

  This function is called every time a message has been parsed.

- virtual void doBuildTeamName (const std::string &name)

  This function is called after parsing a team name.

- virtual void doBuildUNum (int unum)

  This function is called after parsing a uniform number.

- virtual void doBuildGoalie ()

  This function is called after parsing a goalie indicator of a player name.

- virtual void doBuildScore (int time, int our, int opp)

  This function is called after parsing a score message.

CLang Parsing Functions

- virtual void doParsedCLang (const std::string &msg)

  This function is called every time a CLang message has been parsed, if doParsedCLang(const char∗ msg) has not been overridden.

- virtual void doParsedCLang (const char ∗msg)

  This function is called every time a CLang message has been parsed.

Full State Parsing Functions

- virtual void doBuildFullState (int time)

  This function is called after parsing a full state message.

- virtual void doBuildCounts (int kick, int dash, int turn, int katch, int move, int turn_neck, int change_view, int say)

  This function is called after parsing the count section of a version 8 or 9 full state message.

- virtual void doBuildScore (int our, int opp)

  This function is called after parsing the score section of a full state message.

- virtual void doBuildBall (double x, double y, double delta_x, double delta_y)

  This function is called after parsing the ball section of a full state message.

- virtual void doBuildPlayer (int unum, double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, int stamina, double effort, double recovery)

  This function is called after parsing a player section of a version 7 full state message.

- virtual void doBuildPlayer (int unum, int type, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double stamina, double effort, double recovery)

  This function is called after parsing a player section of a version 8 or 9 full state message, where the player is **not** *pointing and the player is* **not** *a goalie.*

- virtual void doBuildPlayer (int unum, int type, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)

  This function is called after parsing a player section of a version 8 or 9 full state message, where the player **is** *pointing and the player is* **not** *a goalie.*

- virtual void doBuildGoalie (int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double stamina, double effort, double recovery)

  This function is called after parsing a player section of a version 8 or 9 full state message, where the player is **not** *pointing and the player* **is** *a goalie.*

- virtual void doBuildGoalie (int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)

  This function is called after parsing a player section of a version 8 or 9 full state message, where the player **is** *pointing and the player* **is** *a goalie.*

PlayMode Parsing Functions

- virtual void doBuildCatchBallLeftPlayMode ()

This function is called after parsing a `goalie_catch_ball_l` *play mode.*

- virtual void doBuildCatchBallRightPlayMode ()
  This function is called after parsing a `goalie_catch_ball_r` *play mode.*
- virtual void doBuildBeforeKickOffPlayMode ()
  This function is called after parsing a `before_kick_off` *play mode.*
- virtual void doBuildTimeOverPlayMode ()
  This function is called after parsing of a `time_over` *play mode.*
- virtual void doBuildPlayOnPlayMode ()
  This function is called after parsing of a `play_on` *play mode.*
- virtual void doBuildKickOffLeftPlayMode ()
  This function is called after parsing of a `kick_off_l` *play mode.*
- virtual void doBuildKickOffRightPlayMode ()
  This function is called after parsing of a `kick_off_r` *play mode.*
- virtual void doBuildKickInLeftPlayMode ()
  This function is called after parsing of a `kick_in_l` *play mode.*
- virtual void doBuildKickInRightPlayMode ()
  This function is called after parsing of a `kick_in_r` *play mode.*
- virtual void doBuildFreeKickLeftPlayMode ()
  This function is called after parsing of a `free_kick_l` *play mode.*
- virtual void doBuildFreeKickRightPlayMode ()
  This function is called after parsing of a `free_kick_r` *play mode.*
- virtual void doBuildCornerKickLeftPlayMode ()
  This function is called after parsing of a `corner_kick_l` *play mode.*
- virtual void doBuildCornerKickRightPlayMode ()
  This function is called after parsing of a `corner_kick_r` *play mode.*
- virtual void doBuildGoalKickLeftPlayMode ()
  This function is called after parsing of a `goal_kick_l` *play mode.*
- virtual void doBuildGoalKickRightPlayMode ()
  This function is called after parsing of a `goal_kick_r` *play mode.*
- virtual void doBuildAfterGoalLeftPlayMode ()
  This function is called after parsing of a `goal_l` *play mode.*
- virtual void doBuildAfterGoalRightPlayMode ()
  This function is called after parsing of a `goal_r` *play mode.*
- virtual void doBuildDropBallPlayMode ()
  This function is called after parsing of a `drop_ball` *play mode.*

- virtual void doBuildOffSideLeftPlayMode ()
  This function is called after parsing of a `offside_l` *play mode.*
- virtual void doBuildOffsideRightPlayMode ()
  This function is called after parsing of a `offside_r` *play mode.*
- virtual void doBuildPenaltyKickLeftPlayMode ()
  This function is called after parsing of a `penalty_kick_l` *play mode.*
- virtual void doBuildPenaltyKickRightPlayMode ()
  This function is called after parsing of a `penalty_kick_r` *play mode.*
- virtual void doBuildFirstHalfOverPlayMode ()
  This function is called after parsing of a `first_half_over` *play mode.*
- virtual void doBuildPausePlayMode ()
  This function is called after parsing of a `pause` *play mode.*
- virtual void doBuildHumanPlayMode ()
  This function is called after parsing of a `human_judge` *play mode.*
- virtual void doBuildFoulLeftPlayMode ()
  This function is called after parsing of a `foul_l` *play mode.*
- virtual void doBuildFoulRightPlayMode ()
  This function is called after parsing of a `foul_r` *play mode.*
- virtual void doBuildFoulChargeLeftPlayMode ()
  This function is called after parsing of a `foul_charge_l` *play mode.*
- virtual void doBuildFoulChargeRightPlayMode ()
  This function is called after parsing of a `foul_charge_r` *play mode.*
- virtual void doBuildFoulPushLeftPlayMode ()
  This function is called after parsing of a `foul_push_l` *play mode.*
- virtual void doBuildFoulPushRightPlayMode ()
  This function is called after parsing of a `foul_push_r` *play mode.*
- virtual void doBuildFoulMultipleAttackerLeftPlayMode ()
  This function is called after parsing of a `foul_multiple_attack_l` *play mode.*
- virtual void doBuildFoulMultipleAttackerRightPlayMode ()
  This function is called after parsing of a `foul_multiple_attack_r` *play mode.*
- virtual void doBuildFoulBallOutLeftPlayMode ()
  This function is called after parsing of a `foul_ballout_l` *play mode.*
- virtual void doBuildFoulBallOutRightPlayMode ()

This function is called after parsing of a `foul_ballout_r` *play mode.*

- virtual void doBuildBackPassLeftPlayMode ()
  This function is called after parsing of a `back_pass_l` *play mode.*
- virtual void doBuildBackPassRightPlayMode ()
  This function is called after parsing of a `back_pass_r` *play mode.*
- virtual void doBuildFreeKickFaultLeftPlayMode ()
  This function is called after parsing of a `free_kick_fault_l` *play mode.*
- virtual void doBuildFreeKickFaultRightPlayMode ()
  This function is called after parsing of a `free_kick_fault_r` *play mode.*
- virtual void doBuildCatchFaultLeftPlayMode ()
  This function is called after parsing of a `catch_fault_l` *play mode.*
- virtual void doBuildCatchFaultRightPlayMode ()
  This function is called after parsing of a `catch_fault_r` *play mode.*
- virtual void doBuildIndirectFreeKickLeftPlayMode ()
  This function is called after parsing of a `indirect_free_kick_l` *play mode.*
- virtual void doBuildIndirectFreeKickRightPlayMode ()
  This function is called after parsing of a `indirect_free_kick_r` *play mode.*
- virtual void doBuildPenaltySetupLeftPlaymode ()
  This function is called after parsing of a `penalty_setup_l` *play mode.*
- virtual void doBuildPenaltySetupRightPlaymode ()
  This function is called after parsing of a `penalty_setup_r` *play mode.*
- virtual void doBuildPenaltyReadyLeftPlaymode ()
  This function is called after parsing of a `penalty_ready_l` *play mode.*
- virtual void doBuildPenaltyReadyRightPlaymode ()
  This function is called after parsing of a `penalty_ready_r` *play mode.*
- virtual void doBuildPenaltyTakenLeftPlaymode ()
  This function is called after parsing of a `penalty_taken_l` *play mode.*
- virtual void doBuildPenaltyTakenRightPlaymode ()
  This function is called after parsing of a `penalty_taken_r` *play mode.*
- virtual void doBuildPenaltyMissLeftPlaymode ()
  This function is called after parsing of a `penalty_miss_l` *play mode.*
- virtual void doBuildPenaltyMissRightPlaymode ()
  This function is called after parsing of a `penalty_miss_r` *play mode.*
- virtual void doBuildPenaltyScoreLeftPlaymode ()
  This function is called after parsing of a `penalty_score_l` *play mode.*

- virtual void doBuildPenaltyScoreRightPlaymode ()
  This function is called after parsing of a `penalty_score_r` *play mode.*
- virtual void doBuildPenaltyOnFieldLeftPlaymode ()
  This function is called after parsing of a `penalty_onfield_l` *play mode.*
- virtual void doBuildPenaltyOnFieldRightPlaymode ()
  This function is called after parsing of a `penalty_onfield_r` *play mode.*
- virtual void doBuildPenaltyFoulLeftPlaymode ()
  This function is called after parsing of a `penalty_foul_l` *play mode.*
- virtual void doBuildPenaltyFoulRightPlaymode ()
  This function is called after parsing of a `penalty_foul_r` *play mode.*
- virtual void doBuildPenaltyWinnerLeftPlaymode ()
  This function is called after parsing of a `penalty_winner_l` *play mode.*
- virtual void doBuildPenaltyWinnerRightPlaymode ()
  This function is called after parsing of a `penalty_winner_r` *play mode.*
- virtual void doBuildPenaltyDrawPlaymode ()
  This function is called after parsing of a `penalty_draw` *play mode.*

Side Parsing Functions

- virtual void doBuildLeftSide ()
  This function is called after parsing a token representing the left team side.
- virtual void doBuildRightSide ()
  This function is called after parsing a token representing the right team side.
- virtual void doBuildOppSide ()
  This function is called after parsing a token representing the client's team's side.
- virtual void doBuildOurSide ()
  This function is called after parsing a token representing opponent's team's side.
- virtual void doBuildNeutralSide ()
  This function is called after parsing a token representing the neither side.

Sense Body Parsing Functions

- virtual void doBuildSenseBody (int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
  This function is called after parsing a sense body message.

- virtual void doBuildSenseBody (int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)

  This function is called after parsing a sense body message, if doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count) has not been overridden.

- virtual void doBuildArmState (int movable_in, int expires_in, double target_x, double target_y, int count)

  This function is called after parsing an arm section of a version 8 or 9 sense body message or a version 8 or 9 full state message.

- virtual void doBuildFocusState (int count)

  This function is called after parsing an focus state section of a version 8 or 9 sense body, where the client is **not** *focused on any player.*

- virtual void doBuildFocusState (int unum, int count)

  This function is called after parsing a focus state section of a version 8 or 9 sense body, where the client **is** *focused on a player.*

- virtual void doBuildTackleState (int expires_in, int count)

  This function is called after parsing a tackle state section of a version 8 or 9 sense body.

View Mode Parsing Functions

- virtual void doBuildViewMode ()

  This function is called after parsing a view mode section of a sense body of full state message.

- virtual void doBuildViewQualityHigh ()

  This function is called after parsing a `high` *view quality in the view mode section of a sense body or full state message.*

- virtual void doBuildViewQualityLow ()

  This function is called after parsing a `low` *view quality in the view mode section of a sense body or full state message.*

- virtual void doBuildViewWidthNarrow ()

  This function is called after parsing a `narrow` *view width in the view mode section of a sense body or full state message.*

- virtual void doBuildViewWidthNormal ()

  This function is called after parsing a `normal` *view width in the view mode section of a sense body or full state message.*

- virtual void doBuildViewWidthWide ()

This function is called after parsing a `wide` *view width in the view mode section of a sense body or full state message.*

Parameter Parsing Functions

- virtual void doBuildServerParam (double gwidth, double inertia_moment, double psize, double pdecay, double prand, double pweight, double pspeed_max, double paccel_max, double stamina_max, double stamina_inc, double recover_init, double recover_dthr, double recover_min, double recover_dec, double effort_init, double effort_dthr, double effort_min, double effort_dec, double effort_ithr, double effort_inc, double kick_rand, int team_actuator_noise, double prand_factor_l, double prand_factor_r, double kick_rand_factor_l, double kick_rand_factor_r, double bsize, double bdecay, double brand, double bweight, double bspeed_max, double baccel_max, double dprate, double kprate, double kmargin, double ctlradius, double ctlradius_width, double maxp, double minp, double maxm, double minm, double maxnm, double minnm, double maxn, double minn, double visangle, double visdist, double windir, double winforce, double winang, double winrand, double kickable_area, double catch_area_l, double catch_area_w, double catch_prob, int goalie_max_moves, double ckmargin, double offside_area, int win_no, int win_random, int say_cnt_max, int SayCoachMsgSize, int clang_win_size, int clang_define_win, int clang_meta_win, int clang_advice_win, int clang_info_win, int clang_mess_delay, int clang_mess_per_cycle, int half_time, int sim_st, int send_st, int recv_st, int sb_step, int lcm_st, int SayMsgSize, int hear_max, int hear_inc, int hear_decay, int cban_cycle, int slow_down_factor, int useoffside, int kickoffoffside, double offside_kick_margin, double audio_dist, double dist_qstep, double land_qstep, double dir_qstep, double dist_qstep_l, double dist_qstep_r, double land_qstep_l, double land_qstep_r, double dir_qstep_l, double dir_qstep_r, int CoachMode, int CwRMode, int old_hear, int sv_st, int start_goal_l, int start_goal_r, int fullstate_l, int fullstate_r, int drop_time)
  This ungainly function is called after parsing a version 7 server parameter message.

- virtual void doBuildServerParam ()
  This function is called after parsing a version 8 or 9 server parameter message.

- virtual void doBuildParam (const std::string &name, int value)
  This function is called after parsing a param section of a version 8 or 9 server parameter, player parameter or player type message, where the type of the parameter is an integer.

- virtual void doBuildParam (const std::string &name, double value)
  This function is called after parsing a param section of a version 8 or 9 server parameter, player parameter or player type message, where the type of the parameter is an real.

- virtual void doBuildParam (const std::string &name, const std::string &value)
  This function is called after parsing a param section of a version 8 or 9 server parameter, player parameter or player type message, where the type of the parameter is an string.

- virtual void doBuildPlayerParam (int player_types, int subs_max, int pt_max, double player_speed_max_delta_min, double player_speed_max_delta_max, double stamina_inc_max_delta_factor, double player_decay_delta_min, double player_decay_delta_max, double inertia_moment_delta_factor, double dash_power_rate_delta_min, double dash_power_rate_delta_max, double player_size_delta_factor, double kickable_margin_delta_min, double kickable_margin_delta_max, double kick_rand_delta_factor, double extra_stamina_delta_min, double extra_stamina_delta_max, double effort_max_delta_factor, double effort_min_delta_factor)

  This function is called after parsing a version 7 player parameter message.

- virtual void doBuildPlayerParam ()

  This function is called after parsing a version 8 or 9 player parameter message.

- virtual void doBuildPlayerType (int id, double player_speed_max, double stamina_inc_max, double player_decay, double inertia_moment, double dash_power_rate, double player_size, double kickable_margin, double kick_rand, double extra_stamina, double effort_max, double effort_min)

  This function is called after parsing a version 7 player type message.

- virtual void doBuildPlayerType ()

  This function is called after parsing a version 8 or 9 player type message.

Player Visual Parsing Functions

- virtual void doBuildVisual (int time)

  This function is called after parsing a player visual message.

- virtual void doBuildLine (double dist, double dir)

  This function is called after parsing a line section of player visual message, when the client is in a high quality view mode.

- virtual void doBuildLine (double dir)

  This function is called after parsing a line section of player visual message, when the client is in a low quality view mode.

- virtual void doBuildFlag (bool close, double dist, double dir, double dist_chg, double dir_chg)

  This function is called after parsing a flag section of player visual message, when the client is in a high quality view mode, the flag is not far away and doBuildFlag(double dist, double dir, double dist_chg, double dir_chg) has not been overridden.

- virtual void doBuildFlag (double dist, double dir, double dist_chg, double dir_chg)

  This function is called after parsing a flag section of player visual message, when the client is in a high quality view mode and the flag is not far away.

- virtual void doBuildFlag (bool close, double dist, double dir)

  This function is called after parsing a flag section of player visual message, when the client is in a high quality view mode and the flag is close or far away.

- virtual void doBuildFlag (bool close, double dir)

  This function is called after parsing a flag section of player visual message, when the client is in a low quality view mode and the flag is close or far away.

- virtual void doBuildGoal (bool close, double dist, double dir, double dist_chg, double dir_chg)

  This function is called after parsing a goal section of player visual message, when the client is in a high quality view mode, the goal is not far away and doBuildGoal(double dist, double dir, double dist_chg, double dir_chg) has not been overridden.

- virtual void doBuildGoal (double dist, double dir, double dist_chg, double dir_chg)

  This function is called after parsing a goal section of player visual message, when the client is in a high quality view mode and the goal is not far away.

- virtual void doBuildGoal (bool close, double dist, double dir)

  This function is called after parsing a goal section of player visual message, when the client is in a high quality view mode and the goal is close or far away.

- virtual void doBuildGoal (bool close, double dir)

  This function is called after parsing a goal section of player visual message, when the client is in a low quality view mode and the goal is close or far away.

- virtual void doBuildBall (bool close, double dist, double dir, double dist_chg, double dir_chg)

  This function is called after parsing a ball section of player visual message, when the client is in a high quality view mode, the ball is not far away and doBuild-PlayerVisBall(double dist, double dir, double dist_chg, double dir_chg) has not been overridden.

- virtual void doBuildPlayerVisBall (double dist, double dir, double dist_chg, double dir_chg)

  This function is called after parsing a ball section of player visual message, when the client is in a high quality view mode and the ball is not far away.

- virtual void doBuildBall (bool close, double dist, double dir)

  This function is called after parsing a ball section of player visual message, when the client is in a high quality view mode and the ball is close or far away.

- virtual void doBuildBall (bool close, double dir)

  This function is called after parsing a ball section of player visual message, when the client is in a low quality view mode and the ball is close or far away.

- virtual void doBuildPlayer (bool close, double dist, double dir, double dist_-chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)

  This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is not far away, is pointing and doBuildPlayer(double dist, double dir, double dist_-chg, double dir_chg, double orientation, double head_orientation, double point_-dir, bool tackling) has not been overridden.

- virtual void doBuildPlayer (double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)

  This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is not far away and is pointing.

- virtual void doBuildPlayer (bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)

  This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is not far away, is not pointing or the client is using protcol version 7 and doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling) has not been overridden.

- virtual void doBuildPlayer (double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)

  This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is not far away and the player is not pointing or the client is using protocol 7.

- virtual void doBuildPlayer (bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)

  This function never called.

- virtual void doBuildPlayer (double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)

  This function never called.

- virtual void doBuildPlayer (bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)

  This function never called.

- virtual void doBuildPlayer (double dist, double dir, double dist_chg, double dir_chg, bool tackling)

  This function never called.

- virtual void doBuildPlayer (bool close, double dist, double dir, double point_dir, bool tackling)

  This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is far away, is pointing and doBuildPlayer(double dist, double dir, double point_dir, bool tackling) has not been overridden.

- virtual void doBuildPlayer (double dist, double dir, double point_dir, bool tackling)

  This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is far away and is pointing.

- virtual void doBuildPlayer (bool close, double dist, double dir, bool tackling)

  This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is far away and not pointing or the player is very far away and the client is using protocol 7, or the player is very close, but outside of the view cone.

- virtual void doBuildPlayer (bool close, double dist, double dir, double dist_-chg, double dir_chg, double orientation, double head_orientation)

  This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is not far away, is not pointing or the client is using protcol version 7 and doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling), and doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling) have not been overridden.

- virtual void doBuildPlayer (bool close, double dist, double dir, double dist_-chg, double dir_chg)

  This function never called.

- virtual void doBuildPlayer (bool close, double dist, double dir)

  This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is far away and not pointing or the player is very far away and the client is using protocol 7, or the player is very close, but outside of the view cone, and doBuildPlayer(bool close, double dist, double dir, bool tackling) has not been overridden.

- virtual void doBuildPlayer (bool close, double dir)

  This function is called after parsing a player section of player visual message, when the client is in a low quality view mode.

Coach Visual Parsing Functions

- virtual void doBuildGlobalVisual (int time)

  This function is called after parsing a coach visual message.

- virtual void doBuildGlobalGoal (double x, double y)

  This function is called after parsing a goal section of a coach visual message.

- virtual void doBuildGlobalBall (double x, double y, double delta_x, double delta_y)

  This function is called after parsing a ball section of a coach visual message.

- virtual void doBuildGlobalPlayer (double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, double point_dir, bool tackle)

  This function is called after parsing a player section of a coach visual message, where the player is pointing.

- virtual void doBuildGlobalPlayer (double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, bool tackle)
  This function is called after parsing a non-pointing player section of a coach visual message.
- virtual void doBuildGlobalPlayer (double x, double y, double delta_x, double delta_y, double orientation, double head_orientation)
  This function is called after parsing a player section of a coach visual message.

Field Location Parsing Functions

- virtual void doBuildFlagOffset (int offset)
  This function is called after parsing a flag offset section of a flag section of a player visual message.
- virtual void doBuildTopLocation ()
  This function is called after parsing a top location of a field object or a visual.
- virtual void doBuildBottomLocation ()
  This function is called after parsing a bottom location of a field object or a visual.
- virtual void doBuildLeftLocation ()
  This function is called after parsing a left location of a field object or a visual.
- virtual void doBuildRightLocation ()
  This function is called after parsing a right location of a field object or a visual.
- virtual void doBuildCenterLocation ()
  This function is called after parsing a center location of a field object or a visual.
- virtual void doBuildPenaltyLocation ()
  This function is called after parsing a penalty location of a field object or a visual.
- virtual void doBuildGoalLocation ()
  This function is called after parsing a goal location of a field object or a visual.

Initialisation Parsing Functions

- virtual void doBuildInit (int unum)
  This function is called after parsing a player init message.
- virtual void doBuildInit ()
  This function is called after parsing a player reconnect message.
- virtual void doBuildCoachInit ()
  This function is called after parsing an coach init message.

Audio Parsing Functions

- virtual void doBuildCoachAudio (int time, const std::string &msg)

  This function is called after parsing an offline coach or non-CLang online coach audio message.

- virtual void doBuildRefAudio (int time)

  This function is called after parsing a non-goal referee message.

- virtual void doBuildGoalRefAudio (int time, int score)

  This function is called after parsing a goal referee message.

- virtual void doBuildUnknownRefAudio (int time, const std::string &message)

  This function is called after parsing an unknown referee message.

- virtual void doBuildPlayerAudio (int time, double dir, const std::string &msg)

  This function is called after parsing a version 7 player audio message or a version 8 opponent player audio message, where the client is a field player.

- virtual void doBuildPlayerAudio (int time, const std::string &msg)

  This function is called after parsing a player audio message, where the client is an online coach.

- virtual void doBuildPlayerAudio (int time, double dir, int unum, const std::string &msg)

  This function is called after parsing a version 8 team-mate player audio message, where the client is a field player.

- virtual void doBuildPlayerAudio (int time, int unum)

  This function is called after parsing a version 8 team-mate player audio message, where the client is a field player and the client has already received a team-mate player audio message this cycle.

