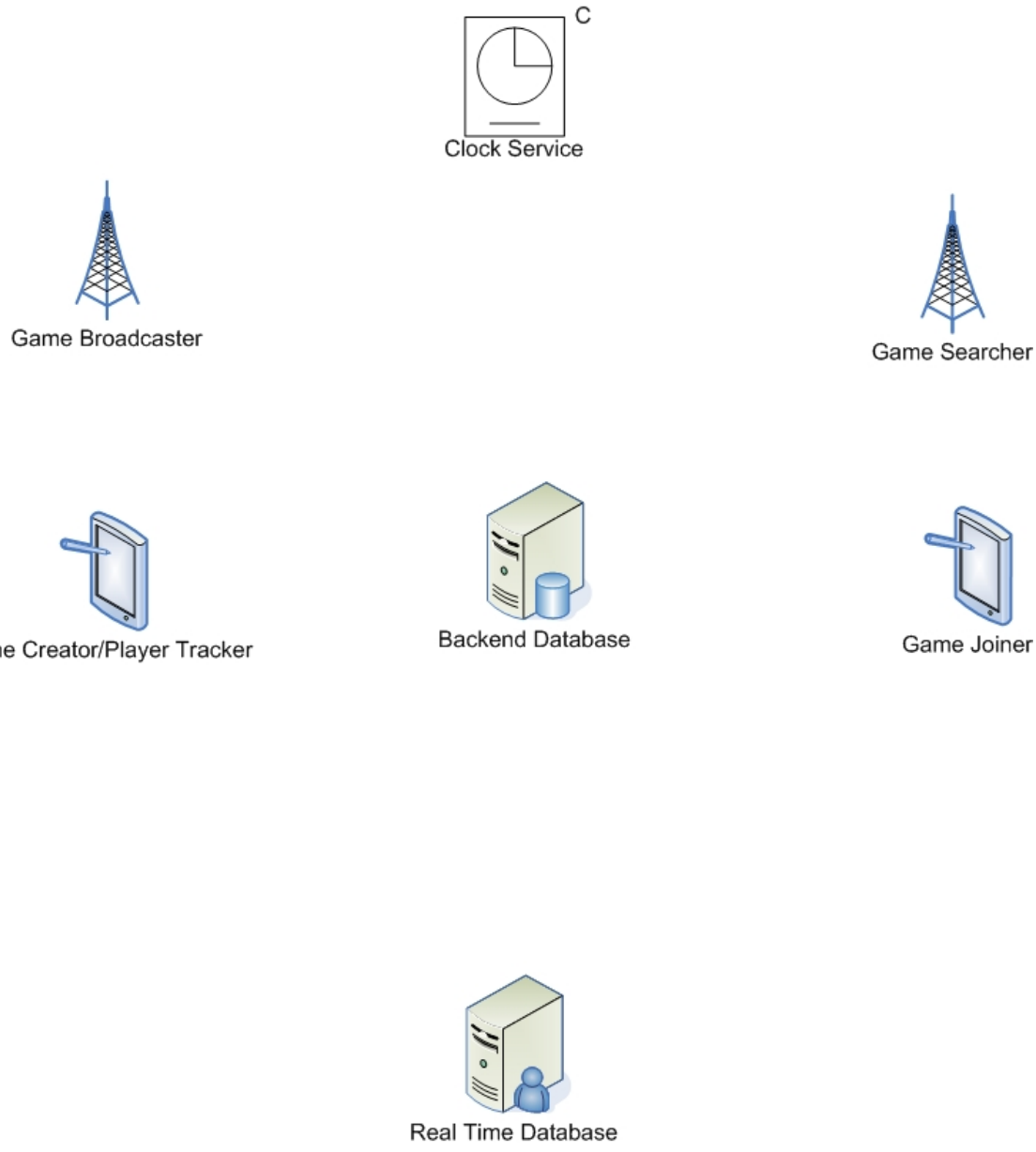
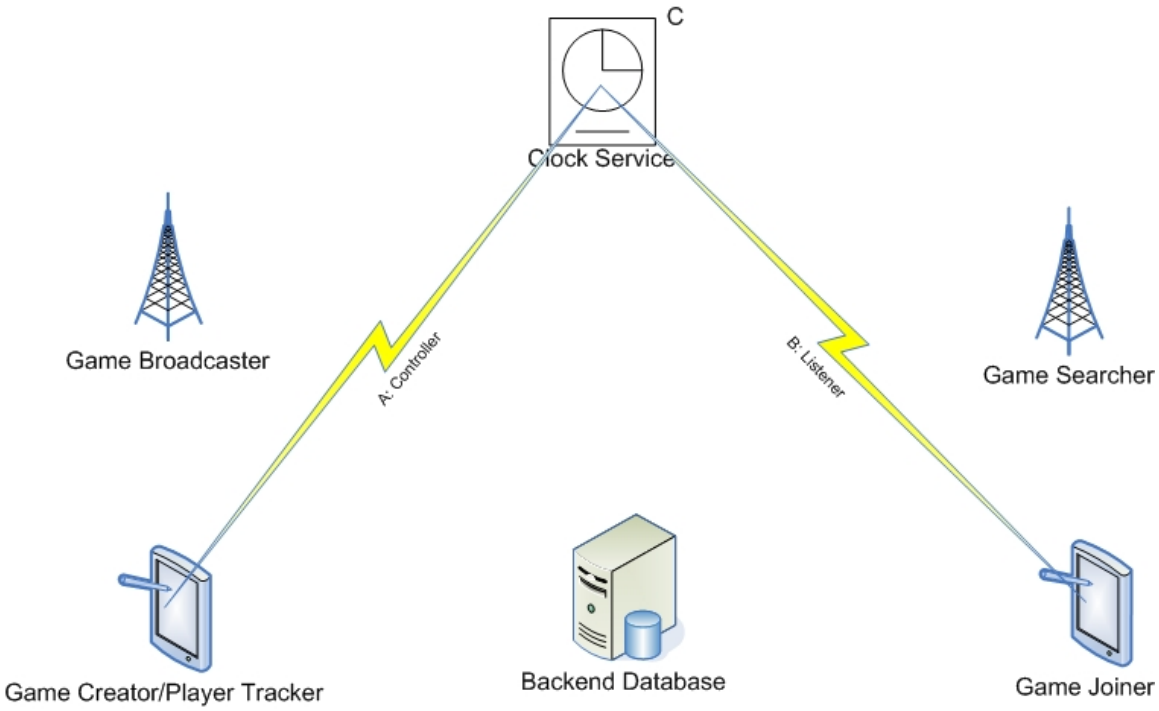
STATe