- virtual void doBuildPlayerAudio (int time)

  This function is called after parsing a version 8 opponent player audio message, where the client is a field player and the client has already received an opponent player audio message this cycle.

- virtual void doBuildSelfAudio (int time, const std::string &msg)

  This function is called after parsing a player audio message, where the client is a field player and the message was sent by the client.

Substitution Parsing Functions

- virtual void doBuildSubstitution (int unum, int type)

  This function is called after parsing a substitutution message, where the player being sub'ed is a team-mate.

- virtual void doBuildSubstitution (int unum)

  This function is called after parsing a substitutution message, where the player being sub'ed is an opponent.

Clang version parsing functions

- virtual void doBuildClangPlayerVersionMsg ()
  This function is called after parsing a clang player version message (where the player reports what version of clang it supports).
- virtual void doBuildClangPlayerVersion (int min, int max)
  This function is called after parsing a player clang version token (where the player reports what version of clang it supports).

Error Parsing Functions

- virtual void doBuildCantReconnect ()
  This function is called after parsing a can't reconnect error message.
- virtual void doBuildInitError ()
  This function is called after parsing a reconnection error message.
- virtual void doBuildNoMoreTeamOrPlayerOrGoalieError ()
  This function is called after parsing a player initialisation error message.
- virtual void doBuildNoSuchTeamOrAlreadyHaveCoachError ()
  This function is called after parsing an online coach initialisation error message.
- virtual void doBuildTooManyMovesError ()
  This function is called after parsing a too many moves error message.
- virtual void doBuildUnknownCommandError ()
  This function is called after parsing an unknown command error message.
- virtual void doBuildIllegalCommandError ()
  This function is called after parsing an illegal command form error message.
- virtual void doBuildSayMessageTooLongError ()
  This function is called after parsing a say too long error message.
- virtual void doBuildIllegalModeError ()
  This function is called after parsing an illegal mode error message.
- virtual void doBuildIllegalObjectFormError ()
  This function is called after parsing an illegal object form error message.
- virtual void doBuildSaidTooManyFreeformMessagesError ()
  This function is called after parsing a said too many freeform error message.
- virtual void doBuildCannotSayFreeformWhilePlayonError ()
  This function is called after parsing a cannot say freeform error message.

- virtual void doBuildSaidTooManyMetaMessagesError ()
  This function is called after parsing a said too many meta error message.
- virtual void doBuildSaidTooManyAdviceMessagesError ()
  This function is called after parsing a said too many advice error message.
- virtual void doBuildSaidTooManyInfoMessagesError ()
  This function is called after parsing a said too many info error message.
- virtual void doBuildSaidTooManyDefineMessagesError ()
  This function is called after parsing a said too many define error message.
- virtual void doBuildCouldNotParseSayError ()
  This function is called after parsing a cound not parse say error message.
- virtual void doBuildSaidTooManyMessagesError ()
  This function is called after parsing a said too many error message.
- virtual void doBuildUnknownError (const std::string &error)
  This function is called after parsing an unknown error message.

Warning Parsing Functions

- virtual void doBuildNoTeamFoundWarning ()
  This function is called after parsing a `no_team_found` *warning message.*
- virtual void doBuildNoSuchPlayerWarning ()
  This function is called after parsing a `no_such_player` *warning message.*
- virtual void doBuildCannotSubWhilePlayOnWarning ()
  This function is called after parsing a `cannot_sub_while_playon` *warning message.*
- virtual void doBuildNoSubsLeftWarning ()
  This function is called after parsing a `no_subs_left` *warning message.*
- virtual void doBuildMaxOfThatPlayerTypeOnFieldWarning ()
  This function is called after parsing a `max_of_that_type_on_field` *warning message.*
- virtual void doBuildCannotChangeGoalieWarning ()
  This function is called after parsing a `cannot_change_goalie` *warning message.*
- virtual void doBuildCompressionWarning ()
  This function is called after parsing a `compression_unsupported` *warning message.*
- virtual void doBuildUnknownWarning (const std::string &warning)
  This function is called after parsing an unknown warning message.

OK Parsing Functions

- virtual void doBuildClangVerOK (int min, int max)
  This function is called after parsing an `clang ver` *OK message.*
- virtual void doBuildEyeOK (bool on)
  This function is called after parsing an `eye` *OK message.*
- virtual void doBuildSayOK ()
  This function is called after parsing an `say` *OK message.*
- virtual void doBuildChangePlayerTypeOK (int unum, int type)
  This function is called after parsing an `change_player_type` *OK message.*
- virtual void doBuildCompressionOK (int level)
  This function is called after parsing an `compression` *OK message.*
- virtual void doBuildUnknownOK (const std::string &msg)
  This function is called after parsing an unknown OK message.

Friends

- class **Param**
- class **RCCLexer**
- int **RCC_lex** (rcc::Holder ∗, rcc::Parser::Param &param)
- rcc::Parser::Param & **getParam** (void ∗)

7.1.1 Detailed Description

The parsing base class.

This is the main class provided by the RCCParser library. Subclass it and overide the virtual functions to provide custom behaviour with the parsed data during parsing. The default behaviour is to do nothing with the parsed data.

Definition at line 454 of file rccparser.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 rcc::Parser::Parser (rcss::Parser & *clang_parser*) `[inline]`

Parser constructor.

You will need to call this constuctor from your subclass's contructor. The clang_-parser parameter is a reference to another parser which will be used for parsing CLang messages. This is compatible with the CLang parser that is part of the simulator.

Definition at line 1645 of file rccparser.h.

7.1.3 Member Function Documentation

7.1.3.1 virtual void rcc::Parser::doBuildAfterGoalLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `goal_l` play mode.

Override this function in your subclass to handle the `goal_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `goal_l` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2665 of file rccparser.h.

7.1.3.2 virtual void rcc::Parser::doBuildAfterGoalRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `goal_r` play mode.

Override this function in your subclass to handle the `goal_r` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `goal_r` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2688 of file rccparser.h.

7.1.3.3 virtual void rcc::Parser::doBuildArmState (int *movable_in*, int *expires_in*, double *target_x*, double *target_y*, int *count*) `[inline, protected, virtual]`

This function is called after parsing an arm section of a version 8 or 9 sense body message or a version 8 or 9 full state message.

Override this function in your subclass to handle the arm section of version 8 or 9 sense body messages and version 8 or 9 full state messages.

Parameters:

movable_in The number of cycles until the client or player can change where it's pointing to.

expires_in The number of cycles until the client or player will automatically stop pointing.

target_x The x co-ordinate of the point the client or player is pointing to.

target_y The y co-ordinate of the point the client or player is pointing to.

count The number of times the client or player has excuted a point to command.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

Either:

- The arm section of a sense body message has been parsed or
- Full state message sending is turned on in the simulator for the team this client is in and the arm section of a full state message has just be parsed.

The following is an example of the arm section of either a version 8 or 9 sense body message or a version 8 or 9 full state message:

```
(arm (movable 3) (expires 13) (target -10 36) (count 15))
```

In this examples the:

- *movable_in* parameter would be 3.
- *expires_in* parameter would be 13.
- *target_x* parameter would be -10.
- *target_y* parameter would be 36.
- *count* parameter would be 15.

See also:

- doBuildFullState(int time)

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)

Definition at line 4139 of file rccparser.h.

7.1.3.4 virtual void rcc::Parser::doBuildBackPassLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `back_pass_l` play mode.

Override this function in your subclass to handle the `back_pass_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `back_pass_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3125 of file rccparser.h.

7.1.3.5 virtual void rcc::Parser::doBuildBackPassRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `back_pass_r` play mode.

Override this function in your subclass to handle the `back_pass_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `back_pass_r` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3148 of file rccparser.h.

7.1.3.6 virtual void rcc::Parser::doBuildBall (bool *close*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a ball section of player visual message, when the client is in a low quality view mode and the ball is close or far away.

Override this function in your subclass to handle low quality close or far ball sections of player visual messages.

Parameters:
close false if the ball is in the client's view cone, true otherwise.

dist The distance (in meters) from the client to the ball.

dir The angle from the client's center of view to the ball.

Precondition:
The client is a field player.

Precondition:
The client is in a low quality view mode.

Precondition:
An entire ball section of a player visual message has been parsed.

Precondition:
A player visual message is being parsed.

The following is an example of a far ball section of a player visual message:

```
((b) -37)
```

In this example the:

- *close* parameter would be false.
- *dir* parameter would be -37.

The following is an example of a close ball section of a player visual message:

```
((B) 46)
```

In this example the:

- *close* parameter would be true.
- *dir* parameter would be 46.

See also:

- doBuildBall(bool close, double dist, double dir, double dist_chg, double dir_-chg)
- doBuildPlayerVisBall(double dist, double dir, double dist_chg, double dir_-chg)
- doBuildBall(bool close, double dist, double dir)
- doBuildVisual(int time)

Definition at line 6268 of file rccparser.h.

7.1.3.7 virtual void rcc::Parser::doBuildBall (bool *close*, double *dist*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a ball section of player visual message, when the client is in a high quality view mode and the ball is close or far away.

Override this function in your subclass to handle high quality close or far ball sections of player visual messages.

Parameters:

close false if the ball is in the client's view cone, true otherwise.

dist The distance (in meters) from the client to the ball.

dir The angle from the client's center of view to the ball.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire ball section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a far ball section of a player visual message:

```
((b) 27.1 -37)
```

In this example the:

- *close* parameter would be false.
- *dist* parameter would be 27.1.
- *dir* parameter would be -37.

The following is an example of a close ball section of a player visual message:

```
((B) 2.7 46)
```

In this example the:

- *close* parameter would be true.
- *dist* parameter would be 2.7.
- *dir* parameter would be 46.

See also:

- doBuildBall(bool close, double dist, double dir, double dist_chg, double dir_-chg)
- doBuildPlayerVisBall(double dist, double dir, double dist_chg, double dir_-chg)
- doBuildBall(double dist, double dir)
- doBuildVisual(int time)

Definition at line 6221 of file rccparser.h.

7.1.3.8 virtual void rcc::Parser::doBuildBall (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*) `[inline, protected, virtual]`

This function is called after parsing a ball section of player visual message, when the client is in a high quality view mode, the ball is not far away and doBuildPlayerVisBall(double dist, double dir, double dist_chg, double dir_chg) has not been overridden.

Override this function in your subclass to handle high quality near ball sections of player visual messages.

Warning:

This function is deprecated. Overide doBuildPlayerVisBall(double dist, double dir, double dist_chg, double dir_chg) instead.

Parameters:

close false if the ball is in the client's view cone, true otherwise. Since dist_chg and dir_chg are only ever sent if the ball is in the client's view cone, *close* is always false, thus this parameter is redundant.

dist The distance (in meters) from the client to the ball.

dir The angle from the client's center of view to the ball.

dist_chg The distance change of the ball relative to the player.

dir_chg This is the direction change of the ball relative to the client.

Precondition:
The client is a field player.

Precondition:
The client is in a high quality view mode.

Precondition:
An entire ball section of a player visual message has been parsed.

Precondition:
A player visual message is being parsed.

The following is an example of a near ball section of a player visual message:

```
((b) 24.5 -36 0.49 -0.8)
```

In this example the:

- *close* parameter would be false.
- *dist* parameter would be 24.5.
- *dir* parameter would be -36.
- *dist_chg* parameter would be 0.49.
- *dir_chg* parameter would be -0.8.

See also:

- doBuildPlayerVisBall(double dist, double dir, double dist_chg, double dir_-chg)
- doBuildBall(bool close, double dist, double dir)
- doBuildBall(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6120 of file rccparser.h.

7.1.3.9 virtual void rcc::Parser::doBuildBall (double *x*, double *y*, double *delta_x*, double *delta_y*) `[inline, protected, virtual]`

This function is called after parsing the ball section of a full state message.

Override this function in your subclass to handle the ball section of a full state message.

Parameters:

x The x co-ordinate of the ball.

y The y co-ordinate of the ball.

delta_x The x velocity of the ball.

delta_y The y velocity of the ball.

Precondition:
The client is a field player.

Precondition:
Full state message sending is turned on in the simulator for the team this client is in.

Precondition:
The ball section of the full state message has been parsed.

The following is a protocol version 8 or 9 example of the score section of a full state message:

```
((b) -20.5 10.7 0.56 -0.3)
```

The following is the same exmaple in protocol version 7:

```
(ball -20.5 10.7 0.56 -0.3)
```

In these examples the:

- *x* parameter would be -20.5.
- *y* parameter would be 10.7.
- *delta_x* parameter would be 0.56.
- *delta_y* parameter would be -0.3.

See also:
- doBuildFullState(int time)

Definition at line 1994 of file rccparser.h.

Referenced by doBuildPlayerVisBall().

7.1.3.10 virtual void rcc::Parser::doBuildBeforeKickOffPlayMode () `[inline, protected, virtual]`

This function is called after parsing a `before_kick_off` play mode.

Override this function in your subclass to handle the `before_kick_off` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `before_kick_off` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2366 of file rccparser.h.

7.1.3.11 virtual void rcc::Parser::doBuildBottomLocation () `[inline, protected, virtual]`

This function is called after parsing a bottom location of a field object or a visual.

Override this function in your subclass to handle bottom field locations.

Precondition:
A bottom loation of a field object has been parsed.

Precondition:
A field object is being parsed.

Precondition:
A a visual message is being parsed.

Definition at line 7481 of file rccparser.h.

7.1.3.12 virtual void rcc::Parser::doBuildCannotChangeGoalieWarning () `[inline, protected, virtual]`

This function is called after parsing a `cannot_change_goalie` warning message.

Override this function in your subclass to handle `cannot_change_goalie` warning messages.

Precondition:
The client is an online coach

Precondition:
The client sent a `change_player_type` but the player specified was a goalie.

Precondition:
An entire `max_of_that_type_on_field` warning message has been parsed.

Definition at line 8420 of file rccparser.h.

7.1.3.13 virtual void rcc::Parser::doBuildCannotSayFreeformWhilePlayonError () `[inline, protected, virtual]`

This function is called after parsing a cannot say freeform error message.

Override this function in your subclass to handle cannot say freeform error messages.

Precondition:
The client is an online coach

Precondition:
The client has sent a freeform CLang message and the playmode is playon.

Precondition:
An entire cannot say freeform error message has been parsed.

Definition at line 8166 of file rccparser.h.

7.1.3.14 virtual void rcc::Parser::doBuildCannotSubWhilePlayOnWarning () `[inline, protected, virtual]`

This function is called after parsing a `cannot_sub_while_playon` warning message.

Override this function in your subclass to handle `cannot_sub_while_playon` warning messages.

Precondition:
The client is an online coach

Precondition:
The client sent a `change_player_type` command during playon

Precondition:
An entire `cannot_sub_while_playon` warning message has been parsed.

Definition at line 8361 of file rccparser.h.

7.1.3.15 virtual void rcc::Parser::doBuildCantReconnect () `[inline, protected, virtual]`

This function is called after parsing a can't reconnect error message.

Override this function in your subclass to handle can't reconnect error messages.

Precondition:
The client is a field player

Precondition:
The client has sent a reconnect message.

Precondition:
The server is in play-on playmode

Precondition:
An entire can't reconnect error message has been parsed.

Definition at line 7934 of file rccparser.h.

7.1.3.16 virtual void rcc::Parser::doBuildCatchBallLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing a `goalie_catch_ball_l` play mode.

Override this function in your subclass to handle the `goalie_catch_ball_l` play mode.

\note: This is and `goalie_catch_ball_r` are only ever sent as part full state message.

Precondition:
A full state message is being parsed.

Precondition:
A `goalie_catch_ball_l` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildCatchBallRightPlayMode()

Definition at line 2322 of file rccparser.h.

7.1.3.17 virtual void rcc::Parser::doBuildCatchBallRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing a `goalie_catch_ball_r` play mode.

Override this function in your subclass to handle the `goalie_catch_ball_r` play mode.

\note: This is and `goalie_catch_ball_l` are only ever sent as part full state message.

Precondition:
A full state message is being parsed.

Precondition:
A `goalie_catch_ball_r` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildCatchBallLeftPlayMode()

Definition at line 2343 of file rccparser.h.

7.1.3.18 virtual void rcc::Parser::doBuildCatchFaultLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `catch_fault_l` play mode.

Override this function in your subclass to handle the `catch_fault_l` play mode.

Precondition:
Either:
- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `catch_fault_l` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3216 of file rccparser.h.

7.1.3.19 virtual void rcc::Parser::doBuildCatchFaultRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `catch_fault_r` play mode.

Override this function in your subclass to handle the `catch_fault_r` play mode.

Precondition:
Either:
- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `catch_fault_r` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3238 of file rccparser.h.

7.1.3.20 virtual void rcc::Parser::doBuildCenterLocation () `[inline, protected, virtual]`

This function is called after parsing a center location of a field object or a visual.

Override this function in your subclass to handle center field locations.

Precondition:
A center loation of a field object has been parsed.

Precondition:
A field object is being parsed.

Precondition:
A a visual message is being parsed.

Definition at line 7532 of file rccparser.h.

7.1.3.21 virtual void rcc::Parser::doBuildChangePlayerTypeOK (int *unum*, int *type*) `[inline, protected, virtual]`

This function is called after parsing an `change_player_type` OK message.

Override this function in your subclass to handle `change_player_type` OK messages.

Precondition:
The client is an online or offline coach

Precondition:
The client sent a `change_player_type` command

Precondition:
An entire `change_player_type` OK message has been parsed.

Definition at line 8535 of file rccparser.h.

7.1.3.22 virtual void rcc::Parser::doBuildClangPlayerVersion (int *min*, int *max*) `[inline, protected, virtual]`

This function is called after parsing a player clang version token (where the player reports what version of clang it supports).

Override this function in your subclass to handle clang player version messages

Precondition:
The client is an online coach

Precondition:
A player specification has been parsed

Definition at line 7910 of file rccparser.h.

7.1.3.23 virtual void rcc::Parser::doBuildClangPlayerVersionMsg () `[inline, protected, virtual]`

This function is called after parsing a clang player version message (where the player reports what version of clang it supports).

Override this function in your subclass to handle clang player version messages

Precondition:
The client is an online coach

Precondition:
An entire clang player version message has been parsed

Precondition:
zero of more clang player version tokens have been parsed

Definition at line 7896 of file rccparser.h.

7.1.3.24 virtual void rcc::Parser::doBuildClangVerOK (int *min*, int *max*) `[inline, protected, virtual]`

This function is called after parsing an `clang ver` OK message.

Override this function in your subclass to handle `clang` \v ver OK messages.

Precondition:
The client is a field player

Precondition:
The client sent a `clang ver` command

Precondition:
An entire `clang ver` OK message has been parsed.

Definition at line 8481 of file rccparser.h.

7.1.3.25 virtual void rcc::Parser::doBuildCoachAudio (int *time*, const std::string & *msg*) `[inline, protected, virtual]`

This function is called after parsing an offline coach or non-CLang online coach audio message.

Override this function in your subclass to handle offline or non-CLang online coach audio messages.

Precondition:
An entire offline coach or non-Clang online coach audio message has been parsed.

Definition at line 7676 of file rccparser.h.

7.1.3.26 virtual void rcc::Parser::doBuildCoachInit () `[inline, protected, virtual]`

This function is called after parsing an coach init message.

Override this function in your subclass to handle player coach init messages

Precondition:
The client is an coach

Precondition:
An entire coach init message has been parsed.

Definition at line 7657 of file rccparser.h.

7.1.3.27 virtual void rcc::Parser::doBuildCompressionOK (int *level*) `[inline, protected, virtual]`

This function is called after parsing an `compression` OK message.

Override this function in your subclass to handle `compression` OK messages.

Precondition:
The client sent a `compression` command

Precondition:
An entire `compression` OK message has been parsed.

Definition at line 8551 of file rccparser.h.

7.1.3.28 virtual void rcc::Parser::doBuildCompressionWarning () `[inline, protected, virtual]`

This function is called after parsing a `compression_unsupported` warning message.

Override this function in your subclass to handle `compression_unsupported` warning messages.

Precondition:
The client sent a `compression` command, but the server was compiled without compression support.

Precondition:
An entire `compression` command warning message has been parsed.

Definition at line 8438 of file rccparser.h.

7.1.3.29 virtual void rcc::Parser::doBuildCornerKickLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `corner_kick_l` play mode.

Override this function in your subclass to handle the `corner_kick_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `corner_kick_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2573 of file rccparser.h.

7.1.3.30 virtual void rcc::Parser::doBuildCornerKickRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `corner_kick_r` play mode.

Override this function in your subclass to handle the `corner_kick_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `corner_kick_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2596 of file rccparser.h.

7.1.3.31 virtual void rcc::Parser::doBuildCouldNotParseSayError () `[inline, protected, virtual]`

This function is called after parsing a cound not parse say error message.

Override this function in your subclass to handle cound not parse say error messages.

Precondition:
The client is an online coach

Precondition:
The client has sent a malformed CLang message

Precondition:
An entire cound not parse say error message has been parsed.

Definition at line 8260 of file rccparser.h.

7.1.3.32 virtual void rcc::Parser::doBuildCounts (int *kick*, int *dash*, int *turn*, int *katch*, int *move*, int *turn_neck*, int *change_view*, int *say*) `[inline, protected, virtual]`

This function is called after parsing the count section of a version 8 or 9 full state message.

Override this function in your subclass to handle the count section of a version 8 or 9 full state message.

Parameters:

kick The number of kick commands executed by the player.

dash The number of dash commands executed by the player.

turn The number of turn commands executed by the player.

katch The number of catch commands executed by the player.

move The number of move commands executed by the player.

turn_neck The number of turn_neck commands executed by the player.

change_view The number of change_view commands executed by the player.

say The number of say commands executed by the player.

Precondition:
The client is using protocol version 8 or 9.

Precondition:
The client is a field player.

Precondition:
Full state message sending is turned on in the simulator for the team this client is in.

Precondition:
The count section of the full state message has been parsed.

The following is an example of the count section of a full state message:

```
(count 5 2 3 9 4 6 7 1)
```

In this example the:

- *kick* parameter would be 5.
- *dash* parameter would be 2.
- *turn* parameter would be 3.
- *katch* parameter would be 9.
- *move* parameter would be 4.
- *turn_neck* parameter would be 6.
- *change_view* parameter would be 7.
- *say* parameter would be 1.

See also:
- doBuildFullState(int time)

Definition at line 1920 of file rccparser.h.

7.1.3.33 virtual void rcc::Parser::doBuildDropBallPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `drop_ball` play mode.

Override this function in your subclass to handle the `drop_ball` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `drop_ball` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2711 of file rccparser.h.

7.1.3.34 virtual void rcc::Parser::doBuildEyeOK (bool *on*) `[inline, protected, virtual]`

This function is called after parsing an `eye` OK message.

Override this function in your subclass to handle `eye` OK messages.

Precondition:
The client is an online or offline coach

Precondition:
The client sent a `eye` command

Precondition:
An entire `eye` OK message has been parsed.

Definition at line 8499 of file rccparser.h.

7.1.3.35 virtual void rcc::Parser::doBuildFirstHalfOverPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `first_half_over` play mode.

Override this function in your subclass to handle the `first_half_over` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `first_half_over` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2826 of file rccparser.h.

7.1.3.36 virtual void rcc::Parser::doBuildFlag (bool *close*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a flag section of player visual message, when the client is in a low quality view mode and the flag is close or far away.

Override this function in your subclass to handle low quality close or far flag sections of player visual messages.

Parameters:

close false if the flag is in the client's view cone, true otherwise.

dir The angle from the client's center of view to the flag.

Precondition:

The client is a field player.

Precondition:

The client is in a low quality view mode.

Precondition:

An entire flag section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a far flag section of a player visual message:

```
((f l t 20) -31)
```

In this example the:

- *dir* parameter would be -31.

The following is an example of a close flag section of a player visual message:

```
((F) 80)
```

In this example the:

- *dir* parameter would be 80.

See also:

- doBuildFlag(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(bool close, double dist, double dir)
- doBuildVisual(int time)
- doBuildFlagOffset(int offset)
- doBuildTopLocation()
- doBuildBottomLocation()
- doBuildLeftLocation()
- doBuildRightLocation()
- doBuildCenterLocation()
- doBuildPenaltyLocation()
- doBuildGoalLocation()

Definition at line 5861 of file rccparser.h.

7.1.3.37 virtual void rcc::Parser::doBuildFlag (bool *close*, double *dist*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a flag section of player visual message, when the client is in a high quality view mode and the flag is close or far away.

Override this function in your subclass to handle high quality close or far flag sections of player visual messages.

Parameters:

close false if the flag is in the client's view cone, true otherwise.

dist The distance (in meters) from the client to the flag.

dir The angle from the client's center of view to the flag.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire flag section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a far flag section of a player visual message:

```
((f l t 20) 67.4 -31)
```

In this example the:

- *dist* parameter would be 67.4.
- *dir* parameter would be -31.

The following is an example of a close flag section of a player visual message:

```
((F) 2.3 80)
```

In this example the:

- *dist* parameter would be 2.3.
- *dir* parameter would be 80.

See also:

- doBuildFlag(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(double dist, double dir, double dist_chg, double dir_chg)

- doBuildFlag(bool close, double dir)
- doBuildVisual(int time)
- doBuildFlagOffset(int offset)
- doBuildTopLocation()
- doBuildBottomLocation()
- doBuildLeftLocation()
- doBuildRightLocation()
- doBuildCenterLocation()
- doBuildPenaltyLocation()
- doBuildGoalLocation()

Definition at line 5809 of file rccparser.h.

7.1.3.38 virtual void rcc::Parser::doBuildFlag (double *dist*, double *dir*, double *dist_chg*, double *dir_chg*) `[inline, protected, virtual]`

This function is called after parsing a flag section of player visual message, when the client is in a high quality view mode and the flag is not far away.

Override this function in your subclass to handle high quality near flag sections of player visual messages.

Note:

This function deprecates doBuildFlag(bool close, double dist, double dir, double dist_chg, double dir_chg).

Parameters:

dist The distance (in meters) from the client to the flag.

dir The angle from the client's center of view to the flag.

dist_chg The distance change of the flag relative to the player. Since the flag is stationary this can be used with *dir_chg* to augment the client's velocity information found in the sense body messages.

dir_chg This is the direction change of the flag relative to the client.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire flag section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a near flag section of a player visual message:

```
((f t 0) 26.8 33 0 0.7)
```

In this example the:

- *dist* parameter would be 26.8.
- *dir* parameter would be 33.
- *dist_chg* parameter would be 0.
- *dir_chg* parameter would be 0.7.

See also:

- doBuildFlag(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(bool close, double dist, double dir)
- doBuildFlag(bool close, double dir)
- doBuildVisual(int time)
- doBuildFlagOffset(int offset)
- doBuildTopLocation()
- doBuildBottomLocation()
- doBuildLeftLocation()
- doBuildRightLocation()
- doBuildCenterLocation()
- doBuildPenaltyLocation()
- doBuildGoalLocation()

Definition at line 5750 of file rccparser.h.

7.1.3.39 virtual void rcc::Parser::doBuildFlag (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*) `[inline, protected, virtual]`

This function is called after parsing a flag section of player visual message, when the client is in a high quality view mode, the flag is not far away and doBuildFlag(double dist, double dir, double dist_chg, double dir_chg) has not been overridden.

Override this function in your subclass to handle high quality near flag sections of player visual messages.

Warning:

This function is deprecated. Overide doBuildFlag(double dist, double dir, double dist_chg, double dir_chg) instead.

Parameters:

close false if the flag is in the client's view cone, true otherwise. Since dist_chg and dir_chg are only ever sent if the flag is in the client's view cone, close is always false, thus this parameter is redundant.

dist The distance (in meters) from the client to the flag.

dir The angle from the client's center of view to the flag.

dist_chg The distance change of the flag relative to the player. Since the flag is stationary this can be used with *dir_chg* to augment the client's velocity information found in the sense body messages.

dir_chg This is the direction change of the flag relative to the client.

Precondition:
The client is a field player.

Precondition:
The client is in a high quality view mode.

Precondition:
An entire flag section of a player visual message has been parsed.

Precondition:
A player visual message is being parsed.

The following is an example of a near flag section of a player visual message:

```
((f t 0) 26.8 33 0 0.7)
```

In this example the:

- *close* parameter would be false.
- *dist* parameter would be 26.8.
- *dir* parameter would be 33.
- *dist_chg* parameter would be 0.
- *dir_chg* parameter would be 0.7.

See also:
- doBuildFlag(double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(bool close, double dist, double dir)
- doBuildFlag(bool close, double dir)
- doBuildVisual(int time)
- doBuildFlagOffset(int offset)
- doBuildTopLocation()
- doBuildBottomLocation()
- doBuildLeftLocation()
- doBuildRightLocation()
- doBuildCenterLocation()
- doBuildPenaltyLocation()
- doBuildGoalLocation()

Definition at line 5694 of file rccparser.h.

Referenced by doBuildFlag().

7.1.3.40 virtual void rcc::Parser::doBuildFlagOffset (int *offset*) `[inline, protected, virtual]`

This function is called after parsing a flag offset section of a flag section of a player visual message.

Override this function in your subclass to handle flag offsets of flags sections of player visual messages.

Parameters:

offset The Flags offset, which helps specify it's location.

Precondition:

The client is a field player.

Precondition:

A flag offset of a flag section of a player visual message has been parsed.

Precondition:

A flag section of a player visual message is being parsed.

Precondition:

A a player visual message is being parsed.

See also:

- doBuildFlag(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(bool close, double dist, double dir)
- doBuildFlag(bool close, double dir) {}

Definition at line 7445 of file rccparser.h.

7.1.3.41 virtual void rcc::Parser::doBuildFocusState (int *unum*, int *count*) `[inline, protected, virtual]`

This function is called after parsing a focus state section of a version 8 or 9 sense body, where the client **is** focused on a player.

Override this function in your subclass to handle the focused focus state section of version 8 or 9 sense body messages.

Parameters:

unum The uniform number of the player the client is focued on.

count The number of times the client has excuted an attention to command.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

A version 8 or 9 sense body message is being parsed.

Precondition:

A focused focus state section of a version 8 or 9 sense body message has just been parsed.

The following is an example of the focused focus section of a version 8 or 9 sense body message:

```
(focus (target l 9) (count 18))
```

In this examples the:

- *unum* parameter would be 9.
- *count* parameter would be 18.

See also:

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildFocusState(int count)

Definition at line 4266 of file rccparser.h.

7.1.3.42 virtual void rcc::Parser::doBuildFocusState (int *count*) `[inline, protected, virtual]`

This function is called after parsing an focus state section of a version 8 or 9 sense body, where the client is **not** focused on any player.

Override this function in your subclass to handle the non-focused focus state section of version 8 or 9 sense body messages.

Parameters:

count The number of times the client has excuted an attention to command.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

A version 8 or 9 sense body message is being parsed.

Precondition:

A non-focused focus state section of a version 8 or 9 sense body message has just been parsed.

The following is an example of the non-focused focus section of a version 8 or 9 sense body message:

```
        (focus (target none) (count 18))
```

In this examples the:

- *count* parameter would be 18.

See also:

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildFocusState(int unum, int count)

Definition at line 4203 of file rccparser.h.

7.1.3.43 virtual void rcc::Parser::doBuildFoulBallOutLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_ballout_l` play mode.

Override this function in your subclass to handle the `foul_ballout_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `foul_ballout_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3079 of file rccparser.h.

7.1.3.44 virtual void rcc::Parser::doBuildFoulBallOutRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_ballout_r` play mode.

Override this function in your subclass to handle the `foul_ballout_r` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `foul_ballout_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3102 of file rccparser.h.

7.1.3.45 virtual void rcc::Parser::doBuildFoulChargeLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_charge_l` play mode.

Override this function in your subclass to handle the `foul_charge_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `foul_charge_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2941 of file rccparser.h.

7.1.3.46 virtual void rcc::Parser::doBuildFoulChargeRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_charge_r` play mode.

Override this function in your subclass to handle the `foul_charge_r` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `foul_charge_r` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2964 of file rccparser.h.

7.1.3.47 virtual void rcc::Parser::doBuildFoulLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_l` play mode.

Override this function in your subclass to handle the `foul_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `foul_l` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2895 of file rccparser.h.

7.1.3.48 virtual void rcc::Parser::doBuildFoulMultipleAttackerLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_multiple_attack_l` play mode.

Override this function in your subclass to handle the `foul_multiple_attack_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `foul_multiple_attack_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3033 of file rccparser.h.

7.1.3.49 virtual void rcc::Parser::doBuildFoulMultipleAttackerRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_multiple_attack_r` play mode.

Override this function in your subclass to handle the `foul_multiple_attack_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `foul_multiple_attack_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3056 of file rccparser.h.

7.1.3.50 virtual void rcc::Parser::doBuildFoulPushLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_push_l` play mode.

Override this function in your subclass to handle the `foul_push_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `foul_push_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2987 of file rccparser.h.

7.1.3.51 virtual void rcc::Parser::doBuildFoulPushRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_push_r` play mode.

Override this function in your subclass to handle the `foul_push_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `foul_push_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3010 of file rccparser.h.

7.1.3.52 virtual void rcc::Parser::doBuildFoulRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `foul_r` play mode.

Override this function in your subclass to handle the `foul_r` play mode.

Precondition:

Either:

- a full state message is being parsed,

- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `foul_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2918 of file rccparser.h.

7.1.3.53 virtual void rcc::Parser::doBuildFreeKickFaultLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `free_kick_fault_l` play mode.

Override this function in your subclass to handle the `free_kick_fault_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `free_kick_fault_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3171 of file rccparser.h.

7.1.3.54 virtual void rcc::Parser::doBuildFreeKickFaultRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `free_kick_fault_r` play mode.

Override this function in your subclass to handle the `free_kick_fault_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `free_kick_fault_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3194 of file rccparser.h.

7.1.3.55 virtual void rcc::Parser::doBuildFreeKickLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `free_kick_l` play mode.

Override this function in your subclass to handle the `free_kick_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `free_kick_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2527 of file rccparser.h.

7.1.3.56 virtual void rcc::Parser::doBuildFreeKickRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `free_kick_r` play mode.

Override this function in your subclass to handle the `free_kick_r` play mode.

Precondition:

Either:

- a full state message is being parsed,

- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `free_kick_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2550 of file rccparser.h.

7.1.3.57 virtual void rcc::Parser::doBuildFullState (int *time*) `[inline, protected, virtual]`

This function is called after parsing a full state message.

Override this function in your subclass to handle full state messages. You will also need to overwrite the functions for parsing the subsections of the full state message if you want to be able to use the data from those subsections.

This message is only sent to field players and only if sending of fullstate messages is specifically turned on in the simulator.

Parameters:

time The time (in server cycles) that the full state message was sent by the simulator.

Precondition:

The client is a field player.

Precondition:

Full state message sending is turned on in the simulator for the team this client is in.

Precondition:

An entire full state message has been parsed.

The following is an example (with line breaks added for clarity) of a version 8 or 9 full state message:

```
(fullstate 28 (pmode kick_off_l)
              (vmode high normal)
              (count 0 0 0 0 0 0 0 0)
              (arm (movable 0) (expires 0) (target 0 0) (count 0))
              (score 0 0)
              ((b) 0 0 0 0)
              ((p l 1 g) -3 -37 0 0 0 0 (stamina 4000 1 1))
              ((p l 2 0) -10 -38 0 0 0 0 (stamina 4000 1 1)))
```

The following is is example (with line breaks added for clarity) in version 7:

```
            (fullstate 28 (pmode kick_off_l)
                          (vmode high normal)
                          (score 0 0)
                          (ball 0 0 0 0)
                          (l_1 -3 -37 0 0 0 0 4000 1 1)
                          (l_2 -10 -38 0 0 0 0 4000 1 1))
```

In these examples the *time* parameter would be 28.

See also:

- doBuildCounts(int kick, int dash, int turn, int katch, int move, int turn_neck, int change_view, int say) -doBuildArmState(int movable_in, int expires_in, double target_x, double target_y, int count)
- doBuildScore(int our, int opp)
- doBuildBall(double x, double y, double delta_x, double delta_y)
- doBuildPlayer(int unum, double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, int stamina, double effort, double recovery)
- doBuildPlayer(int unum, int type, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double stamina, double effort, double recovery)
- doBuildPlayer(int unum, int type, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)
- doBuildGoalie(int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double stamina, double effort, double recovery)
- doBuildGoalie(int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)
- doBuildCatchBallLeftPlayMode()
- doBuildCatchBallRightPlayMode()
- doBuildBeforeKickOffPlayMode()
- doBuildTimeOverPlayMode()
- doBuildPlayOnPlayMode()
- doBuildKickOffLeftPlayMode()
- doBuildKickOffRightPlayMode()
- doBuildKickInLeftPlayMode()
- doBuildKickInRightPlayMode()
- doBuildFreeKickLeftPlayMode()
- doBuildFreeKickRightPlayMode()
- doBuildCornerKickLeftPlayMode()
- doBuildCornerKickRightPlayMode()
- doBuildGoalKickLeftPlayMode()
- doBuildGoalKickRightPlayMode()
- doBuildAfterGoalLeftPlayMode()
- doBuildAfterGoalRightPlayMode()
- doBuildDropBallPlayMode()
- doBuildOffSideLeftPlayMode()
- doBuildOffsideRightPlayMode()
- doBuildPenaltyKickLeftPlayMode()

- doBuildPenaltyKickRightPlayMode()
- doBuildFirstHalfOverPlayMode()
- doBuildPausePlayMode()
- doBuildHumanPlayMode()
- doBuildFoulLeftPlayMode()
- doBuildFoulRightPlayMode()
- doBuildFoulChargeLeftPlayMode()
- doBuildFoulChargeRightPlayMode()
- doBuildFoulPushLeftPlayMode()
- doBuildFoulPushRightPlayMode()
- doBuildFoulMultipleAttackerLeftPlayMode()
- doBuildFoulMultipleAttackerRightPlayMode()
- doBuildFoulBallOutLeftPlayMode()
- doBuildFoulBallOutRightPlayMode()
- doBuildBackPassLeftPlayMode()
- doBuildBackPassRightPlayMode()
- doBuildFreeKickFaultLeftPlayMode()
- doBuildFreeKickFaultRightPlayMode()

Definition at line 1875 of file rccparser.h.

7.1.3.58 virtual void rcc::Parser::doBuildGlobalBall (double *x*, double *y*, double *delta_x*, double *delta_y*) `[inline, protected, virtual]`

This function is called after parsing a ball section of a coach visual message.

Override this function in your subclass to handle ball sections of a coach visual message.

Parameters:

x The x co-ordinate of the ball.

y The y co-ordinate of the ball.

delta_x The magnitude of the ball's velociy along the x co-ordinate.

delta_y The magnitude of the ball's velociy along the y co-ordinate.

Precondition:

The client is a coach.

Precondition:

An entire ball section of coach visual message has been parsed.

Precondition:

A coach visual message is being parsed.

See also:

- doBuildGlobalVisual(int time)

Definition at line 7280 of file rccparser.h.

7.1.3.59 virtual void rcc::Parser::doBuildGlobalGoal (double *x*, double *y*) `[inline, protected, virtual]`

This function is called after parsing a goal section of a coach visual message.

Override this function in your subclass to handle goal sections of a coach visual message.

Parameters:

x The x co-ordinate of the goal.

y The y co-ordinate of the goal.

Precondition:

The client is a coach.

Precondition:

An entire goal section of coach visual message has been parsed.

Precondition:

A coach visual message is being parsed.

See also:

- doBuildGlobalVisual(int time)

Definition at line 7253 of file rccparser.h.

7.1.3.60 virtual void rcc::Parser::doBuildGlobalPlayer (double *x*, double *y*, double *delta_x*, double *delta_y*, double *orientation*, double *head_orientation*) `[inline, protected, virtual]`

This function is called after parsing a player section of a coach visual message.

Override this function in your subclass to handle player sections of a coach visual message.

Warning:

This function deprecated. Use doBuildGlobalPlayer(double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, double point_-dir, bool tackle) and doBuildGlobalPlayer(double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, bool tackle) instead.

Parameters:

x The x co-ordinate of the player.

y The y co-ordinate of the player.

delta_x The magnitude of the player's velociy along the x co-ordinate.

delta_y The magnitude of the player's velociy along the y co-ordinate.

orientation The direction the player's body is facing.

head_orientation The direction the player's head is facing.

Precondition:
The client is a coach.

Precondition:
An entire player section of coach visual message has been parsed.

Precondition:
A coach visual message is being parsed.

See also:
- doBuildGlobalVisual(int time)

Definition at line 7410 of file rccparser.h.

7.1.3.61 virtual void rcc::Parser::doBuildGlobalPlayer (double *x*, double *y*, double *delta_x*, double *delta_y*, double *orientation*, double *head_orientation*, bool *tackle*) `[inline, protected, virtual]`

This function is called after parsing a non-pointing player section of a coach visual message.

Override this function in your subclass to handle non-pointing player sections of a coach visual message.

Note:
This function deprecates doBuildGlobalPlayer(x, y, delta_x, delta_y, orientation, head_orientation).

Parameters:
x The x co-ordinate of the player.
y The y co-ordinate of the player.
delta_x The magnitude of the player's velociy along the x co-ordinate.
delta_y The magnitude of the player's velociy along the y co-ordinate.
orientation The direction the player's body is facing.
head_orientation The direction the player's head is facing.
tackle True if the player is tackling. False otherwise.

Precondition:
The player is not pointing.

Precondition:
The client is a coach.

Precondition:
An entire player section of coach visual message has been parsed.

Precondition:
A coach visual message is being parsed.

See also:

- doBuildGlobalVisual(int time)

Definition at line 7362 of file rccparser.h.

7.1.3.62 virtual void rcc::Parser::doBuildGlobalPlayer (double *x*, double *y*, double *delta_x*, double *delta_y*, double *orientation*, double *head_orientation*, double *point_dir*, bool *tackle*) `[inline, protected, virtual]`

This function is called after parsing a player section of a coach visual message, where the player is pointing.

Override this function in your subclass to handle pointing player sections of a coach visual message.

Note:

This function deprecates doBuildGlobalPlayer(x, y, delta_x, delta_y, orientation, head_orientation).

Parameters:

x The x co-ordinate of the player.

y The y co-ordinate of the player.

delta_x The magnitude of the player's velociy along the x co-ordinate.

delta_y The magnitude of the player's velociy along the y co-ordinate.

orientation The direction the player's body is facing.

head_orientation The direction the player's head is facing.

point_dir The direction the player is pointing in.

tackle True if the player is tackling. False otherwise.

Precondition:

The player is pointing.

Precondition:

The client is a coach.

Precondition:

An entire player section of coach visual message has been parsed.

Precondition:

A coach visual message is being parsed.

See also:

- doBuildGlobalVisual(int time)

Definition at line 7318 of file rccparser.h.

Referenced by doBuildGlobalPlayer().

7.1.3.63 virtual void rcc::Parser::doBuildGlobalVisual (int *time*) `[inline, protected, virtual]`

This function is called after parsing a coach visual message.

Override this function in your subclass to handle coach visual messages.

Parameters:

time The time (in server cycles) that the coach visual message was sent by the simulator.

Precondition:

The client is a coach player.

Precondition:

An entire coach visual message has been parsed.

Definition at line 7230 of file rccparser.h.

7.1.3.64 virtual void rcc::Parser::doBuildGoal (bool *close*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a goal section of player visual message, when the client is in a low quality view mode and the goal is close or far away.

Override this function in your subclass to handle low quality close or far goal sections of player visual messages.

Parameters:

close false if the goal is in the client's view cone, true otherwise.

dist The distance (in meters) from the client to the goal.

dir The angle from the client's center of view to the goal.

Precondition:

The client is a field player.

Precondition:

The client is in a low quality view mode.

Precondition:

An entire goal section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a far goal section of a player visual message:

```
((g l) -36)
```

In this example the:

- *close* parameter would be false.
- *dir* parameter would be -36.

The following is an example of a close goal section of a player visual message:

```
((G) 166)
```

In this example the:

- *close* parameter would be true.
- *dir* parameter would be -166.

See also:

- doBuildGoal(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildGoal(double dist, double dir, double dist_chg, double dir_chg)
- doBuildGoal(bool close, double dist, double dir)
- doBuildVisual(int time)
- doBuildLeftLocation()
- doBuildRightLocation()

Definition at line 6070 of file rccparser.h.

7.1.3.65 virtual void rcc::Parser::doBuildGoal (bool *close*, double *dist*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a goal section of player visual message, when the client is in a high quality view mode and the goal is close or far away.

Override this function in your subclass to handle high quality close or far goal sections of player visual messages.

Parameters:

close false if the goal is in the client's view cone, true otherwise.

dist The distance (in meters) from the client to the goal.

dir The angle from the client's center of view to the goal.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire goal section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a far goal section of a player visual message:

```
((g l) 82.3 -36)
```

In this example the:

- *close* parameter would be false.
- *dist* parameter would be 82.3.
- *dir* parameter would be -36.

The following is an example of a close goal section of a player visual message:

```
((G) 2.9 166)
```

In this example the:

- *close* parameter would be true.
- *dist* parameter would be 2.9.
- *dir* parameter would be -166.

See also:

- doBuildGoal(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildGoal(double dist, double dir, double dist_chg, double dir_chg)
- doBuildGoal(bool close, double dir)
- doBuildVisual(int time)
- doBuildLeftLocation()
- doBuildRightLocation()

Definition at line 6021 of file rccparser.h.

7.1.3.66 virtual void rcc::Parser::doBuildGoal (double *dist*, double *dir*, double *dist_chg*, double *dir_chg*) `[inline, protected, virtual]`

This function is called after parsing a goal section of player visual message, when the client is in a high quality view mode and the goal is not far away.

Override this function in your subclass to handle high quality near goal sections of player visual messages.

Note:

This function deprecates doBuildGoal(bool close, double dist, double dir, double dist_chg, double dir_chg).

Parameters:

dist The distance (in meters) from the client to the goal.

dir The angle from the client's center of view to the goal.

dist_chg The distance change of the goal relative to the player. Since the flag is stationary this can be used with *dir_chg* to augment the client's velocity information found in the sense body messages.

dir_chg This is the direction change of the goal relative to the client.

Precondition:
The client is a field player.

Precondition:
The client is in a high quality view mode.

Precondition:
An entire goal section of a player visual message has been parsed.

Precondition:
A player visual message is being parsed.

The following is an example of a near goal section of a player visual message:

```
((g r) 34.8 -19 0 -0.1)
```

In this example the:

- *dist* parameter would be 34.8.
- *dir* parameter would be -19.
- *dist_chg* parameter would be 0.
- *dir_chg* parameter would be -0.1.

See also:
- doBuildGoal(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildGoal(bool close, double dist, double dir)
- doBuildGoal(bool close, double dir)
- doBuildVisual(int time)
- doBuildLeftLocation()
- doBuildRightLocation()

Definition at line 5966 of file rccparser.h.

7.1.3.67 virtual void rcc::Parser::doBuildGoal (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*) `[inline, protected, virtual]`

This function is called after parsing a goal section of player visual message, when the client is in a high quality view mode, the goal is not far away and doBuildGoal(double dist, double dir, double dist_chg, double dir_chg) has not been overridden.

Override this function in your subclass to handle high quality near goal sections of player visual messages.

Warning:

This function is deprecated. Overide doBuildGoal(double dist, double dir, double dist_chg, double dir_chg) instead.

Parameters:

close false if the goal is in the client's view cone, true otherwise. Since dist_chg and dir_chg are only ever sent if the goal is in the client's view cone, close is always false, thus this parameter is redundant.

dist The distance (in meters) from the client to the goal.

dir The angle from the client's center of view to the goal.

dist_chg The distance change of the goal relative to the player. Since the flag is stationary this can be used with *dir_chg* to augment the client's velocity information found in the sense body messages.

dir_chg This is the direction change of the goal relative to the client.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire goal section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a near goal section of a player visual message:

```
((g r) 34.8 -19 0 -0.1)
```

In this example the:

- *close* parameter would be false.
- *dist* parameter would be 34.8.
- *dir* parameter would be -19.
- *dist_chg* parameter would be 0.
- *dir_chg* parameter would be -0.1.

See also:

- doBuildGoal(double dist, double dir, double dist_chg, double dir_chg)
- doBuildGoal(bool close, double dist, double dir)
- doBuildGoal(bool close, double dir)
- doBuildVisual(int time)
- doBuildLeftLocation()
- doBuildRightLocation()

Definition at line 5915 of file rccparser.h.

Referenced by doBuildGoal().

7.1.3.68 virtual void rcc::Parser::doBuildGoalKickLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `goal_kick_l` play mode.

Override this function in your subclass to handle the `goal_kick_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `goal_kick_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2619 of file rccparser.h.

7.1.3.69 virtual void rcc::Parser::doBuildGoalKickRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `goal_kick_r` play mode.

Override this function in your subclass to handle the `goal_kick_r` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `goal_kick_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2642 of file rccparser.h.

7.1.3.70 virtual void rcc::Parser::doBuildGoalLocation () `[inline, protected, virtual]`

This function is called after parsing a goal location of a field object or a visual.

Override this function in your subclass to handle goal field locations.

Precondition:
A goal loation of a field object has been parsed.

Precondition:
A field object is being parsed.

Precondition:
A a visual message is being parsed.

Definition at line 7566 of file rccparser.h.

7.1.3.71 virtual void rcc::Parser::doBuildGoalRefAudio (int *time*, int *score*) `[inline, protected, virtual]`

This function is called after parsing a goal referee message.

Override this function in your subclass to handle goal referee messages.

Precondition:
An entire goal referee message has been parsed.

Definition at line 7700 of file rccparser.h.

7.1.3.72 virtual void rcc::Parser::doBuildGoalie () `[inline, protected, virtual]`

This function is called after parsing a goalie indicator of a player name.

Override this function in your subclass to handle goalie indicators.

Precondition:
A goalie inidicator has been parsed.

Definition at line 7602 of file rccparser.h.

7.1.3.73 virtual void rcc::Parser::doBuildGoalie (int *unum*, double *pos_x*, double *pos_y*, double *vel_x*, double *vel_y*, double *body_dir*, double *head_dir*, double *arm_mag*, double *arm_dir*, double *stamina*, double *effort*, double *recovery*) `[inline, protected, virtual]`

This function is called after parsing a player section of a version 8 or 9 full state message, where the player **is** pointing and the player **is** a goalie.

Override this function in your subclass to handle pointing goalie player sections of version 8 or 9 full state messages.

Parameters:

unum The uniform number of the player.

x The x co-ordinate of the player.

y The y co-ordinate of the player.

delta_x The x velocity of the player.

delta_y The y velocity of the player.

orientation The direction the player's body is facing.

head_orientation The direction the player's head is facing.

arm_mag The distance from the center of the field to the point the player is pointing to.

arm_dir The direction the player is pointing in.

stamina The player's stamina.

effort The player's effort.

recovery The player's recovery.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

Full state message sending is turned on in the simulator for the team this client is in.

Precondition:

A pointing goalie player section of a version 8 or 9 full state message has been parsed.

The following is an example of a pointing goalie player section of a version 8 or 9 full state message:

```
((p l 5 g) -20.5 10.7 0.56 -0.3 13.5 4.8 13.6 -10.2 (stamina 2964.5 0.93 0.54))
```

In this examples the:

- *unum* parameter would be 5.
- *x* parameter would be -20.5.
- *y* parameter would be 10.7.
- *delta_x* parameter would be 0.56.
- *delta_y* parameter would be -0.3.
- *orientation* parameter would be 13.5.

- *head_orientation* parameter would be 4.8.
- *arm_mag* parameter would be 13.6.
- *arm_dir* parameter would be -10.2.
- *stamina* parameter would be 2964.5.
- *effort* parameter would be 0.93.
- *recovery* parameter would be 0.54.

See also:

- doBuildFullState(int time)
- doBuildLeftSide()
- doBuildRightSide()

Definition at line 2292 of file rccparser.h.

7.1.3.74 virtual void rcc::Parser::doBuildGoalie (int *unum*, double *pos_x*, double *pos_y*, double *vel_x*, double *vel_y*, double *body_dir*, double *head_dir*, double *stamina*, double *effort*, double *recovery*) `[inline, protected, virtual]`

This function is called after parsing a player section of a version 8 or 9 full state message, where the player is **not** pointing and the player **is** a goalie.

Override this function in your subclass to handle non-pointing goalie player sections of version 8 or 9 full state messages.

Parameters:

unum The uniform number of the player.

x The x co-ordinate of the player.

y The y co-ordinate of the player.

delta_x The x velocity of the player.

delta_y The y velocity of the player.

orientation The direction the player's body is facing.

head_orientation The direction the player's head is facing.

stamina The player's stamina.

effort The player's effort.

recovery The player's recovery.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

Full state message sending is turned on in the simulator for the team this client is in.

Precondition:
A non-pointing, goalie player section of a version 8 or 9 full state message has been parsed.

The following is an example of a non-pointing goalie player section of a version 8 or 9 full state message:

```
((p l 5 g) -20.5 10.7 0.56 -0.3 13.5 4.8 (stamina 2964.5 0.93 0.54))
```

In this examples the:

- *unum* parameter would be 5.
- *x* parameter would be -20.5.
- *y* parameter would be 10.7.
- *delta_x* parameter would be 0.56.
- *delta_y* parameter would be -0.3.
- *orientation* parameter would be 13.5.
- *head_orientation* parameter would be 4.8.
- *stamina* parameter would be 2964.5.
- *effort* parameter would be 0.93.
- *recovery* parameter would be 0.54.

See also:
- doBuildFullState(int time)
- doBuildLeftSide()
- doBuildRightSide()

Definition at line 2229 of file rccparser.h.

7.1.3.75 virtual void rcc::Parser::doBuildHumanPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `human_judge` play mode.

Override this function in your subclass to handle the `human_judge` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `human_judge` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2872 of file rccparser.h.

7.1.3.76 virtual void rcc::Parser::doBuildIllegalCommandError () `[inline, protected, virtual]`

This function is called after parsing an illegal command form error message.

Override this function in your subclass to handle illegal command form error messages.

Precondition:
The client has sent a command whoes format does not match that expected by the server

Precondition:
An entire illegal command form error message has been parsed.

Definition at line 8074 of file rccparser.h.

7.1.3.77 virtual void rcc::Parser::doBuildIllegalModeError () `[inline, protected, virtual]`

This function is called after parsing an illegal mode error message.

Override this function in your subclass to handle illegal mode error messages.

Precondition:
The client is an offline coach

Precondition:
The client has sent a change_mode message with a playmode of PM_Null

Precondition:
An entire illegal mode error message has been parsed.

Definition at line 8108 of file rccparser.h.

7.1.3.78 virtual void rcc::Parser::doBuildIllegalObjectFormError () `[inline, protected, virtual]`

This function is called after parsing an illegal object form error message.

Override this function in your subclass to handle illegal object form error messages.

Precondition:
The client is an offline coach

Precondition:
The client has sent a move message, where the the name of the object to be moved is not in the format expected by the server.

Precondition:
An entire illegal object form error message has been parsed.

Definition at line 8128 of file rccparser.h.

7.1.3.79 virtual void rcc::Parser::doBuildIndirectFreeKickLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `indirect_free_kick_l` play mode.

Override this function in your subclass to handle the `indirect_free_kick_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `indirect_free_kick_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3260 of file rccparser.h.

7.1.3.80 virtual void rcc::Parser::doBuildIndirectFreeKickRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `indirect_free_kick_r` play mode.

Override this function in your subclass to handle the `indirect_free_kick_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `indirect_free_kick_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3282 of file rccparser.h.

7.1.3.81 virtual void rcc::Parser::doBuildInit () `[inline, protected, virtual]`

This function is called after parsing a player reconnect message.

Override this function in your subclass to handle player reconnect messages

Precondition:
The client is a player

Precondition:
An entire player reconnect has been parsed.

Definition at line 7643 of file rccparser.h.

7.1.3.82 virtual void rcc::Parser::doBuildInit (int *unum*) `[inline, protected, virtual]`

This function is called after parsing a player init message.

Override this function in your subclass to handle player init messages

Precondition:
The client is a field player

Precondition:
An entire player init message has been parsed.

Definition at line 7630 of file rccparser.h.

7.1.3.83 virtual void rcc::Parser::doBuildInitError () `[inline, protected, virtual]`

This function is called after parsing a reconnection error message.

Override this function in your subclass to handle reconnection error messages.

Precondition:
The client is a field player

Precondition:
The client has sent a reconnect message.

Precondition:
Either:

- There was an error in the reconnect message,
- There was an error connecting to the clients socket, or
- The team-name and uniform number do not match any field player

Precondition:

An entire reconnection error message has been parsed.

Definition at line 7958 of file rccparser.h.

7.1.3.84 virtual void rcc::Parser::doBuildKickInLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `kick_in_l` play mode.

Override this function in your subclass to handle the `kick_in_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `kick_in_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2481 of file rccparser.h.

7.1.3.85 virtual void rcc::Parser::doBuildKickInRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `kick_in_r` play mode.

Override this function in your subclass to handle the `kick_in_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `kick_in_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2504 of file rccparser.h.

7.1.3.86 virtual void rcc::Parser::doBuildKickOffLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `kick_off_l` play mode.

Override this function in your subclass to handle the `kick_off_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `kick_off_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2435 of file rccparser.h.

7.1.3.87 virtual void rcc::Parser::doBuildKickOffRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `kick_off_r` play mode.

Override this function in your subclass to handle the `kick_off_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `kick_off_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2458 of file rccparser.h.

7.1.3.88 virtual void rcc::Parser::doBuildLeftLocation () `[inline, protected, virtual]`

This function is called after parsing a left location of a field object or a visual.

Override this function in your subclass to handle left field locations.

Precondition:
A left loation of a field object has been parsed.

Precondition:
A field object is being parsed.

Precondition:
A a visual message is being parsed.

Definition at line 7498 of file rccparser.h.

7.1.3.89 virtual void rcc::Parser::doBuildLeftSide () `[inline, protected, virtual]`

This function is called after parsing a token representing the left team side.

Override this function in your subclass to handle left team sides.

Precondition:
Either:

- a player section of a full state message is being parsed,
- an init message or reconnect message is being parsed,
- a focus section of a sense body message is being parsed, where the client is focused on a player,
- a referee goal audio message is being parsed or
- a non CLang coach audio message is being parsed.

Precondition:
A token representing the left team has been parsed.

See also:

- doBuildPlayer(int unum, double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, int stamina, double effort, double recovery)
- doBuildPlayer(int unum, int type, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)
- doBuildGoalie(int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double stamina, double effort, double recovery)
- doBuildGoalie(int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)

- doBuildInit(int unum)
- doBuildInit()
- doBuildCoachInit()
- doBuildFocusState(int unum, int count)
- doBuildGoalRefAudio(int time, int score)
- doBuildCoachAudio(int time, const std::string& msg)

Definition at line 3670 of file rccparser.h.

7.1.3.90 virtual void rcc::Parser::doBuildLine (double *dir*) `[inline, protected, virtual]`

This function is called after parsing a line section of player visual message, when the client is in a low quality view mode.

Override this function in your subclass to handle low quality line sections of player visual messages.

Parameters:

dist The distance (in meters) from the client to the point the center of the client's field of view intersects the line.

Precondition:

The client is a field player.

Precondition:

The client is in a low quality view mode.

Precondition:

An entire line section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a line section of a player visual message:

```
((l r) 90)
```

In this example the:

- *dir* paramter would be 90.

See also:

- doBuildLine(double dist, double dir)
- doBuildVisual(int time)
- doBuildTopLocation()
- doBuildBottomLocation()
- doBuildLeftLocation()
- doBuildRightLocation()

Definition at line 5633 of file rccparser.h.

7.1.3.91 virtual void rcc::Parser::doBuildLine (double *dist*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a line section of player visual message, when the client is in a high quality view mode.

Override this function in your subclass to handle high quality line sections of player visual messages.

Parameters:

dist The distance (in meters) from the client to the point the center of the client's field of view intersects the line.

dir The angle of the line to the center of the client's field of view

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire line section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a line section of a player visual message:

```
((l r) 72.2 90)
```

In this example the:

- *dist* parameter would be 72.2.
- *dir* paramter would be 90.

See also:

- doBuildLine(double dir)
- doBuildVisual(int time)
- doBuildTopLocation()
- doBuildBottomLocation()
- doBuildLeftLocation()
- doBuildRightLocation()

Definition at line 5596 of file rccparser.h.

7.1.3.92 virtual void rcc::Parser::doBuildMaxOfThatPlayerTypeOnFieldWarning () `[inline, protected, virtual]`

This function is called after parsing a `max_of_that_type_on_field` warning message.

Override this function in your subclass to handle `max_of_that_type_on_field` warning messages.

Precondition:
The client is an online coach

Precondition:
The client sent a `change_player_type` command but there are already the prescribed maximum number of of that type already on the field.

Precondition:
An entire `max_of_that_type_on_field` warning message has been parsed.

Definition at line 8401 of file rccparser.h.

7.1.3.93 virtual void rcc::Parser::doBuildNeutralSide () `[inline, protected, virtual]`

This function is called after parsing a token representing the neither side.

Override this function in your subclass to handle neutral sides.

Precondition:
A offline coach audio message is being parsed.

Precondition:
A token representing the offline coach been parsed.

See also:
- doBuildCoachAudio(int time, const std::string& msg)

Definition at line 3777 of file rccparser.h.

7.1.3.94 virtual void rcc::Parser::doBuildNoMoreTeamOrPlayerOrGoalieError () `[inline, protected, virtual]`

This function is called after parsing a player initialisation error message.

Override this function in your subclass to handle player initialisation error messages.

Precondition:
The client is a field player

Precondition:
The client has sent an init message.

Precondition:
Either:

- There was an error in the init message,
- There were illegal characters in the team-name,
- The team-name was too long,
- There are two teams already connected and the team-name matched neither,
- There are already 11 players connected for the team specified,
- The client tried to connect as a goalie and the team specified already has a goalie connected,
- The client tried to connect with a non-existant protocol version, or
- There was an error connecting to the clients socket.

Precondition:
An entire player initialisation error message has been parsed.

Definition at line 7990 of file rccparser.h.

7.1.3.95 virtual void rcc::Parser::doBuildNoSubsLeftWarning () `[inline, protected, virtual]`

This function is called after parsing a `no_subs_left` warning message.

Override this function in your subclass to handle `no_subs_left` warning messages.

Precondition:
The client is an online coach

Precondition:
The client sent a `change_player_type` command but the prescribed numbers of subs have already been used.

Precondition:
An entire `no_subs_left` warning message has been parsed.

Definition at line 8381 of file rccparser.h.

7.1.3.96 virtual void rcc::Parser::doBuildNoSuchPlayerWarning () `[inline, protected, virtual]`

This function is called after parsing a `no_such_player` warning message.

Override this function in your subclass to handle `no_team_found` warning messages.

Precondition:
The client is an online or offline coach

Precondition:
The client sent a `change_player_type` command and the player specified does not exist

Precondition:
An entire `no_such_player` warning message has been parsed.

Definition at line 8342 of file rccparser.h.

7.1.3.97 virtual void rcc::Parser::doBuildNoSuchTeamOrAlreadyHaveCoachError () `[inline, protected, virtual]`

This function is called after parsing an online coach initialisation error message.

Override this function in your subclass to handle online coach initialisation error messages.

Precondition:
The client is an online coach.

Precondition:
The client has sent an init message.

Precondition:
Either:

- There was an error in the init message,
- The version number specified was less than 5,
- The team-name specified was too long,
- The team-name specified does not match any team that is connected,
- The specified team already has a coach connected,
- The client tried to connect with a non-existant protocol version, or
- There was an error connecting to the clients socket.

Precondition:
An entire coach initialisation error message has been parsed.

Definition at line 8019 of file rccparser.h.

7.1.3.98 virtual void rcc::Parser::doBuildNoTeamFoundWarning () `[inline, protected, virtual]`

This function is called after parsing a `no_team_found` warning message.

Override this function in your subclass to handle `no_team_found` warning messages.

Precondition:
The client is an offline coach

Precondition:
The client sent a `change_player_type` command and the team specified does not exist

Precondition:
An entire `no_team_found` warning message has been parsed.

Definition at line 8323 of file rccparser.h.

7.1.3.99 virtual void rcc::Parser::doBuildOffSideLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `offside_l` play mode.

Override this function in your subclass to handle the `offside_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `offside_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2734 of file rccparser.h.

7.1.3.100 virtual void rcc::Parser::doBuildOffsideRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `offside_r` play mode.

Override this function in your subclass to handle the `offside_r` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `offside_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2757 of file rccparser.h.

7.1.3.101 virtual void rcc::Parser::doBuildOppSide () `[inline, protected, virtual]`

This function is called after parsing a token representing the client's team's side.

Override this function in your subclass to handle client team sides.

Precondition:
A version 8 or 9 player audio message is being parsed.

Precondition:
A token representing the client's team has been parsed.

See also:

- doBuildPlayerAudio(int time, double dir, const std::string& msg)
- doBuildPlayerAudio(int time, double dir, int unum, const std::string& msg)
- doBuildPlayerAudio(int time, int unum)
- doBuildPlayerAudio(int time)

Definition at line 3741 of file rccparser.h.

7.1.3.102 virtual void rcc::Parser::doBuildOurSide () `[inline, protected, virtual]`

This function is called after parsing a token representing opponent's team's side.

Override this function in your subclass to handle opponent team sides.

Precondition:
A version 8 or 9 player audio message is being parsed.

Precondition:
A token representing the opponent team has been parsed.

See also:

- doBuildPlayerAudio(int time, double dir, const std::string& msg)
- doBuildPlayerAudio(int time, double dir, int unum, const std::string& msg)
- doBuildPlayerAudio(int time, int unum)
- doBuildPlayerAudio(int time)

Definition at line 3761 of file rccparser.h.

7.1.3.103 virtual void rcc::Parser::doBuildParam (const std::string & *name*, const std::string & *value*) `[inline, protected, virtual]`

This function is called after parsing a param section of a version 8 or 9 server parameter, player parameter or player type message, where the type of the parameter is an string.

Override this function in your subclass to handle string parameters of version 8 or 9 a server parameter, player parameter or player type message.

Parameters:

name The name of the parameter.

value The vaule of the parameter.

Precondition:

The client is using protocol version 8 or 9

Precondition:

Either:

- A version 8 or 9 server parameter message is being parsed.
- A version 8 or 9 player parameter message is being parsed.
- A version 8 or 9 player type message is being parsed.

Precondition:

A string parameter section of a version 8 or 9 server parameter, player parameter or player type message has just been parsed.

The following is an example of a string parameter section of a version 8 or 9 server param message:

```
(landmark_file "~/.rcssserver-landmark.xml")
```

See also:

- doBuildServerParam()
- doBuildPlayerParam()
- doBuildPlayerType()
- doBuildParam(const std::string& name, int value)
- doBuildParam(const std::string& name, const std::string& value)

Definition at line 5210 of file rccparser.h.

7.1.3.104 virtual void rcc::Parser::doBuildParam (const std::string & *name*, double *value*) `[inline, protected, virtual]`

This function is called after parsing a param section of a version 8 or 9 server parameter, player parameter or player type message, where the type of the parameter is an real.

Override this function in your subclass to handle real parameters of version 8 or 9 a server parameter, player parameter or player type message.

Parameters:

name The name of the parameter.

value The vaule of the parameter.

Precondition:

The client is using protocol version 8 or 9

Precondition:

Either:

- A version 8 or 9 server parameter message is being parsed.
- A version 8 or 9 player parameter message is being parsed.
- A version 8 or 9 player type message is being parsed.

Precondition:

A real parameter section of a version 8 or 9 server parameter, player parameter or player type message has just been parsed.

The following is an example of a real parameter section of a version 8 or 9 server param message:

```
(stopped_ball_vel 0.01)
```

See also:

- doBuildServerParam()
- doBuildPlayerParam()
- doBuildPlayerType()
- doBuildParam(const std::string& name, int value)
- doBuildParam(const std::string& name, const std::string& value)

Definition at line 5168 of file rccparser.h.

7.1.3.105 virtual void rcc::Parser::doBuildParam (const std::string & *name*, int *value*) `[inline, protected, virtual]`

This function is called after parsing a param section of a version 8 or 9 server parameter, player parameter or player type message, where the type of the parameter is an integer.

Override this function in your subclass to handle integer parameters of version 8 or 9 a server parameter, player parameter or player type message.

Parameters:

name The name of the parameter.

value The vaule of the parameter.

Precondition:

The client is using protocol version 8 or 9

Precondition:

Either:

- A version 8 or 9 server parameter message is being parsed.
- A version 8 or 9 player parameter message is being parsed.
- A version 8 or 9 player type message is being parsed.

Precondition:

A integer parameter section of a version 8 or 9 server parameter, player parameter or player type message has just been parsed.

The following is an example of a integer parameter section of a version 8 or 9 server param message:

```
        (catch_ban_cycle 5)
```

See also:

- doBuildServerParam()
- doBuildPlayerParam()
- doBuildPlayerType()
- doBuildParam(const std::string& name, double value)
- doBuildParam(const std::string& name, const std::string& value)

Definition at line 5126 of file rccparser.h.

7.1.3.106 virtual void rcc::Parser::doBuildPausePlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `pause` play mode.

Override this function in your subclass to handle the `pause` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `pause` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2849 of file rccparser.h.

7.1.3.107 virtual void rcc::Parser::doBuildPenaltyDrawPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_draw` play mode.

Override this function in your subclass to handle the `penalty_draw` play mode.

Precondition:
A referee audio message is beign parsed.

Precondition:
A `penalty_draw` play mode has just been parsed.

See also:
- doBuildRefAudio(int time)

Definition at line 3614 of file rccparser.h.

7.1.3.108 virtual void rcc::Parser::doBuildPenaltyFoulLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_foul_l` play mode.

Override this function in your subclass to handle the `penalty_foul_l` play mode.

Precondition:
A referee audio message is beign parsed.

Precondition:
A `penalty_foul_l` play mode has just been parsed.

See also:
- doBuildRefAudio(int time)

Definition at line 3550 of file rccparser.h.

7.1.3.109 virtual void rcc::Parser::doBuildPenaltyFoulRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_foul_r` play mode.

Override this function in your subclass to handle the `penalty_foul_r` play mode.

Precondition:
A referee audio message is beign parsed.

Precondition:
A `penalty_foul_r` play mode has just been parsed.

See also:

- doBuildRefAudio(int time)

Definition at line 3566 of file rccparser.h.

7.1.3.110 virtual void rcc::Parser::doBuildPenaltyKickLeftPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_kick_l` play mode.

Override this function in your subclass to handle the `penalty_kick_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `penalty_kick_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2780 of file rccparser.h.

7.1.3.111 virtual void rcc::Parser::doBuildPenaltyKickRightPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_kick_r` play mode.

Override this function in your subclass to handle the `penalty_kick_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `penalty_kick_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2803 of file rccparser.h.

7.1.3.112 virtual void rcc::Parser::doBuildPenaltyLocation () `[inline, protected, virtual]`

This function is called after parsing a penalty location of a field object or a visual.

Override this function in your subclass to handle penalty field locations.

Precondition:
A penalty loation of a field object has been parsed.

Precondition:
A field object is being parsed.

Precondition:
A a visual message is being parsed.

Definition at line 7549 of file rccparser.h.

7.1.3.113 virtual void rcc::Parser::doBuildPenaltyMissLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_miss_l` play mode.

Override this function in your subclass to handle the `penalty_miss_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `penalty_miss_l` play mode has just been parsed.

See also:
- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3436 of file rccparser.h.

7.1.3.114 virtual void rcc::Parser::doBuildPenaltyMissRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_miss_r` play mode.

Override this function in your subclass to handle the `penalty_miss_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `penalty_miss_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3458 of file rccparser.h.

7.1.3.115 virtual void rcc::Parser::doBuildPenaltyOnFieldLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_onfield_l` play mode.

Override this function in your subclass to handle the `penalty_onfield_l` play mode.

Precondition:

A referee audio message is beign parsed.

Precondition:

A `penalty_onfield_l` play mode has just been parsed.

See also:

- doBuildRefAudio(int time)

Definition at line 3518 of file rccparser.h.

7.1.3.116 virtual void rcc::Parser::doBuildPenaltyOnFieldRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_onfield_r` play mode.

Override this function in your subclass to handle the `penalty_onfield_r` play mode.

Precondition:

A referee audio message is beign parsed.

Precondition:

A `penalty_onfield_r` play mode has just been parsed.

See also:

- doBuildRefAudio(int time)

Definition at line 3534 of file rccparser.h.

7.1.3.117 virtual void rcc::Parser::doBuildPenaltyReadyLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_ready_l` play mode.

Override this function in your subclass to handle the `penalty_ready_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `penalty_ready_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3348 of file rccparser.h.

7.1.3.118 virtual void rcc::Parser::doBuildPenaltyReadyRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_ready_r` play mode.

Override this function in your subclass to handle the `penalty_ready_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `penalty_ready_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3370 of file rccparser.h.

7.1.3.119 virtual void rcc::Parser::doBuildPenaltyScoreLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_score_l` play mode.

Override this function in your subclass to handle the `penalty_score_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `penalty_score_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3480 of file rccparser.h.

7.1.3.120 virtual void rcc::Parser::doBuildPenaltyScoreRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_score_r` play mode.

Override this function in your subclass to handle the `penalty_score_r` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `penalty_score_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3502 of file rccparser.h.

7.1.3.121 virtual void rcc::Parser::doBuildPenaltySetupLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_setup_l` play mode.

Override this function in your subclass to handle the `penalty_setup_l` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `penalty_setup_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3304 of file rccparser.h.

7.1.3.122 virtual void rcc::Parser::doBuildPenaltySetupRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_setup_r` play mode.

Override this function in your subclass to handle the `penalty_setup_r` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `penalty_setup_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3326 of file rccparser.h.

7.1.3.123 virtual void rcc::Parser::doBuildPenaltyTakenLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_taken_l` play mode.

Override this function in your subclass to handle the `penalty_taken_l` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `penalty_taken_l` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3392 of file rccparser.h.

7.1.3.124 virtual void rcc::Parser::doBuildPenaltyTakenRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_taken_r` play mode.

Override this function in your subclass to handle the `penalty_taken_r` play mode.

Precondition:

Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `penalty_taken_r` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 3414 of file rccparser.h.

7.1.3.125 virtual void rcc::Parser::doBuildPenaltyWinnerLeftPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_winner_l` play mode.

Override this function in your subclass to handle the `penalty_winner_l` play mode.

Precondition:
A referee audio message is beign parsed.

Precondition:
A `penalty_winner_l` play mode has just been parsed.

See also:
- doBuildRefAudio(int time)

Definition at line 3582 of file rccparser.h.

7.1.3.126 virtual void rcc::Parser::doBuildPenaltyWinnerRightPlaymode () `[inline, protected, virtual]`

This function is called after parsing of a `penalty_winner_r` play mode.

Override this function in your subclass to handle the `penalty_winner_r` play mode.

Precondition:
A referee audio message is beign parsed.

Precondition:
A `penalty_winner_r` play mode has just been parsed.

See also:
- doBuildRefAudio(int time)

Definition at line 3598 of file rccparser.h.

7.1.3.127 virtual void rcc::Parser::doBuildPlayOnPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `play_on` play mode.

Override this function in your subclass to handle the `play_on` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:

A `play_on` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2412 of file rccparser.h.

7.1.3.128 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a player section of player visual message, when the client is in a low quality view mode.

Override this function in your subclass to handle low quality player sections of player visual messages.

Parameters:

close false if the ball is in the client's view cone, true otherwise.

dir The angle from the client's center of view to the player.

Precondition:

The client is a field player.

Precondition:

The client is in a low quality view mode.

Precondition:

An entire player section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildVisual(int time)

Definition at line 7208 of file rccparser.h.

7.1.3.129 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*) `[inline, protected, virtual]`

This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is far away and not pointing or the player is very far away and the client is using protocol 7, or the player is very close, but outside of the view cone, and doBuildPlayer(bool close, double dist, double dir, bool tackling) has not been overridden.

Override this function in your subclass to handle high quality far non-pointing or version 7, or very close player sections of player visual messages.

Warning:

This function is deprecated. Override doBuildPlayer(bool close, double dist, double dir, bool tackling) instead.

Parameters:

close false if the ball is in the client's view cone, true otherwise.

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire far non-pointing or version 7, or very close player section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 7139 of file rccparser.h.

7.1.3.130 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*) `[inline, protected, virtual]`

This function never called.

Warning:

This function is never called because the server never sends dist_chg and dir_chg without body and head directions.

Definition at line 7060 of file rccparser.h.

7.1.3.131 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, double *orientation*, double *head_orientation*) `[inline, protected, virtual]`

This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is not far away, is not pointing or the client is using protcol version 7 and doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling), and doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling) have not been overridden.

Override this function in your subclass to handle version 8 or 9 high quality near non-pointing or version 7 high quality near player sections of player visual messages.

Warning:
This function is deprecated. Overide doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling) instead.

Parameters:

close false if the ball is in the client's view cone, true otherwise. Since dist_chg, dir_chg, etc are only ever sent if the player is in the client's view cone, *close* is always false, thus this parameter is redundant.

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

dist_chg The distance change of the player relative to the client.

dir_chg The the direction change of the player relative to the client.

orientation The the direction of the player's body relative to the client.

head_orientation The the direction of the player's head relative to the client.

Precondition:
Either:

- The client is using protocol 8 or 9 and the player is not pointing, or
- The client is using protocol 7.

Precondition:
The client is a field player.

Precondition:
The client is in a high quality view mode.

Precondition:
Either an entire:

- near, non-pointing or
- version 7 player section of a player visual message has been parsed.

Precondition:
A player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)

- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 7047 of file rccparser.h.

7.1.3.132 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, bool *tackling*) `[inline, protected, virtual]`

This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is far away and not pointing or the player is very far away and the client is using protocol 7, or the player is very close, but outside of the view cone.

Override this function in your subclass to handle high quality far non-pointing or version 7, or very close player sections of player visual messages.

Note:

This function is deprecates doBuildPlayer(bool close, double dist, double dir).

Parameters:

close false if the ball is in the client's view cone, true otherwise.

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

tackling true if the player is tackling and is in the clients view cone, false otherwise.

Precondition:

The client is a field player.

Precondition:
The client is in a high quality view mode.

Precondition:
An entire far non-pointing or version 7, or very close player section of a player visual message has been parsed.

Precondition:
A player visual message is being parsed.

See also:
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6947 of file rccparser.h.

7.1.3.133 virtual void rcc::Parser::doBuildPlayer (double *dist*, double *dir*, double *point_dir*, bool *tackling*) `[inline, protected, virtual]`

This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is far away and is pointing.

Override this function in your subclass to handle version 8 or 9 high quality far pointing player sections of player visual messages.

Note:

This function is deprecates doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling).

Parameters:

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

point_dir The the direction of the player is pointing relative to the client.

tackling true if the player is tackling, false otherwise.

Precondition:

The client is using protocol 8 or 9.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire far, pointing, player section of a player visual message has been parsed.

Precondition:

A version 8 or 9 player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)

- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6867 of file rccparser.h.

7.1.3.134 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, double *point_dir*, bool *tackling*) `[inline, protected, virtual]`

This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is far away, is pointing and doBuildPlayer(double dist, double dir, double point_dir, bool tackling) has not been overridden.

Override this function in your subclass to handle version 8 or 9 high quality far pointing player sections of player visual messages.

Warning:

This function is deprecated. Overide doBuildPlayer(double dist, double dir, double point_dir, bool tackling) instead.

Parameters:

close false if the ball is in the client's view cone, true otherwise. Since point_dir and tackling are only ever sent if the player is in the client's view cone, *close* is always false, thus this parameter is redundant.

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

point_dir The the direction of the player is pointing relative to the client.

tackling true if the player is tackling, false otherwise.

Precondition:

The client is using protocol 8 or 9.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire far, pointing, player section of a player visual message has been parsed.

Precondition:

A version 8 or 9 player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)

- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6787 of file rccparser.h.

7.1.3.135 virtual void rcc::Parser::doBuildPlayer (double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, bool *tackling*) `[inline, protected, virtual]`

This function never called.

Warning:

This function is never called because the server never sends dist_chg and dir_chg without body and head directions.

Definition at line 6700 of file rccparser.h.

7.1.3.136 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, bool *tackling*) `[inline, protected, virtual]`

This function never called.

Warning:

This function is never called because the server never sends dist_chg and dir_chg without body and head directions.

Definition at line 6685 of file rccparser.h.

7.1.3.137 virtual void rcc::Parser::doBuildPlayer (double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, double *point_dir*, bool *tackling*) `[inline, protected, virtual]`

This function never called.

Warning:
This function is never called because the server never sends dist_chg and dir_chg without body and head directions.

Definition at line 6671 of file rccparser.h.

7.1.3.138 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, double *point_dir*, bool *tackling*) `[inline, protected, virtual]`

This function never called.

Warning:
This function is never called because the server never sends dist_chg and dir_chg without body and head directions.

Definition at line 6656 of file rccparser.h.

7.1.3.139 virtual void rcc::Parser::doBuildPlayer (double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, double *orientation*, double *head_orientation*, bool *tackling*) `[inline, protected, virtual]`

This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is not far away and the player is not pointing or the client is using protocol 7.

Override this function in your subclass to handle version 8 or 9 high quality near non-pointing or version 7 high quality near player sections of player visual messages.

Note:
This function is deprecates doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling).

Parameters:

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

dist_chg The distance change of the player relative to the client.

dir_chg The the direction change of the player relative to the client.

orientation The the direction of the player's body relative to the client.

head_orientation The the direction of the player's head relative to the client.

tackling false if the player is not tackling or the client is using protocol 7, true otherwise.

Precondition:

Either:

- The client is using protocol 8 or 9 and the player is not pointing, or
- The client is using protocol 7.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire near, non-pointing or version 7, player section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6640 of file rccparser.h.

7.1.3.140 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, double *orientation*, double *head_orientation*, bool *tackling*) `[inline, protected, virtual]`

This function is called after parsing a player section of player visual message, when the client is in a high quality view mode, the player is not far away, is not pointing or the client is using protcol version 7 and doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling) has not been overridden.

Override this function in your subclass to handle version 8 or 9 high quality near non-pointing or version 7 high quality near player sections of player visual messages.

Warning:

This function is deprecated. Overide doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling) instead.

Note:

This function deprecates doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation).

Parameters:

close false if the ball is in the client's view cone, true otherwise. Since dist_chg, dir_chg, etc are only ever sent if the player is in the client's view cone, *close* is always false, thus this parameter is redundant.

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

dist_chg The distance change of the player relative to the client.

dir_chg The the direction change of the player relative to the client.

orientation The the direction of the player's body relative to the client.

head_orientation The the direction of the player's head relative to the client.

tackling false if the player is not tackling or protocol 7 is being used, true otherwise.

Precondition:

Either:

- The client is using protocol 8 or 9 and the player is not pointing, or
- The client is using protocol 7.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

Either an entire:

- near, non-pointing or
- version 7 player section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6549 of file rccparser.h.

7.1.3.141 virtual void rcc::Parser::doBuildPlayer (double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, double *orientation*, double *head_orientation*, double *point_dir*, bool *tackling*) `[inline, protected, virtual]`

This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is not far away and is pointing.

Override this function in your subclass to handle version 8 or 9 high quality near pointing player sections of player visual messages.

Note:

This function is deprecates doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling).

Parameters:

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

dist_chg The distance change of the player relative to the client.

dir_chg The the direction change of the player relative to the client.

orientation The the direction of the player's body relative to the client.

head_orientation The the direction of the player's head relative to the client.

point_dir The the direction of the player is pointing relative to the client.

tackling true if the player is tackling, false otherwise.

Precondition:

The client is using protocol 8 or 9.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire near, pointing, player section of a player visual message has been parsed.

Precondition:

A version 8 or 9 player visual message is being parsed.

See also:

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)

- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6444 of file rccparser.h.

7.1.3.142 virtual void rcc::Parser::doBuildPlayer (bool *close*, double *dist*, double *dir*, double *dist_chg*, double *dir_chg*, double *orientation*, double *head_orientation*, double *point_dir*, bool *tackling*) `[inline, protected, virtual]`

This function is called after parsing a version 8 or 9 player section of player visual message, when the client is in a high quality view mode, the player is not far away, is pointing and doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling) has not been overridden.

Override this function in your subclass to handle version 8 or 9 high quality near pointing player sections of player visual messages.

Warning:

This function is deprecated. Overide doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling) instead.

Parameters:

close false if the ball is in the client's view cone, true otherwise. Since dist_chg, dir_chg, etc are only ever sent if the player is in the client's view cone, *close* is always false, thus this parameter is redundant.

dist The distance (in meters) from the client to the player.

dir The angle from the client's center of view to the player.

dist_chg The distance change of the player relative to the client.

dir_chg The the direction change of the player relative to the client.

orientation The the direction of the player's body relative to the client.

head_orientation The the direction of the player's head relative to the client.

point_dir The the direction of the player is pointing relative to the client.

tackling true if the player is tackling, false otherwise.

Precondition:

The client is using protocol 8 or 9.

Precondition:
The client is a field player.

Precondition:
The client is in a high quality view mode.

Precondition:
An entire near, pointing, player section of a player visual message has been parsed.

Precondition:
A version 8 or 9 player visual message is being parsed.

See also:

- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6356 of file rccparser.h.

7.1.3.143 virtual void rcc::Parser::doBuildPlayer (int *unum*, int *type*, double *pos_x*, double *pos_y*, double *vel_x*, double *vel_y*, double *body_dir*, double *head_dir*, double *arm_mag*, double *arm_dir*, double *stamina*, double *effort*, double *recovery*) `[inline, protected, virtual]`

This function is called after parsing a player section of a version 8 or 9 full state message, where the player **is** pointing and the player is **not** a goalie.

Override this function in your subclass to handle pointing non-goalie player sections of version 8 or 9 full state messages.

Parameters:

unum The uniform number of the player.

type The player's type.

x The x co-ordinate of the player.

y The y co-ordinate of the player.

delta_x The x velocity of the player.

delta_y The y velocity of the player.

orientation The direction the player's body is facing.

head_orientation The direction the player's head is facing.

arm_mag The distance from the center of the field to the point the player is pointing to.

arm_dir The direction the player is pointing in.

stamina The player's stamina.

effort The player's effort.

recovery The player's recovery.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

Full state message sending is turned on in the simulator for the team this client is in.

Precondition:

A pointing non-goalie player section of a version 8 or 9 full state message has been parsed.

The following is an example of a pointing non-goalie player section of a version 8 or 9 full state message:

```
((p l 5 4) -20.5 10.7 0.56 -0.3 13.5 4.8 13.6 -10.2 (stamina 2964.5 0.93 0.54))
```

In this examples the:

- *unum* parameter would be 5.
- *type* parameter would be 4.
- *x* parameter would be -20.5.
- *y* parameter would be 10.7.
- *delta_x* parameter would be 0.56.
- *delta_y* parameter would be -0.3.
- *orientation* parameter would be 13.5.
- *head_orientation* parameter would be 4.8.

- *arm_mag* parameter would be 13.6.
- *arm_dir* parameter would be -10.2.
- *stamina* parameter would be 2964.5.
- *effort* parameter would be 0.93.
- *recovery* parameter would be 0.54.

See also:

- doBuildFullState(int time)
- doBuildLeftSide()
- doBuildRightSide()

Definition at line 2170 of file rccparser.h.

7.1.3.144 virtual void rcc::Parser::doBuildPlayer (int *unum*, int *type*, double *pos_x*, double *pos_y*, double *vel_x*, double *vel_y*, double *body_dir*, double *head_dir*, double *stamina*, double *effort*, double *recovery*) `[inline, protected, virtual]`

This function is called after parsing a player section of a version 8 or 9 full state message, where the player is **not** pointing and the player is **not** a goalie.

Override this function in your subclass to handle non-pointing non-goalie player sections of version 8 or 9 full state messages.

Parameters:

unum The uniform number of the player.

type The player's type.

x The x co-ordinate of the player.

y The y co-ordinate of the player.

delta_x The x velocity of the player.

delta_y The y velocity of the player.

orientation The direction the player's body is facing.

head_orientation The direction the player's head is facing.

stamina The player's stamina.

effort The player's effort.

recovery The player's recovery.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

Full state message sending is turned on in the simulator for the team this client is in.

Precondition:

A non-pointing, non-goalie player section of a version 8 or 9 full state message has been parsed.

The following is an example of a non-pointing non-goalie player section of a version 8 or 9 full state message:

```
((p l 5 4) -20.5 10.7 0.56 -0.3 13.5 4.8 (stamina 2964.5 0.93 0.54))
```

In this examples the:

- *unum* parameter would be 5.
- *type* parameter would be 4.
- *x* parameter would be -20.5.
- *y* parameter would be 10.7.
- *delta_x* parameter would be 0.56.
- *delta_y* parameter would be -0.3.
- *orientation* parameter would be 13.5.
- *head_orientation* parameter would be 4.8.
- *stamina* parameter would be 2964.5.
- *effort* parameter would be 0.93.
- *recovery* parameter would be 0.54.

See also:

- doBuildFullState(int time)
- doBuildLeftSide()
- doBuildRightSide()

Definition at line 2105 of file rccparser.h.

7.1.3.145 virtual void rcc::Parser::doBuildPlayer (int *unum*, double *x*, double *y*, double *delta_x*, double *delta_y*, double *orientation*, double *head_orientation*, int *stamina*, double *effort*, double *recovery*) `[inline, protected, virtual]`

This function is called after parsing a player section of a version 7 full state message.

Override this function in your subclass to handle player sections of version 7 full state messages.

Parameters:

unum	The uniform number of the player.
x	The x co-ordinate of the player.
y	The y co-ordinate of the player.
delta_x	The x velocity of the player.
delta_y	The y velocity of the player.
orientation	The direction the player's body is facing.

head_orientation The direction the player's head is facing.

stamina The player's stamina.

effort The player's effort.

recovery The player's recovery.

Precondition:
The client is using protocol version 7.

Precondition:
The client is a field player.

Precondition:
Full state message sending is turned on in the simulator for the team this client is in.

Precondition:
A player section of a full state message has been parsed.

The following is an example of the player section of a version 7 full state message:

```
(l_5 -20.5 10.7 0.56 -0.3 13.5 4.8 2964.5 0.93 0.54)
```

In this examples the:

- *unum* parameter would be 5.
- *x* parameter would be -20.5.
- *y* parameter would be 10.7.
- *delta_x* parameter would be 0.56.
- *delta_y* parameter would be -0.3.
- *orientation* parameter would be 13.5.
- *head_orientation* parameter would be 4.8.
- *stamina* parameter would be 2965.
- *effort* parameter would be 0.93.
- *recovery* parameter would be 0.54.

See also:
- doBuildFullState(int time)
- doBuildLeftSide()
- doBuildRightSide()

Definition at line 2046 of file rccparser.h.

Referenced by doBuildPlayer().

7.1.3.146 virtual void rcc::Parser::doBuildPlayerAudio (int *time*) `[inline, protected, virtual]`

This function is called after parsing a version 8 opponent player audio message, where the client is a field player and the client has already received an opponent player audio message this cycle.

Override this function in your subclass to handle version 8 incomplete opponent player audio messages.

Precondition:
The client is a field player

Precondition:
The client is using protocol version 8

Precondition:
The message is from an opponent

Precondition:
An entire version 8 incomplete opponent player audio message has been parsed.

Definition at line 7817 of file rccparser.h.

7.1.3.147 virtual void rcc::Parser::doBuildPlayerAudio (int *time*, int *unum*) `[inline, protected, virtual]`

This function is called after parsing a version 8 team-mate player audio message, where the client is a field player and the client has already received a team-mate player audio message this cycle.

Override this function in your subclass to handle version 8 incomplete team-mate player audio messages.

Precondition:
The client is a field player

Precondition:
The client is using protocol version 8

Precondition:
The message is from a team-mate

Precondition:
An entire version 8 incomplete team-mate player audio message has been parsed.

Definition at line 7795 of file rccparser.h.

7.1.3.148 virtual void rcc::Parser::doBuildPlayerAudio (int *time*, double *dir*, int *unum*, const std::string & *msg*) `[inline, protected, virtual]`

This function is called after parsing a version 8 team-mate player audio message, where the client is a field player.

Override this function in your subclass to handle version 8 team-mate player audio messages.

Precondition:
The client is a field player

Precondition:
The client is using protocol version 8

Precondition:
The message is from a team-mate

Precondition:
An entire version 8 team-mate player audio message has been parsed.

Definition at line 7772 of file rccparser.h.

7.1.3.149 virtual void rcc::Parser::doBuildPlayerAudio (int *time*, const std::string & *msg*) `[inline, protected, virtual]`

This function is called after parsing a player audio message, where the client is an online coach.

Override this function in your subclass to player audio messages to online coaches

Precondition:
The client is an online coach

Precondition:
An entire player audio message has been parsed.

Definition at line 7752 of file rccparser.h.

7.1.3.150 virtual void rcc::Parser::doBuildPlayerAudio (int *time*, double *dir*, const std::string & *msg*) `[inline, protected, virtual]`

This function is called after parsing a version 7 player audio message or a version 8 opponent player audio message, where the client is a field player.

Override this function in your subclass to handle version 7 player audio or version 8 opponent player audio messages.

Precondition:
The client is a field player

Precondition:
Either:

- The client is using protocol version 7, or
- The audio message is from an opponent

Precondition:
An entire version 7 player audio or version 8 opponent player audio message has been parsed.

Definition at line 7737 of file rccparser.h.

7.1.3.151 virtual void rcc::Parser::doBuildPlayerParam () `[inline, protected, virtual]`

This function is called after parsing a version 8 or 9 player parameter message.

Override this function in your subclass to handle version 8 or 9 player parameter messages.

Precondition:
The client is using protocol version 8 or 9

Precondition:
A version 8 or 9 player param message has just been parsed.

The following is an example (with line breaks added for clarity) of a version 8 or 9 player param message:

```
(player_param (player_types 7)(pt_max 3)(random_seed -1)(subs_max 3)
              (dash_power_rate_delta_max 0)(dash_power_rate_delta_min 0)
              (effort_max_delta_factor -0.002)(effort_min_delta_factor -0.002)
              (extra_stamina_delta_max 100)(extra_stamina_delta_min 0)
              (inertia_moment_delta_factor 25)(kick_rand_delta_factor 0.5)
              (kickable_margin_delta_max 0.2)(kickable_margin_delta_min 0)
              (new_dash_power_rate_delta_max 0.002)
              (new_dash_power_rate_delta_min 0)
              (new_stamina_inc_max_delta_factor -10000)
              (player_decay_delta_max 0.2)(player_decay_delta_min 0)
              (player_size_delta_factor -100)(player_speed_max_delta_max 0.2)
              (player_speed_max_delta_min 0)(stamina_inc_max_delta_factor 0))
```

See also:

- doBuildPlayerParam(int player_types, int subs_max, int pt_max, double player_speed_max_delta_min, double player_speed_max_delta_max, double stamina_inc_max_delta_factor, double player_decay_delta_min, double player_decay_delta_max, double inertia_moment_delta_factor, double dash_power_rate_delta_min, double dash_power_rate_delta_max, double player_size_delta_factor, double kickable_margin_delta_min, double kickable_margin_delta_max, double kick_rand_delta_factor, double extra_stamina_delta_min, double extra_stamina_delta_max, double effort_max_delta_factor, double effort_min_delta_factor)

- doBuildParam(const std::string& name, int value)
- doBuildParam(const std::string& name, double value)
- doBuildParam(const std::string& name, const std::string& value)

Definition at line 5333 of file rccparser.h.

7.1.3.152 virtual void rcc::Parser::doBuildPlayerParam (int *player_types*, int *subs_max*, int *pt_max*, double *player_speed_max_delta_min*, double *player_speed_max_delta_max*, double *stamina_inc_max_delta_factor*, double *player_decay_delta_min*, double *player_decay_delta_max*, double *inertia_moment_delta_factor*, double *dash_power_rate_delta_min*, double *dash_power_rate_delta_max*, double *player_size_delta_factor*, double *kickable_margin_delta_min*, double *kickable_margin_delta_max*, double *kick_rand_delta_factor*, double *extra_stamina_delta_min*, double *extra_stamina_delta_max*, double *effort_max_delta_factor*, double *effort_min_delta_factor*) `[inline, protected, virtual]`

This function is called after parsing a version 7 player parameter message.

Override this function in your subclass to handle version 7 player parameter messages.

Parameters:

player_types The total number of player types.

subs_max The total number of substitutions allowed per game.

pt_max The maximum number of non-default player types a team is allowed to have ant any one time.

player_speed_max_delta_min The minimum delta factor for the max player speed.

player_speed_max_delta_max The maximum delta factor for the max player speed.

stamina_inc_max_delta_factor The stamina increment max delta factor.

player_decay_delta_min The minimum delta factor for the player decay.

player_decay_delta_max The maximum delta factor for the player decay.

inertia_moment_delta_factor The inertia moment delta factor.

dash_power_rate_delta_min The minimum delta factor for the dash power rate.

dash_power_rate_delta_max The maximum delta factor for the dash power rate.

player_size_delta_factor The player size delta factor.

kickable_margin_delta_min The minimum delta factor for the kickable margin.

kickable_margin_delta_max The maximum delta factor for the kickable margin.

kick_rand_delta_factor The kick rand delta factor.

extra_stamina_delta_min The minimum delta factor for the extra stamina.

extra_stamina_delta_max The maximum delta factor for the extra stamina.

effort_max_delta_factor The effort max delta factor.

effort_min_delta_factor The effort min delta factor.

Precondition:
The client is using protocol version 7

Precondition:
A version 7 player parameter meaage has just been parsed.

The following is an example (with line breaks added for clarity) of a version 7 player param message:

```
(player_param 7 3 3 0 0.2 -100 0 0.2 25 0 0.002
              -100 0 0.2 0.5 0 100 -0.002 -0.002)
```

See also:
- doBuildPlayerParam()

Definition at line 5257 of file rccparser.h.

7.1.3.153 virtual void rcc::Parser::doBuildPlayerType () `[inline, protected, virtual]`

This function is called after parsing a version 8 or 9 player type message.

Override this function in your subclass to handle version 8 or 9 player type messages.

Precondition:
The client is using protocol version 8 or 9

Precondition:
A version 8 or 9 player type message has just been parsed.

The following is an example (with line breaks added for clarity) of a version 8 or 9 player type message:

```
(player_type (id 0) (player_speed_max 1) (stamina_inc_max 45)
             (player_decay 0.4) (inertia_moment 5) (dash_power_rate 0.006)
             (player_size 0.3) (kickable_margin 0.7) (kick_rand 0)
             (extra_stamina 0) (effort_max 1) (effort_min 0.6))
```

See also:
- doBuildPlayerType(int id, double player_speed_max, double stamina_inc_max, double player_decay, double inertia_moment, double dash_power_rate, double player_size, double kickable_margin, double kick_rand, double extra_stamina, double effort_max, double effort_min)

- doBuildParam(const std::string& name, int value)
- doBuildParam(const std::string& name, double value)
- doBuildParam(const std::string& name, const std::string& value)

Definition at line 5437 of file rccparser.h.

7.1.3.154 virtual void rcc::Parser::doBuildPlayerType (int *id*, double *player_speed_max*, double *stamina_inc_max*, double *player_decay*, double *inertia_moment*, double *dash_power_rate*, double *player_size*, double *kickable_margin*, double *kick_rand*, double *extra_stamina*, double *effort_max*, double *effort_min*) `[inline, protected, virtual]`

This function is called after parsing a version 7 player type message.

Override this function in your subclass to handle version 7 player type messages.

Parameters:

id The player type's ID

player_speed_max The maximum speed of the player type.

stamina_inc_max The maximum stamina increment of the player type.

player_decay The velocity decay of the player type.

inertia_moment The player type's inertia in turns.

dash_power_rate The dash power rate of the player type.

player_size The size of the player type.

kickable_margin The kickable margin of the player type.

kick_rand The percent of random nosie in kicks by this player type.

extra_stamina The extra stamina for this player type.

effort_max The maximum effort for this player type.

effort_min The minimum effort for this player type.

Precondition:

The client is using protocol version 7

Precondition:

A version 7 player type meaage has just been parsed.

The following is an exampleof a version 7 player type message:

```
(player_type 0 1 45 0.4 5 0.006 0.3 0.7 0 0 1 0.6)
```

In this example the:

- *id* parameter would be 0.
- *player_speed_max* parameter would be 1.

- *stamina_inc_max* parameter would be 45.
- *player_decay* parameter would be 0.4.
- *inertia_moment* parameter would be 5.
- *dash_power_rate* parameter would be 0.006.
- *player_size* parameter would be 0.3.
- *kickable_margin* parameter would be 0.7.
- *kick_rand* parameter would be 0.
- *extra_stamina* parameter would be 0.
- *effort_max* parameter would be 1.
- *effort_min* parameter would be 0.6.

See also:

- doBuildPlayerType()

Definition at line 5383 of file rccparser.h.

7.1.3.155 virtual void rcc::Parser::doBuildPlayerVisBall (double *dist*, double *dir*, double *dist_chg*, double *dir_chg*) `[inline, protected, virtual]`

This function is called after parsing a ball section of player visual message, when the client is in a high quality view mode and the ball is not far away.

Override this function in your subclass to handle high quality near ball sections of player visual messages.

Note:

This function deprecates doBuildBall(bool close, double dist, double dir, double dist_chg, double dir_chg).

Parameters:

dist The distance (in meters) from the client to the ball.

dir The angle from the client's center of view to the ball.

dist_chg The distance change of the ball relative to the player.

dir_chg This is the direction change of the ball relative to the client.

Precondition:

The client is a field player.

Precondition:

The client is in a high quality view mode.

Precondition:

An entire ball section of a player visual message has been parsed.

Precondition:

A player visual message is being parsed.

The following is an example of a near ball section of a player visual message:

```
((b) 24.5 -36 0.49 -0.8)
```

In this example the:

- *dist* parameter would be 24.5.
- *dir* parameter would be -36.
- *dist_chg* parameter would be 0.49.
- *dir_chg* parameter would be -0.8.

See also:

- doBuildBall(bool close, double dist, double dir, double dist_chg, double dir_-chg)
- doBuildBall(bool close, double dist, double dir)
- doBuildBall(bool close, double dir)
- doBuildVisual(int time)

Definition at line 6168 of file rccparser.h.

7.1.3.156 virtual void rcc::Parser::doBuildRefAudio (int *time*) `[inline, protected, virtual]`

This function is called after parsing a non-goal referee message.

Override this function in your subclass to handle non-goal referee messages.

Precondition:

An entire non-goal referee message has been parsed.

Definition at line 7688 of file rccparser.h.

7.1.3.157 virtual void rcc::Parser::doBuildRightLocation () `[inline, protected, virtual]`

This function is called after parsing a right location of a field object or a visual.

Override this function in your subclass to handle right field locations.

Precondition:

A right loation of a field object has been parsed.

Precondition:

A field object is being parsed.

Precondition:

A a visual message is being parsed.

Definition at line 7515 of file rccparser.h.

7.1.3.158 virtual void rcc::Parser::doBuildRightSide () `[inline, protected, virtual]`

This function is called after parsing a token representing the right team side.

Override this function in your subclass to handle right team sides.

Precondition:
Either:

- a player section of a full state message is being parsed,
- an init message or reconnect message is being parsed,
- a focus section of a sense body message is being parsed,
- a referee goal audio message is being parsed or
- a non-CLang coach audio message is being parsed.

Precondition:
A token representing the right team has been parsed.

See also:

- doBuildPlayer(int unum, double x, double y, double delta_x, double delta_y, double orientation, double head_orientation, int stamina, double effort, double recovery)
- doBuildPlayer(int unum, int type, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)
- doBuildGoalie(int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double stamina, double effort, double recovery)
- doBuildGoalie(int unum, double pos_x, double pos_y, double vel_x, double vel_y, double body_dir, double head_dir, double arm_mag, double arm_dir, double stamina, double effort, double recovery)
- doBuildInit(int unum)
- doBuildInit()
- doBuildCoachInit()
- doBuildFocusState(int unum, int count)
- doBuildGoalRefAudio(int time, int score)
- doBuildCoachAudio(int time, const std::string& msg)

Definition at line 3721 of file rccparser.h.

7.1.3.159 virtual void rcc::Parser::doBuildSaidTooManyAdviceMessagesError () `[inline, protected, virtual]`

This function is called after parsing a said too many advice error message.

Override this function in your subclass to handle said too many advice error messages.

Precondition:
The client is an online coach

Precondition:
The client has sent a advice CLang message and exceeded it's current quota

Precondition:
An entire said too many advice error message has been parsed.

Definition at line 8204 of file rccparser.h.

7.1.3.160 virtual void rcc::Parser::doBuildSaidTooManyDefineMessagesError () `[inline, protected, virtual]`

This function is called after parsing a said too many define error message.

Override this function in your subclass to handle said too many define error messages.

Precondition:
The client is an online coach

Precondition:
The client has sent a define CLang message and exceeded it's current quota

Precondition:
An entire said too many define error message has been parsed.

Definition at line 8242 of file rccparser.h.

7.1.3.161 virtual void rcc::Parser::doBuildSaidTooManyFreeformMessagesError () `[inline, protected, virtual]`

This function is called after parsing a said too many freeform error message.

Override this function in your subclass to handle said too many freeform error messages.

Precondition:
The client is an online coach

Precondition:
The client has sent a freeform CLang message and exceeded it's current quota

Precondition:
An entire said too many freeform error message has been parsed.

Definition at line 8147 of file rccparser.h.

7.1.3.162 virtual void rcc::Parser::doBuildSaidTooManyInfoMessagesError () `[inline, protected, virtual]`

This function is called after parsing a said too many info error message.

Override this function in your subclass to handle said too many info error messages.

Precondition:
The client is an online coach

Precondition:
The client has sent a info CLang message and exceeded it's current quota

Precondition:
An entire said too many info error message has been parsed.

Definition at line 8223 of file rccparser.h.

7.1.3.163 virtual void rcc::Parser::doBuildSaidTooManyMessagesError () `[inline, protected, virtual]`

This function is called after parsing a said too many error message.

Override this function in your subclass to handle said too many error messages.

Precondition:
The client is an online coach

Precondition:
The client is using a pre version 7 protocol (this isn't actually supported with this parser)

Precondition:
An entire said too many error message has been parsed.

Definition at line 8279 of file rccparser.h.

7.1.3.164 virtual void rcc::Parser::doBuildSaidTooManyMetaMessagesError () `[inline, protected, virtual]`

This function is called after parsing a said too many meta error message.

Override this function in your subclass to handle said too many meta error messages.

Precondition:
The client is an online coach

Precondition:
The client has sent a meta CLang message and exceeded it's current quota

Precondition:
An entire said too many meta error message has been parsed.

Definition at line 8185 of file rccparser.h.

7.1.3.165 virtual void rcc::Parser::doBuildSayMessageTooLongError () `[inline, protected, virtual]`

This function is called after parsing a say too long error message.

Override this function in your subclass to handle say too long error messages.

Precondition:
The client has sent a say command and the message is longer than those prescribed by the server.

Precondition:
An entire say too long error message has been parsed.

Definition at line 8089 of file rccparser.h.

7.1.3.166 virtual void rcc::Parser::doBuildSayOK () `[inline, protected, virtual]`

This function is called after parsing an `say` OK message.

Override this function in your subclass to handle `say` OK messages.

Precondition:
The client is an online or offline coach

Precondition:
The client sent a `say` command

Precondition:
An entire `say` OK message has been parsed.

Definition at line 8517 of file rccparser.h.

7.1.3.167 virtual void rcc::Parser::doBuildScore (int *time*, int *our*, int *opp*) `[inline, protected, virtual]`

This function is called after parsing a score message.

Override this function in your subclass to handle score messages

Precondition:
An entire score message has been parsed.

Definition at line 7613 of file rccparser.h.

7.1.3.168 virtual void rcc::Parser::doBuildScore (int *our*, int *opp*) `[inline, protected, virtual]`

This function is called after parsing the score section of a full state message.

Override this function in your subclass to handle the score section of a full state message.

Parameters:

our If protocol version 7 is being used, then this is the number of goals the left team has scored. Otherwise this is the number of goals the client's team has scored.

our If protocol version 7 is being used, then this is the number of goals the right team has scored. Otherwise this is the number of goals the opponent team has scored.

Precondition:

The client is a field player.

Precondition:

Full state message sending is turned on in the simulator for the team this client is in.

Precondition:

The score section of the full state message has been parsed.

The following is an example of the score section of a full state message:

```
(score 2 1)
```

In this example the:

- *our* parameter would be 2.
- *opp* parameter would be 1.

See also:

- doBuildFullState(int time)

Definition at line 1956 of file rccparser.h.

7.1.3.169 virtual void rcc::Parser::doBuildSelfAudio (int *time*, const std::string & *msg*) `[inline, protected, virtual]`

This function is called after parsing a player audio message, where the client is a field player and the message was sent by the client.

Override this function in your subclass to handle self player audio messages

Precondition:

The client is a field player

Precondition:
The message is from the client

Precondition:
An entire self player audio message has been parsed.

Definition at line 7836 of file rccparser.h.

7.1.3.170 virtual void rcc::Parser::doBuildSenseBody (int *time*, int *stamina*, int *effort*, double *speed_mag*, double *speed_head*, double *head_angle*, int *kick_count*, int *dash_count*, int *turn_count*, int *say_count*, int *turn_neck_count*, int *catch_count*, int *move_count*, int *chg_view_count*) `[inline, protected, virtual]`

This function is called after parsing a sense body message, if doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count) has not been overridden.

Override this function in your subclass to handle sense body messages. You will also need to overwrite the functions for parsing the subsections of the sense body message if you want to be able to use the data from those subsections.

\Warning This function is deprecated. Override doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count) instead.

Parameters:

time The time (in server cycles) that the message was sent by the simulator.

stamina The client's stamina.

effort The client's effort.

speed_mag The magnitude of the client's velocity.

speed_head The heading of the client's velocity, relative to it's body direction.

head_angle The client's head angle relative to it's body direction.

kick_count The number of kick commands the client has executed.

dash_count The number of dash commands the client has executed.

turn_count The number of turn commands the client has executed.

say_count The number of say commands the client has executed.

turn_neck_count The number of turn neck commands the client has executed.

catch_count The number of catch commands the client has executed.

move_count The number of move commands the client has executed.

chg_view_count The number of change view commands the client has executed.

Precondition:
The client is a field player.

The following is an example (with line breaks added for clarity) of a version 8 or 9 sense body message:

```
(sense_body 2127 (view_mode high normal)
                 (stamina 2694.5 0.93)
                 (speed 0.2 -3.56)
                 (head_angle 12.7)
                 (kick 128)
                 (dash 897)
                 (turn 1256)
                 (say 54)
                 (turn_neck 563)
                 (catch 0)
                 (move 2)
                 (change_view 89)
                 (arm (movable 0) (expires 0) (target 0 0) (count 0))
                 (focus (target none) (count 0))
                 (tackle (expires 0) (count 0)))
```

The following is the same example (with line breaks added for clarity) in the version 7 protocol:

```
(sense_body 2127 (view_mode high normal)
                 (stamina 2694.5 0.93)
                 (speed 0.2 -3.56)
                 (head_angle 12.7)
                 (kick 128)
                 (dash 897)
                 (turn 1256)
                 (say 54)
                 (turn_neck 563)
                 (catch 0)
                 (move 2)
                 (change_view 89)
```

In these examples the:

- *time* parameter would be 2127.
- *stamina* parameter would be 2695.
- *effort* parameter would be 1.
- *speed_mag* parameter would be 0.2.
- *speed_head* parameter would be -3.56.
- *head_angle* parameter would be 12.7.
- *kick_count* parameter would be 128.
- *dash_count* parameter would be 897.
- *turn_count* parameter would be 1256.
- *say_count* parameter would be 54.
- *turn_neck_count* parameter would be 563.
- *catch_count* parameter would be 0.
- *move_count* parameter would be 2.
- *chg_view_count* parameter would be 89.

See also:

- doBuildArmState(int movable_in, int expires_in, double target_x, double target_y, int count)
- doBuildFocusState(int count)
- doBuildFocusState(int unum, int count)
- doBuildTackleState(int expires_in, int count)
- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count).

Definition at line 4052 of file rccparser.h.

7.1.3.171 virtual void rcc::Parser::doBuildSenseBody (int *time*, double *stamina*, double *effort*, double *speed_mag*, double *speed_head*, double *head_angle*, int *kick_count*, int *dash_count*, int *turn_count*, int *say_count*, int *turn_neck_count*, int *catch_count*, int *move_count*, int *chg_view_count*) `[inline, protected, virtual]`

This function is called after parsing a sense body message.

Override this function in your subclass to handle sense body messages. You will also need to overwrite the functions for parsing the subsections of the sense body message if you want to be able to use the data from those subsections.

Note:

This function deprecates doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count). If this function is not overridden, it will call the deprecated version.

Parameters:

time The time (in server cycles) that the message was sent by the simulator.

stamina The client's stamina.

effort The client's effort.

speed_mag The magnitude of the client's velocity.

speed_head The heading of the client's velocity, relative to it's body direction.

head_angle The client's head angle relative to it's body direction.

kick_count The number of kick commands the client has executed.

dash_count The number of dash commands the client has executed.

turn_count The number of turn commands the client has executed.

say_count The number of say commands the client has executed.

turn_neck_count The number of turn neck commands the client has executed.

catch_count The number of catch commands the client has executed.

move_count The number of move commands the client has executed.

chg_view_count The number of change view commands the client has executed.

Precondition:
The client is a field player.

The following is an example (with line breaks added for clarity) of a version 8 or 9 sense body message:

```
(sense_body 2127 (view_mode high normal)
                 (stamina 2694.5 0.93)
                 (speed 0.2 -3.56)
                 (head_angle 12.7)
                 (kick 128)
                 (dash 897)
                 (turn 1256)
                 (say 54)
                 (turn_neck 563)
                 (catch 0)
                 (move 2)
                 (change_view 89)
                 (arm (movable 0) (expires 0) (target 0 0) (count 0))
                 (focus (target none) (count 0))
                 (tackle (expires 0) (count 0)))
```

The following is the same example (with line breaks added for clarity) in the version 7 protocol:

```
(sense_body 2127 (view_mode high normal)
                 (stamina 2694.5 0.93)
                 (speed 0.2 -3.56)
                 (head_angle 12.7)
                 (kick 128)
                 (dash 897)
                 (turn 1256)
                 (say 54)
                 (turn_neck 563)
                 (catch 0)
                 (move 2)
                 (change_view 89)
```

In these examples the:

- *time* parameter would be 2127
- *stamina* parameter would be 2694.5.
- *effort* parameter would be 0.93.
- *speed_mag* parameter would be 0.2.
- *speed_head* parameter would be -3.56.
- *head_angle* parameter would be 12.7.
- *kick_count* parameter would be 128.
- *dash_count* parameter would be 897.
- *turn_count* parameter would be 1256.
- *say_count* parameter would be 54.

- *turn_neck_count* parameter would be 563.
- *catch_count* parameter would be 0.
- *move_count* parameter would be 2.
- *chg_view_count* parameter would be 89.

See also:

- doBuildArmState(int movable_in, int expires_in, double target_x, double target_y, int count)
- doBuildFocusState(int count)
- doBuildFocusState(int unum, int count)
- doBuildTackleState(int expires_in, int count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count).

Definition at line 3898 of file rccparser.h.

7.1.3.172 virtual void rcc::Parser::doBuildServerParam () `[inline, protected, virtual]`

This function is called after parsing a version 8 or 9 server parameter message.

Override this function in your subclass to handle version 8 or 9 server parameter messages.

Precondition:

The client is using protocol version 8 or 9

Precondition:

A version 8 or 9 server param message has just been parsed.

The following is an example (with line breaks added for clarity) of a version 8 or 9 server param message:

```
(server_param (catch_ban_cycle 5)(clang_advice_win 1)(clang_define_win 1)
              (clang_del_win 1)(clang_info_win 1)(clang_mess_delay 50)
              (clang_mess_per_cycle 1)(clang_meta_win 1)(clang_rule_win 1)
              (clang_win_size 300)(coach_port 6001)(drop_ball_time 200)
              (freeform_send_period 20)(freeform_wait_period 600)(game_log_compre
              (game_log_version 3)(goalie_max_moves 2)(half_time 3000)(hear_decay
              (hear_inc 1)(hear_max 1)(max_goal_kicks 3)(olcoach_port 6002)(point
              (point_to_duration 20)(port 6000)(recv_step 10)(say_coach_cnt_max 1
              (say_coach_msg_size 128)(say_msg_size 10)(send_step 150)(send_vi_st
              (sense_body_step 100)(simulator_step 100)(slow_down_factor 1)(start
              (start_goal_r 0)(synch_micro_sleep 1)(synch_offset 60)(tackle_cycle
              (text_log_compression 0)(game_log_dir "/home/thoward/data")
              (game_log_fixed_name "rcssserver")(landmark_file "~/.rcssserver-lan
              (log_date_format "%Y%m%d%H%M-")(text_log_dir "/home/thoward/data")
              (text_log_fixed_name "rcssserver")(coach 0)(coach_w_referee 1)
              (old_coach_hear 0)(wind_none 0)(wind_random 0)(back_passes 1)
              (forbid_kick_off_offside 1)(free_kick_faults 1)(fullstate_l 1)(full
(game_log_dated 1)(game_log_fixed 1)(game_logging 1)(log_times 0)(profile 1)
(proper_goal_kicks 0)(record_messages 0)(send_comms 0)(synch_mode 0)
(team_actuator_noise 0)(text_log_dated 1)(text_log_fixed 1)(text_logging 1)
(use_offside 1)(verbose 0)(audio_cut_dist 50)(ball_accel_max 2.7)
(ball_decay 0.94)(ball_rand 0.05)(ball_size 0.085)(ball_speed_max 2.7)
(ball_weight 0.2)(catch_probability 1)(catchable_area_l 2)(catchable_area_w 1)
(ckick_margin 1)(control_radius 2)(dash_power_rate 0.006)(effort_dec 0.005)
(effort_dec_thr 0.3)(effort_inc 0.01)(effort_inc_thr 0.6)(effort_init 0)
(effort_min 0.6)(goal_width 14.02)(inertia_moment 5)(kick_power_rate 0.027)
(kick_rand 0)(kick_rand_factor_l 1)(kick_rand_factor_r 1)(kickable_margin 0.7)
(maxmoment 180)(maxneckang 90)(maxneckmoment 180)(maxpower 100)
(minmoment -180)(minneckang -90)(minneckmoment -180)(minpower -100)
(offside_active_area_size 2.5)(offside_kick_margin 9.15)(player_accel_max 1)
(player_decay 0.4)(player_rand 0.1)(player_size 0.3)(player_speed_max 1)
(player_weight 60)(prand_factor_l 1)(prand_factor_r 1)(quantize_step 0.1)
(quantize_step_l 0.01)(recover_dec 0.002)(recover_dec_thr 0.3)
(recover_min 0.5)(slowness_on_top_for_left_team 1)
(slowness_on_top_for_right_team 1)(stamina_inc_max 45)(stamina_max 4000)
(stopped_ball_vel 0.01)(tackle_back_dist 0.5)(tackle_dist 2.5)
(tackle_exponent 6)(tackle_power_rate 0.027)(tackle_width 1.25)
(visible_angle 90)(visible_distance 3)(wind_ang 0)(wind_dir 0)(wind_force 0)
(wind_rand 0))
```

See also:

- doBuildServerParam(double gwidth, double inertia_moment, double psize, double pdecay, double prand, double pweight, double pspeed_max, double paccel_max, double stamina_max, double stamina_inc, double recover_init, double recover_dthr, double recover_min, double recover_dec, double effort_init, double effort_dthr, double effort_min, double effort_dec, double effort_ithr, double effort_inc, double kick_rand, int team_actuator_noise, double prand_factor_l, double prand_factor_r, double kick_rand_factor_l, double kick_rand_factor_r, double bsize, double bdecay, double brand, double bweight, double bspeed_max, double baccel_max, double dprate, double kprate, double kmargin, double ctlradius, double ctlradius_width, double maxp, double minp, double maxm, double minm, double maxnm, double minnm, double maxn, double minn, double visangle, double visdist, double windir, double winforce, double winang, double winrand, double kickable_area, double catch_area_l, double catch_area_w, double catch_prob, int goalie_max_moves, double ckmargin, double offside_area, int win_no, int win_random, int say_cnt_max, int SayCoachMsgSize, int clang_win_size, int clang_define_win, int clang_meta_win, int clang_advice_win, int clang_info_win, int clang_mess_delay, int clang_mess_per_cycle, int half_time, int sim_st, int send_st, int recv_st, int sb_step, int lcm_st, int SayMsgSize, int hear_max, int hear_inc, int hear_decay, int cban_cycle, int slow_down_factor, int useoffside, int kickoffoffside, double offside_kick_margin, double audio_dist, double dist_qstep, double land_qstep, double dir_qstep, double dist_qstep_l, double dist_qstep_r, double land_qstep_l, double land_qstep_r, double dir_qstep_l, double dir_qstep_r, int CoachMode, int CwRMode, int old_hear, int sv_st, int start_goal_l, int start_goal_r, int fullstate_l, int fullstate_r, int drop_time)
- doBuildParam(const std::string& name, int value)
- doBuildParam(const std::string& name, double value)
- doBuildParam(const std::string& name, const std::string& value)

Definition at line 5085 of file rccparser.h.

7.1.3.173 virtual void rcc::Parser::doBuildServerParam (double *gwidth*, double *inertia_moment*, double *psize*, double *pdecay*, double *prand*, double *pweight*, double *pspeed_max*, double *paccel_max*, double *stamina_max*, double *stamina_inc*, double *recover_init*, double *recover_dthr*, double *recover_min*, double *recover_dec*, double *effort_init*, double *effort_dthr*, double *effort_min*, double *effort_dec*, double *effort_ithr*, double *effort_inc*, double *kick_rand*, int *team_actuator_noise*, double *prand_factor_l*, double *prand_factor_r*, double *kick_rand_factor_l*, double *kick_rand_factor_r*, double *bsize*, double *bdecay*, double *brand*, double *bweight*, double *bspeed_max*, double *baccel_max*, double *dprate*, double *kprate*, double *kmargin*, double *ctlradius*, double *ctlradius_width*, double *maxp*, double *minp*, double *maxm*, double *minm*, double *maxnm*, double *minnm*, double *maxn*, double *minn*, double *visangle*, double *visdist*, double *windir*, double *winforce*, double *winang*, double *winrand*, double *kickable_area*, double *catch_area_l*, double *catch_area_w*, double *catch_prob*, int *goalie_max_moves*, double *ckmargin*, double *offside_area*, int *win_no*, int *win_random*, int *say_cnt_max*, int *SayCoachMsgSize*, int *clang_win_size*, int *clang_define_win*, int *clang_meta_win*, int *clang_advice_win*, int *clang_info_win*, int *clang_mess_delay*, int *clang_mess_per_cycle*, int *half_time*, int *sim_st*, int *send_st*, int *recv_st*, int *sb_step*, int *lcm_st*, int *SayMsgSize*, int *hear_max*, int *hear_inc*, int *hear_decay*, int *cban_cycle*, int *slow_down_factor*, int *useoffside*, int *kickoffoffside*, double *offside_kick_margin*, double *audio_dist*, double *dist_qstep*, double *land_qstep*, double *dir_qstep*, double *dist_qstep_l*, double *dist_qstep_r*, double *land_qstep_l*, double *land_qstep_r*, double *dir_qstep_l*, double *dir_qstep_r*, int *CoachMode*, int *CwRMode*, int *old_hear*, int *sv_st*, int *start_goal_l*, int *start_goal_r*, int *fullstate_l*, int *fullstate_r*, int *drop_time*) `[inline, protected, virtual]`

This ungainly function is called after parsing a version 7 server parameter message.

Override this function in your subclass to handle version 7 server parameter messages.

Parameters:

gwidth The width of the goals.

inertia_moment The players inertia during a turn.

psize The size of the default player.

pdecay The velocity decay of the default player.

prand The maxmuim percent of random motion in player for each cycle.

pweight The weight of the player.

pspeed_max The maximum velocity of the default player.

paccel_max The maximum acceleration of the players.

stamina_max The maximum stamina of the players.

stamina_inc The maximum stamina increment of the default player.

recover_init The initial recovery value for players.

recover_dthr The recovery decrement threshold for players.

recover_min The minimum recovery a player can have.

recover_dec The player recovery decrement.

effort_init The intial and maximum effort for the default player.

effort_dthr The effort decrement threshold for players.

effort_min The minimum effort a default player can have.

effort_dec The player effort decrement.

effort_ithr The effort increment threshold for players.

effort_inc The player effort increment.

kick_rand The percent of random noise added to kicks.

team_actuator_noise The flag for whether team specific actuator noise is used.

prand_factor_l The prand multiplication factor for the left team.

prand_factor_r The prand multiplication factor for the right team.

kick_rand_factor_l The kick_rand multiplication factor for the left team.

kick_rand_factor_r The kick_rand multiplication factor for the right team.

bsize The size of the ball.

bdecay The velocity decay of the ball.

brand The maxmuim percent of random motion in the ball for each cycle.

bweight The weight of the ball.

bspeed_max The maximum speed of the ball.

baccel_max The maximum velocity of the ball.

dprate The dash power rate for the default player.

kprate The kick power rate for players.

kmargin The kickable margin for the default player.

ctlradius Obsolete and no longer used.

ctlradius_width Obsolete and no longer used.

maxp The maximum power that can be used by a player in one command.

minp The minimum power that can be used by a player in one command.

maxm The maximum moment that can be used by a player in one command.

minm The minimum moment that can be used by a player in one command.

maxnm The maximum moment that can be used by a player in one turn neck command.

minnm The minimum moment that can be used by a player in one command.

maxn The maximum a player can turn it's neck.

minn The minimum a player can turn it's neck.

visangle The view width of a player in `normal` view mode.

visdist The distance under which objects can be ”sensed” by players.

windir The direction of the wind.

winforce The force of the wind.

winang Obsolete and no longer used.

winrand The randomisation factor in the wind.

kickable_area kmargin + bsize + psize.

catch_area_l The goalie catchable area length.

catch_area_w The goalie catchable area width.

catch_prob The goalie cacth probability.

goalie_max_moves The maximum number of move a goalie can make after a catch and before releasing the ball.

ckmargin The distance from the side and goal line that the ball is placed when a corner kick is awarded.

offside_area The distance to the ball that under which an offside player is considered to be in active offside.

win_no The flag for no wind.

win_random The flag for random wind.

say_cnt_max The maximum number of say messages an online coach can execute in one game.

SayCoachMsgSize The maximum size of a online coach non-CLang or freeform say message.

clang_win_size The size of each CLang send window.

clang_define_win The number of CLang define messages allowed per CLang window.

clang_meta_win The number of CLang meta messages allowed per Clang window.

clang_advice_win The number of CLang advice messages allowed per Clang window.

clang_info_win The number of CLang info messages allowed per Clang window.

clang_mess_delay The number of cycles the server will delay CLang messages before sending them to the players.

clang_mess_per_cycle The maximum number of CLang messages that a coach can be send per cycle.

half_time The length of a game half (in seconds).

sim_st The length of a simulator step (in ms).

send_st The length between player visuals when the player is in the `normal high` view mode(in ms).

recv_st The length between the simulator checking for client messages.

sb_step The length between player sense body messages.

lcm_st The lowest common multiple of the above steps.

SayMsgSize The maximum size of a player say message.

hear_max The maximum hearing capacity of a player.

hear_inc The player hear capacity increament per cycle.

hear_decay The amount the player hear capacity is reduced by every time they hear a player audio message.

cban_cycle The number of cycles for which a goalie cannot catch after executing a catch.

slow_down_factor The amount the sim_st, send_st, recv_st and sb_step have been mulitplied by.

useoffside The flag for whether offsides are used.

kickoffoffside The flag for whether players are moved to a onside position before kickoffs.

offside_kick_margin The radius of the area that opponent players are moved from the ball when an offside has been awarded.

audio_dist The maximum distance from which players can hear each other.

dist_qstep The quatization factor for movable objects (players and the ball).

land_qstep The quatization factor for fixed objects (flags).

dir_qstep The quatization factor for direction change in a objects velocity.

dist_qstep_l The dist_qstep for the left team.

dist_qstep_r The dist_qstep for the right team.

land_qstep_l The land_qstep for the left team.

land_qstep_r The land_qstep for the right team.

dir_qstep_l The dir_qstep for the left team.

dir_qstep_r The dir_qstep for the right team.

CoachMode The flag for if the offline coach is allowed with the referee disabled.

CwRMode The flag for if the offline coach is allowed with the referee enabled.

old_hear Obsolete and no longer used.

sv_st The time between sending coach visuals.

start_goal_l The number of goals that the left team has started on.

start_goal_r The number of goals that the right teams has started on.

fullstate_l The flag for whether full state messages are sent to the left players.

fullstate_r The flag for whether full state messages are sent to the right players.

drop_time The amount of time after a freekick in which the play mode is changed to drop ball if the kick has not been taken.

Precondition:
The client is using protocol version 7

Precondition:
A server param message has just been parsed.

The following is an example (with line breaks added for what every clarity we can get) of a version 7 server param message:

```
(server_param 14.02 5 0.3 0.4 0.1 60 1 1 4000 45 0 0.3 0.5 0.002
              0 0.3 0.6 0.005 0.6 0.01 0 0 1 1 1 1 0.085 0.94 0.05
              0.2 2.7 2.7 0.006 0.027 0.7 2 1.7 100 -100 180 -180
              180 -180 90 -90 90 3 0 0 0 0 1.085 2 1 1 2 1 5 0 0
              128 128 300 1 1 1 1 50 1 3000 100 150 10 100 300 512
              2 1 2 5 1 1 1 9.15 50 0.1 0.01 -1 -1 -1 -1 -1 -1 -1
              0 0 0 100 0 0 0 0 200)
```

See also:
- doBuildServerParam()

Definition at line 4815 of file rccparser.h.

7.1.3.174 virtual void rcc::Parser::doBuildSubstitution (int *unum*) `[inline, protected, virtual]`

This function is called after parsing a substitutution message, where the player being sub'ed is an opponent.

Override this function in your subclass to handle opponent substitution messages.

Precondition:
The client is a field player

Precondition:
An opponent has been substituted.

Precondition:
An entire opponent substitution message has been parsed.

Definition at line 7877 of file rccparser.h.

7.1.3.175 virtual void rcc::Parser::doBuildSubstitution (int *unum*, int *type*) `[inline, protected, virtual]`

This function is called after parsing a substitutution message, where the player being sub'ed is a team-mate.

Override this function in your subclass to handle team-mate substitution messages.

Precondition:
The client is a field player

Precondition:
A team-mate has been substituted.

Precondition:
An entire team-mate substitution message has been parsed.

Definition at line 7859 of file rccparser.h.

7.1.3.176 virtual void rcc::Parser::doBuildTackleState (int *expires_in*, int *count*) `[inline, protected, virtual]`

This function is called after parsing a tackle state section of a version 8 or 9 sense body.

Override this function in your subclass to handle the tackle state section of version 8 or 9 sense body messages.

Parameters:

expires_in The number of cycles till client finishes tackling.

count The number of times the client has excuted an tackle command.

Precondition:

The client is using protocol version 8 or 9.

Precondition:

The client is a field player.

Precondition:

A version 8 or 9 sense body message is being parsed.

Precondition:

A tackle state section of a version 8 or 9 sense body message has just been parsed.

The following is an example of the focused focus section of a version 8 or 9 sense body message:

```
(tackle (expires 3) (count 18))
```

In this examples the:

- *expires_in* parameter would be 3.
- *count* parameter would be 18.

See also:

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)

Definition at line 4327 of file rccparser.h.

7.1.3.177 virtual void rcc::Parser::doBuildTeamName (const std::string & *name*) `[inline, protected, virtual]`

This function is called after parsing a team name.

Override this function in your subclass to handle team names.

Precondition:
A team name has been parsed.

Definition at line 7578 of file rccparser.h.

7.1.3.178 virtual void rcc::Parser::doBuildTimeOverPlayMode () `[inline, protected, virtual]`

This function is called after parsing of a `time_over` play mode.

Override this function in your subclass to handle the `time_over` play mode.

Precondition:
Either:

- a full state message is being parsed,
- a init or reconnect message is being parsed or
- a referee audio message is beign parsed.

Precondition:
A `time_over` play mode has just been parsed.

See also:

- doBuildFullState(int time)
- doBuildInit(int unum)
- doBuildInit()
- doBuildRefAudio(int time)

Definition at line 2389 of file rccparser.h.

7.1.3.179 virtual void rcc::Parser::doBuildTooManyMovesError () `[inline, protected, virtual]`

This function is called after parsing a too many moves error message.

Override this function in your subclass to handle too many moves error messages.

Precondition:
The client is a field player.

Precondition:
The client is a goalie.

Precondition:
The client has caught the ball.

Precondition:
The client has already moved the prescribed maximum amount of times while the ball is caught and another move command was sent.

Precondition:
An entire too many moves error message has been parsed.

Definition at line 8042 of file rccparser.h.

7.1.3.180 virtual void rcc::Parser::doBuildTopLocation () `[inline, protected, virtual]`

This function is called after parsing a top location of a field object or a visual.

Override this function in your subclass to handle top field locations.

Precondition:
A top loation of a field object has been parsed.

Precondition:
A field object is being parsed.

Precondition:
A a visual message is being parsed.

Definition at line 7463 of file rccparser.h.

7.1.3.181 virtual void rcc::Parser::doBuildUNum (int *unum*) `[inline, protected, virtual]`

This function is called after parsing a uniform number.

Override this function in your subclass to handle uniform numbers.

Precondition:
A uniform number has been parsed.

Definition at line 7589 of file rccparser.h.

7.1.3.182 virtual void rcc::Parser::doBuildUnknownCommandError () `[inline, protected, virtual]`

This function is called after parsing an unknown command error message.

Override this function in your subclass to handle unknown command error messages.

Precondition:
The client has send a command that the server does not recognise

Precondition:
An entire unknown commad error message has been parsed.

Definition at line 8058 of file rccparser.h.

7.1.3.183 virtual void rcc::Parser::doBuildUnknownError (const std::string & *error*) `[inline, protected, virtual]`

This function is called after parsing an unknown error message.

Override this function in your subclass to handle unknown error messages.

Unknown error message only occur when the parser is used with a newer version of the simulator and the parser has not been updated. By using this function you can provide support in your teams for new error messages without having to wait for the parser to be upgraded.

Precondition:
An entire unknown error message has been parsed.

Definition at line 8299 of file rccparser.h.

7.1.3.184 virtual void rcc::Parser::doBuildUnknownOK (const std::string & *msg*) `[inline, protected, virtual]`

This function is called after parsing an unknown OK message.

Override this function in your subclass to handle unknown OK messages.

Unknown OK message only occur when the parser is used with a newer version of the simulator and the parser has not been updated. By using this function you can provide support in your teams for new OK messages without having to wait for the parser to be upgraded.

Precondition:
An entire unknown OK message has been parsed.

Definition at line 8571 of file rccparser.h.

7.1.3.185 virtual void rcc::Parser::doBuildUnknownRefAudio (int *time*, const std::string & *message*) `[inline, protected, virtual]`

This function is called after parsing an unknown referee message.

This will normally only be called if new referee messages have been added to the simulator, but not to this parser. By using this function you can provide support in your team for the new messages without having to wait for the parser to be updated.

Override this function in your subclass to handle unknown referee messages.

Precondition:
An entire unknown referee message has been parsed.

Definition at line 7716 of file rccparser.h.

7.1.3.186 virtual void rcc::Parser::doBuildUnknownWarning (const std::string & *warning*) `[inline, protected, virtual]`

This function is called after parsing an unknown warning message.

Override this function in your subclass to handle unknown warning messages.

Unknown warning message only occur when the parser is used with a newer version of the simulator and the parser has not been updated. By using this function you can provide support in your teams for new warning messages without having to wait for the parser to be upgraded.

Precondition:
An entire unknown warning message has been parsed.

Definition at line 8458 of file rccparser.h.

7.1.3.187 virtual void rcc::Parser::doBuildViewMode () `[inline, protected, virtual]`

This function is called after parsing a view mode section of a sense body of full state message.

Override this function in your subclass to handle the view mode section of sense body or full state messages.

Precondition:
The client is a field player.

Precondition:
Either:

- a sense body message is being parsed.
- a full state message is being parsed.

Precondition:
The view mode section of a sense body or full state message has just been parsed.

The following is an example of the view_mode section of a sense body message:

```
(view_mode high normal)
```

The following is the same example as part of a full state message:

```
(vmode high normal)
```

See also:

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildFullState(int time)
- doBuildViewQualityHigh()
- doBuildViewQualityLow()
- doBuildViewWidthNarrow()
- doBuildViewWidthNormal()
- doBuildViewWidthWide()

Definition at line 4396 of file rccparser.h.

7.1.3.188 virtual void rcc::Parser::doBuildViewQualityHigh () `[inline, protected, virtual]`

This function is called after parsing a `high` view quality in the view mode section of a sense body or full state message.

Override this function in your subclass to handle `high` view qualities in the view mode section of a sense body or full state message.

Precondition:
The client is a field player.

Precondition:
Either:

- a sense body message is being parsed.
- a full state message is being parsed.

Precondition:
The view mode section of a sense body or full state is being parsed.

Precondition:
The `high` view quality has just been parsed.

See also:

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)

- doBuildFullState(int time)
- doBuildViewMode()
- doBuildViewQualityLow()

Definition at line 4452 of file rccparser.h.

7.1.3.189 virtual void rcc::Parser::doBuildViewQualityLow () `[inline, protected, virtual]`

This function is called after parsing a `low` view quality in the view mode section of a sense body or full state message.

Override this function in your subclass to handle `low` view qualities in the view mode section of a sense body or full state message.

Precondition:
The client is a field player.

Precondition:
Either:

- a sense body message is being parsed.
- a full state message is being parsed.

Precondition:
The view mode section of a sense body or full state is being parsed.

Precondition:
The `low` view quality has just been parsed.

See also:

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildFullState(int time)
- doBuildViewMode()
- doBuildViewQualityHigh()

Definition at line 4508 of file rccparser.h.

7.1.3.190 virtual void rcc::Parser::doBuildViewWidthNarrow () `[inline, protected, virtual]`

This function is called after parsing a `narrow` view width in the view mode section of a sense body or full state message.

Override this function in your subclass to handle `narrow` view qualities in the view mode section of a sense body or full state message.

Precondition:
The client is a field player.

Precondition:
Either:

- a sense body message is being parsed.
- a full state message is being parsed.

Precondition:
The view mode section of a sense body or full state is being parsed.

Precondition:
The `narrow` view width has just been parsed.

See also:

- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildFullState(int time)
- doBuildViewMode()
- doBuildViewWidthNormal()
- doBuildViewWidthWide()

Definition at line 4565 of file rccparser.h.

7.1.3.191 virtual void rcc::Parser::doBuildViewWidthNormal () `[inline, protected, virtual]`

This function is called after parsing a `normal` view width in the view mode section of a sense body or full state message.

Override this function in your subclass to handle `normal` view qualities in the view mode section of a sense body or full state message.

Precondition:
The client is a field player.

Precondition:
Either:

- a sense body message is being parsed.
- a full state message is being parsed.

Precondition:
The view mode section of a sense body or full state is being parsed.

Precondition:
The `normal` view width has just been parsed.

See also:
- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildFullState(int time)
- doBuildViewMode()
- doBuildViewWidthNarrow()
- doBuildViewWidthWide()

Definition at line 4622 of file rccparser.h.

7.1.3.192 virtual void rcc::Parser::doBuildViewWidthWide () `[inline, protected, virtual]`

This function is called after parsing a `wide` view width in the view mode section of a sense body or full state message.

Override this function in your subclass to handle `wide` view qualities in the view mode section of a sense body or full state message.

Precondition:
The client is a field player.

Precondition:
Either:

- a sense body message is being parsed.
- a full state message is being parsed.

Precondition:
The view mode section of a sense body or full state is being parsed.

Precondition:
The `wide` view width has just been parsed.

See also:
- doBuildSenseBody(int time, double stamina, double effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)

- doBuildSenseBody(int time, int stamina, int effort, double speed_mag, double speed_head, double head_angle, int kick_count, int dash_count, int turn_count, int say_count, int turn_neck_count, int catch_count, int move_count, int chg_view_count)
- doBuildFullState(int time)
- doBuildViewMode()
- doBuildViewWidthNarrow()
- doBuildViewWidthNormal()

Definition at line 4679 of file rccparser.h.

7.1.3.193 virtual void rcc::Parser::doBuildVisual (int *time*) `[inline, protected, virtual]`

This function is called after parsing a player visual message.

Override this function in your subclass to handle player visual messages.

Parameters:

time The time (in server cycles) that the player visual message was sent by the simulator.

Precondition:

The client is a field player.

Precondition:

An entire player visual message has been parsed.

The following is an example (with line breaks added for clarity) of a player visual message:

```
(see 101 ((g r) 72.2 0)
         ((f c) 20.1 0 -0 0)
         ((f r t) 79.8 -25)
         ((f r b) 79.8 25)
         ((f g r b) 73 6)
         ((f g r t) 73 -6)
         ((f p r b) 59.7 20)
         ((f p r c) 56.3 0)
         ((f p r t) 59.7 -20)
         ((f t r 20) 55.7 -44)
         ((f t r 30) 63.4 -38)
         ((f t r 40) 71.5 -33)
         ((f t r 50) 79.8 -29)
         ((f b r 20) 55.7 44)
         ((f b r 30) 63.4 38)
         ((f b r 40) 71.5 33)
         ((f b r 50) 79.8 29)
         ((f r 0) 77.5 0)
         ((f r t 10) 78.3 -7)
         ((f r t 20) 79.8 -14)
         ((f r t 30) 83.1 -21)
         ((f r b 10) 78.3 7)
((f r b 20) 79.8 14)
((f r b 30) 83.1 21)
((b) 24.5 -13 0.98 0.6)
((p "one" 2) 22.2 32 0 -0 -81 -82)
((p "one" 3) 22.2 -36 0 -0.3 104 65)
((p "one" 4) 22.2 -17 -0 -0.1 179 -180)
((p "one") 24.5 0)
((p "one" 6) 27.1 27 0 -0 -100 -100)
((p "one" 7) 22.2 -14 0.444 0.2 -1 0)
((p "one" 8) 20.1 -7 -0 0.1 -101 -68)
((p "one" 9) 30 35 0 -0 -101 -101)
((p "one" 10) 33.1 -42 0 -0 109 109)
((p "two") 30 -28)
((p "two") 30 -28)
((p "two" 3) 24.5 20 -0 -0 -87 -87)
((p "two" 5) 27.1 1 0 0 -170 -135)
((p "two" 6) 27.1 4 0 0 -124 -159)
((p "two" 7) 22.2 34 0 -0 -12 -80)
((p "two" 8) 14.9 5 0 -0 -4 -33)
((p "two" 10) 24.5 40 0 -0 -77 -78)
((p) 66.7 0)
((l r) 72.2 90))
```

In this example the

- *time* parameter would be 101.

See also:

- doBuildLine(double dist, double dir)
- doBuildLine(double dir)
- doBuildFlag(double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildFlag(bool close, double dist, double dir)
- doBuildFlag(bool close, double dir)
- doBuildGoal(bool close, double dist, double dir)
- doBuildGoal(bool close, double dir)
- doBuildBall(bool close, double dist, double dir, double dist_chg, double dir_-chg)
- doBuildBall(bool close, double dist, double dir)
- doBuildBall(bool close, double dir)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double point_dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, bool tackling)
- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg, double orientation, double head_orientation)

- doBuildPlayer(bool close, double dist, double dir, double dist_chg, double dir_chg)
- doBuildPlayer(bool close, double dist, double dir)
- doBuildPlayer(bool close, double dir)

Definition at line 5557 of file rccparser.h.

7.1.3.194 virtual void rcc::Parser::doMsgParsed () `[inline, protected, virtual]`

This function is called every time a message has been parsed.

Override this function in your subclass to execute code **after** a message has been parsed.

Precondition:
A message has been parsed.

Definition at line 1697 of file rccparser.h.

7.1.3.195 virtual void rcc::Parser::doParsedCLang (const char ∗ *msg*) `[inline, protected, virtual]`

This function is called every time a CLang message has been parsed.

Override this function in your subclass to execute code **after** a CLang message has been parsed.

If this function is not overridden then the default behaviour is to call the deprecated function doParsedCLang(const std::string& msg).

Parameters:
msg Contains the CLang message that has just been parsed.

Precondition:
A CLang message has been parsed.

See also:
- doParsedCLang(const std::string& msg).

Definition at line 1742 of file rccparser.h.

7.1.3.196 virtual void rcc::Parser::doParsedCLang (const std::string & *msg*) `[inline, protected, virtual]`

This function is called every time a CLang message has been parsed, if doParsedCLang(const char∗ msg) has not been overridden.

Override this function in your subclass to execute code **after** a CLang message has been parsed.

Parameters:
msg The CLang message that has just been parsed

Warning:
This function is deprecated. Override doParsedCLang(const char∗ msg) instead.

Precondition:
doParsedCLang(const char∗ msg) has not been overridden.

Precondition:
A CLang message has been parsed.

See also:
- doParsedCLang(const char∗ msg).

Definition at line 1721 of file rccparser.h.

Referenced by doParsedCLang().

7.1.3.197 bool rcc::Parser::parse (const std::string & *file*)

Secondary parsing function.

Sometimes it may be usefull to parse data that has been stored in a file (usually to detect errors). This function opens the file specfied and passes the std::ifstream to parse(std::istream& strm).

Parameters:
file The name of the file on which parsing will occur.

Returns:
True if there where no errors, false otherwise.

7.1.3.198 bool rcc::Parser::parse (std::istream & *strm*)

Main parsing function.

This is the main parsing function of the rccparser library. Pass the stream you wish to parse to commence parsing.

Parameters:
strm The stream on which parsing will occur.

Returns:
True if there where no errors, false otherwise.

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

- rccparser.h
- rccparser.cpp

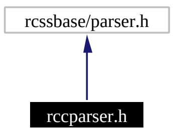
Chapter 8

RoboCup Client Parser (rccparser) File Documentation

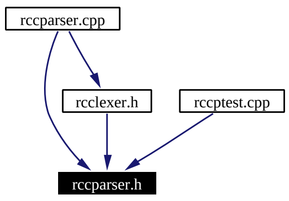
8.1 rccparser.h File Reference

```
#include <rcssbase/parser.h>
```

Include dependency graph for rccparser.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace rcc

Functions

- int RCC_parse (void ∗)

  The Bison parsing function.

8.1.1 Detailed Description

This is the header for rcc::Parser class. It is the main header file for the RCCParser library. It defines the rcc::Parser abstract base class, which your client needs to subclass to prevode custom behaviour during parsing.

Definition in file rccparser.h.

8.1.2 Function Documentation

8.1.2.1 int RCC_parse (void ∗)

The Bison parsing function.

This is the parsing function produced by Bison. You should **not** call this function directly. Instead you should use rcc::Parser::parse(), which makes sure that the correct arguments are passed to RCC_parse().

Index

~Parser
rcc::Parser, 18

doBuildAfterGoalLeftPlayMode
rcc::Parser, 35
doBuildAfterGoalRightPlayMode
rcc::Parser, 35
doBuildArmState
rcc::Parser, 35
doBuildBackPassLeftPlayMode
rcc::Parser, 37
doBuildBackPassRightPlayMode
rcc::Parser, 37
doBuildBall
rcc::Parser, 38–41
doBuildBeforeKickOffPlayMode
rcc::Parser, 42
doBuildBottomLocation
rcc::Parser, 43
doBuildCannotChangeGoalieWarning
rcc::Parser, 43
doBuildCannotSayFreeformWhilePlayonError
rcc::Parser, 43
doBuildCannotSubWhilePlayOnWarning
rcc::Parser, 44
doBuildCantReconnect
rcc::Parser, 44
doBuildCatchBallLeftPlayMode
rcc::Parser, 45
doBuildCatchBallRightPlayMode
rcc::Parser, 45
doBuildCatchFaultLeftPlayMode
rcc::Parser, 46
doBuildCatchFaultRightPlayMode
rcc::Parser, 46
doBuildCenterLocation
rcc::Parser, 46
doBuildChangePlayerTypeOK
rcc::Parser, 47
doBuildClangPlayerVersion
rcc::Parser, 47
doBuildClangPlayerVersionMsg
rcc::Parser, 47
doBuildClangVerOK
rcc::Parser, 48
doBuildCoachAudio
rcc::Parser, 48
doBuildCoachInit
rcc::Parser, 48
doBuildCompressionOK
rcc::Parser, 49
doBuildCompressionWarning
rcc::Parser, 49
doBuildCornerKickLeftPlayMode
rcc::Parser, 49
doBuildCornerKickRightPlayMode
rcc::Parser, 50
doBuildCouldNotParseSayError
rcc::Parser, 50
doBuildCounts
rcc::Parser, 51
doBuildDropBallPlayMode
rcc::Parser, 52
doBuildEyeOK
rcc::Parser, 52
doBuildFirstHalfOverPlayMode
rcc::Parser, 53
doBuildFlag
rcc::Parser, 53, 54, 56, 57
doBuildFlagOffset
rcc::Parser, 58
doBuildFocusState
rcc::Parser, 59, 60
doBuildFoulBallOutLeftPlayMode
rcc::Parser, 61
doBuildFoulBallOutRightPlayMode
rcc::Parser, 61
doBuildFoulChargeLeftPlayMode
rcc::Parser, 62
doBuildFoulChargeRightPlayMode
rcc::Parser, 62
doBuildFoulLeftPlayMode

rcc::Parser, 63
doBuildFoulMultipleAttackerLeftPlayMode
rcc::Parser, 63
doBuildFoulMultipleAttackerRightPlayMode
rcc::Parser, 64
doBuildFoulPushLeftPlayMode
rcc::Parser, 64
doBuildFoulPushRightPlayMode
rcc::Parser, 65
doBuildFoulRightPlayMode
rcc::Parser, 65
doBuildFreeKickFaultLeftPlayMode
rcc::Parser, 66
doBuildFreeKickFaultRightPlayMode
rcc::Parser, 66
doBuildFreeKickLeftPlayMode
rcc::Parser, 67
doBuildFreeKickRightPlayMode
rcc::Parser, 67
doBuildFullState
rcc::Parser, 68
doBuildGlobalBall
rcc::Parser, 70
doBuildGlobalGoal
rcc::Parser, 70
doBuildGlobalPlayer
rcc::Parser, 71–73
doBuildGlobalVisual
rcc::Parser, 73
doBuildGoal
rcc::Parser, 74–77
doBuildGoalie
rcc::Parser, 80, 82
doBuildGoalKickLeftPlayMode
rcc::Parser, 78
doBuildGoalKickRightPlayMode
rcc::Parser, 79
doBuildGoalLocation
rcc::Parser, 79
doBuildGoalRefAudio
rcc::Parser, 80
doBuildHumanPlayMode
rcc::Parser, 83
doBuildIllegalCommandError
rcc::Parser, 83
doBuildIllegalModeError
rcc::Parser, 84
doBuildIllegalObjectFormError
rcc::Parser, 84
doBuildIndirectFreeKickLeftPlayMode
rcc::Parser, 84
doBuildIndirectFreeKickRightPlayMode
rcc::Parser, 85
doBuildInit
rcc::Parser, 85, 86
doBuildInitError
rcc::Parser, 86
doBuildKickInLeftPlayMode
rcc::Parser, 87
doBuildKickInRightPlayMode
rcc::Parser, 87
doBuildKickOffLeftPlayMode
rcc::Parser, 87
doBuildKickOffRightPlayMode
rcc::Parser, 88
doBuildLeftLocation
rcc::Parser, 88
doBuildLeftSide
rcc::Parser, 89
doBuildLine
rcc::Parser, 90
doBuildMaxOfThatPlayerTypeOnFieldWarning
rcc::Parser, 91
doBuildNeutralSide
rcc::Parser, 92
doBuildNoMoreTeamOrPlayerOrGoalieError
rcc::Parser, 92
doBuildNoSubsLeftWarning
rcc::Parser, 93
doBuildNoSuchPlayerWarning
rcc::Parser, 93
doBuildNoSuchTeamOrAlreadyHaveCoachError
rcc::Parser, 94
doBuildNoTeamFoundWarning
rcc::Parser, 94
doBuildOffSideLeftPlayMode
rcc::Parser, 95
doBuildOffsideRightPlayMode
rcc::Parser, 95
doBuildOppSide
rcc::Parser, 96
doBuildOurSide
rcc::Parser, 96
doBuildParam
rcc::Parser, 96–98
doBuildPausePlayMode
rcc::Parser, 99
doBuildPenaltyDrawPlaymode
rcc::Parser, 99
doBuildPenaltyFoulLeftPlaymode

rcc::Parser, 100
doBuildPenaltyFoulRightPlaymode
rcc::Parser, 100
doBuildPenaltyKickLeftPlayMode
rcc::Parser, 101
doBuildPenaltyKickRightPlayMode
rcc::Parser, 101
doBuildPenaltyLocation
rcc::Parser, 101
doBuildPenaltyMissLeftPlaymode
rcc::Parser, 102
doBuildPenaltyMissRightPlaymode
rcc::Parser, 102
doBuildPenaltyOnFieldLeftPlaymode
rcc::Parser, 103
doBuildPenaltyOnFieldRightPlaymode
rcc::Parser, 103
doBuildPenaltyReadyLeftPlaymode
rcc::Parser, 104
doBuildPenaltyReadyRightPlaymode
rcc::Parser, 104
doBuildPenaltyScoreLeftPlaymode
rcc::Parser, 104
doBuildPenaltyScoreRightPlaymode
rcc::Parser, 105
doBuildPenaltySetupLeftPlaymode
rcc::Parser, 105
doBuildPenaltySetupRightPlaymode
rcc::Parser, 106
doBuildPenaltyTakenLeftPlaymode
rcc::Parser, 106
doBuildPenaltyTakenRightPlaymode
rcc::Parser, 107
doBuildPenaltyWinnerLeftPlaymode
rcc::Parser, 107
doBuildPenaltyWinnerRightPlaymode
rcc::Parser, 108
doBuildPlayer
rcc::Parser, 109–111, 113, 114, 116–119, 121, 123, 124, 126, 127
doBuildPlayerAudio
rcc::Parser, 128–130
doBuildPlayerParam
rcc::Parser, 131, 132
doBuildPlayerType
rcc::Parser, 133, 134
doBuildPlayerVisBall
rcc::Parser, 135
doBuildPlayOnPlayMode
rcc::Parser, 108
doBuildRefAudio
rcc::Parser, 136
doBuildRightLocation
rcc::Parser, 136
doBuildRightSide
rcc::Parser, 136
doBuildSaidTooManyAdviceMessagesError
rcc::Parser, 137
doBuildSaidTooManyDefineMessagesError
rcc::Parser, 138
doBuildSaidTooManyFreeformMessagesError
rcc::Parser, 138
doBuildSaidTooManyInfoMessagesError
rcc::Parser, 138
doBuildSaidTooManyMessagesError
rcc::Parser, 139
doBuildSaidTooManyMetaMessagesError
rcc::Parser, 139
doBuildSayMessageTooLongError
rcc::Parser, 140
doBuildSayOK
rcc::Parser, 140
doBuildScore
rcc::Parser, 140
doBuildSelfAudio
rcc::Parser, 141
doBuildSenseBody
rcc::Parser, 142, 144
doBuildServerParam
rcc::Parser, 146, 147
doBuildSubstitution
rcc::Parser, 152
doBuildTackleState
rcc::Parser, 152
doBuildTeamName
rcc::Parser, 153
doBuildTimeOverPlayMode
rcc::Parser, 154
doBuildTooManyMovesError
rcc::Parser, 154
doBuildTopLocation
rcc::Parser, 155
doBuildUnknownCommandError
rcc::Parser, 155
doBuildUnknownError
rcc::Parser, 156
doBuildUnknownOK
rcc::Parser, 156
doBuildUnknownRefAudio

rcc::Parser, 156
doBuildUnknownWarning
rcc::Parser, 157
doBuildUNum
rcc::Parser, 155
doBuildViewMode
rcc::Parser, 157
doBuildViewQualityHigh
rcc::Parser, 158
doBuildViewQualityLow
rcc::Parser, 159
doBuildViewWidthNarrow
rcc::Parser, 159
doBuildViewWidthNormal
rcc::Parser, 160
doBuildViewWidthWide
rcc::Parser, 161
doBuildVisual
rcc::Parser, 162
doMsgParsed
rcc::Parser, 164
doParsedCLang
rcc::Parser, 164

getParam
rcc::Parser, 34

Lexer
rcc::Parser, 18

Param
rcc::Parser, 34
parse
rcc::Parser, 165
Parser
rcc::Parser, 34

rcc, 15
rcc::Parser, 17
∼Parser, 18
doBuildAfterGoalLeftPlayMode, 35
doBuildAfterGoalRightPlayMode, 35
doBuildArmState, 35
doBuildBackPassLeftPlayMode, 37
doBuildBackPassRightPlayMode, 37
doBuildBall, 38–41
doBuildBeforeKickOffPlayMode, 42
doBuildBottomLocation, 43
doBuildCannotChangeGoalieWarning, 43
doBuildCannotSayFreeformWhilePlayonError, 43
doBuildCannotSubWhilePlayOnWarning, 44
doBuildCantReconnect, 44
doBuildCatchBallLeftPlayMode, 45
doBuildCatchBallRightPlayMode, 45
doBuildCatchFaultLeftPlayMode, 46
doBuildCatchFaultRightPlayMode, 46
doBuildCenterLocation, 46
doBuildChangePlayerTypeOK, 47
doBuildClangPlayerVersion, 47
doBuildClangPlayerVersionMsg, 47
doBuildClangVerOK, 48
doBuildCoachAudio, 48
doBuildCoachInit, 48
doBuildCompressionOK, 49
doBuildCompressionWarning, 49
doBuildCornerKickLeftPlayMode, 49
doBuildCornerKickRightPlayMode, 50
doBuildCouldNotParseSayError, 50
doBuildCounts, 51
doBuildDropBallPlayMode, 52
doBuildEyeOK, 52
doBuildFirstHalfOverPlayMode, 53
doBuildFlag, 53, 54, 56, 57
doBuildFlagOffset, 58
doBuildFocusState, 59, 60
doBuildFoulBallOutLeftPlayMode, 61
doBuildFoulBallOutRightPlayMode, 61
doBuildFoulChargeLeftPlayMode, 62

doBuildFoulChargeRightPlay-
Mode, 62
doBuildFoulLeftPlayMode, 63
doBuildFoulMultipleAttacker-
LeftPlayMode, 63
doBuildFoulMultipleAttacker-
RightPlayMode, 64
doBuildFoulPushLeftPlayMode,
64
doBuildFoulPushRightPlay-
Mode, 65
doBuildFoulRightPlayMode, 65
doBuildFreeKickFaultLeftPlay-
Mode, 66
doBuildFreeKickFaultRightPlay-
Mode, 66
doBuildFreeKickLeftPlayMode,
67
doBuildFreeKickRightPlay-
Mode, 67
doBuildFullState, 68
doBuildGlobalBall, 70
doBuildGlobalGoal, 70
doBuildGlobalPlayer, 71–73
doBuildGlobalVisual, 73
doBuildGoal, 74–77
doBuildGoalie, 80, 82
doBuildGoalKickLeftPlayMode,
78
doBuildGoalKickRightPlay-
Mode, 79
doBuildGoalLocation, 79
doBuildGoalRefAudio, 80
doBuildHumanPlayMode, 83
doBuildIllegalCommandError, 83
doBuildIllegalModeError, 84
doBuildIllegalObjectFormError,
84
doBuildIndirectFreeKickLeft-
PlayMode, 84
doBuildIndirectFreeKickRight-
PlayMode, 85
doBuildInit, 85, 86
doBuildInitError, 86
doBuildKickInLeftPlayMode, 87
doBuildKickInRightPlayMode,
87
doBuildKickOffLeftPlayMode,
87
doBuildKickOffRightPlayMode,
88
doBuildLeftLocation, 88
doBuildLeftSide, 89
doBuildLine, 90
doBuildMaxOfThatPlayerType-
OnFieldWarning, 91
doBuildNeutralSide, 92
doBuildNoMoreTeamOr-
PlayerOrGoalieError,
92
doBuildNoSubsLeftWarning, 93
doBuildNoSuchPlayerWarning,
93
doBuildNoSuchTeamOrAlready-
HaveCoachError, 94
doBuildNoTeamFoundWarning,
94
doBuildOffSideLeftPlayMode,
95
doBuildOffsideRightPlayMode,
95
doBuildOppSide, 96
doBuildOurSide, 96
doBuildParam, 96–98
doBuildPausePlayMode, 99
doBuildPenaltyDrawPlaymode,
99
doBuildPenaltyFoulLeftPlay-
mode, 100
doBuildPenaltyFoulRightPlay-
mode, 100
doBuildPenaltyKickLeftPlay-
Mode, 101
doBuildPenaltyKickRightPlay-
Mode, 101
doBuildPenaltyLocation, 101
doBuildPenaltyMissLeftPlay-
mode, 102
doBuildPenaltyMissRightPlay-
mode, 102
doBuildPenaltyOnFieldLeftPlay-
mode, 103
doBuildPenaltyOnFieldRight-
Playmode, 103
doBuildPenaltyReadyLeftPlay-
mode, 104
doBuildPenaltyReadyRightPlay-
mode, 104

doBuildPenaltyScoreLeftPlaymode, 104
doBuildPenaltyScoreRightPlaymode, 105
doBuildPenaltySetupLeftPlaymode, 105
doBuildPenaltySetupRightPlaymode, 106
doBuildPenaltyTakenLeftPlaymode, 106
doBuildPenaltyTakenRightPlaymode, 107
doBuildPenaltyWinnerLeftPlaymode, 107
doBuildPenaltyWinnerRightPlaymode, 108
doBuildPlayer, 109–111, 113, 114, 116–119, 121, 123, 124, 126, 127
doBuildPlayerAudio, 128–130
doBuildPlayerParam, 131, 132
doBuildPlayerType, 133, 134
doBuildPlayerVisBall, 135
doBuildPlayOnPlayMode, 108
doBuildRefAudio, 136
doBuildRightLocation, 136
doBuildRightSide, 136
doBuildSaidTooManyAdviceMessagesError, 137
doBuildSaidTooManyDefineMessagesError, 138
doBuildSaidTooManyFreeformMessagesError, 138
doBuildSaidTooManyInfoMessagesError, 138
doBuildSaidTooManyMessagesError, 139
doBuildSaidTooManyMetaMessagesError, 139
doBuildSayMessageTooLongError, 140
doBuildSayOK, 140
doBuildScore, 140
doBuildSelfAudio, 141
doBuildSenseBody, 142, 144
doBuildServerParam, 146, 147
doBuildSubstitution, 152
doBuildTackleState, 152
doBuildTeamName, 153
doBuildTimeOverPlayMode, 154
doBuildTooManyMovesError, 154
doBuildTopLocation, 155
doBuildUnknownCommandError, 155
doBuildUnknownError, 156
doBuildUnknownOK, 156
doBuildUnknownRefAudio, 156
doBuildUnknownWarning, 157
doBuildUNum, 155
doBuildViewMode, 157
doBuildViewQualityHigh, 158
doBuildViewQualityLow, 159
doBuildViewWidthNarrow, 159
doBuildViewWidthNormal, 160
doBuildViewWidthWide, 161
doBuildVisual, 162
doMsgParsed, 164
doParsedCLang, 164
getParam, 34
Lexer, 18
Param, 34
parse, 165
Parser, 34
RCC_lex, 34
RCCLexer, 34

RCC_lex
    rcc::Parser, 34
RCC_parse
    rccparser.h, 168
RCCLexer
    rcc::Parser, 34
rccparser.h, 167
    RCC_parse, 168