Architectural Overview



A Reference of the STATe.Resources.dll & System Sevices

Part I: Clock Service



Part I: The Clock Service

Purpose: To keep a standard of game time across all the views that are joined to the current game.

Namespaces: STATe.Clockservice
STATe.Clock

Classes: STATeClock
ClockController
ClockListener
TimeInfo
GameTime

Overview: The clock is designed in a server-client relationship. There is only one type of server, the class STATeClock that runs as a system service. There are two types of clients, ClockController and ClockListener. The function of the listener is to signal events to the client program, and the controller causes those events. Each of these three roles is described below.

Note: All connections are made on Port 400.

STATeClock

The clock service sends a serialized TimeInfo object out every 200 milliseconds to any registered listeners. This TimeInfo object has only one member, a long integer that is the amount of time, in seconds, that has elapsed during the game. The service uses Asynchronous socket connections, and should have no problem handling as much as 15 (this is a very low estimate) current connections.

The service should be restarted for every new game, as each service can only keep track of one game clock at a time. When started, the service is passed a string to store as a pass phrase. If the service ever receives this pass phrase from a client, then that client will be boosted into a privileged mode, where the client is able to start, stop, & drift the clock.

There are special values that can be sent in the TimeInfo object that signal clock control events. If the value is -1, this means the clock has been stopped. If the value is -2, this means the clock has been started.

ClockListner

The ClockListener class is used as a client to receive game clock events, and to perform callbacks on those events. The events that the listener currently understands are:

start:	Triggered whenever the listener receives a -2 from the clock service. (clock started)
stop:	Triggered whenever the listener receives a -1 from the clock service. (clock stopped)
update:	Triggered whenever the listener receives a time from the clock service.

Each of these events can be assigned a callback function using the delegate: void ClockHandeler(ClockListener sender, long gametime).

To begin receiving updates and start triggering events, a ClockListener must connect to the clock service by calling the void Connect(System.Net.IPAddress ip) member, where the address supplied is the IP address the clock service resides on.

Once connected, events will be triggered as they are received. Additionally you may use the function get_game_time() to get the last valid clock value received from the clock service. (See DBConnection for a possible use of this function).

ClockController

The ClockController class is the client that can take control of the clock service. To connect, use the void Connect(System.Net.IPAddress ip, string password) member, where the address supplied is the IP address the clock service resides on, and the password is a the string that was passed to the service on it's start. (This exists so that any program on any machine could take control of the clock, although this feature has not been used yet.)

Once in control of the clock, the following four functions are useful.

Start():	Starts the clock if it's stopped, otherwise does nothing.
Stop():	Stops the clock if it's started, otherwise does nothing.
Increment():	Drifts the clock forward one second (Time elapsed +1).
Decrement():	Drifts the clock backward one second (Time elapsed -1).

Note: A Clock controller has no events, and does not receive information from the clock service. As such, if you require updates and control from/to the clock, then you will need to have both client roles within your program.

GameTime

Since the clock service sends an amount of time elapsed during a game, it is up to the program to decide how to interpret that time. The GameTime class is a versatile solution to this problem, as it allows for different types of game clocks to be used.

When calling the constructor you must pass it a ClockTimingSettings object (from the STATe.Collections.Settings namespace). This object has values for the number of periods in a normal game, the number of seconds in a normal period, and the number of seconds in an overtime period. It is these values that the GameTime class uses to interpret the data received from the clock service.

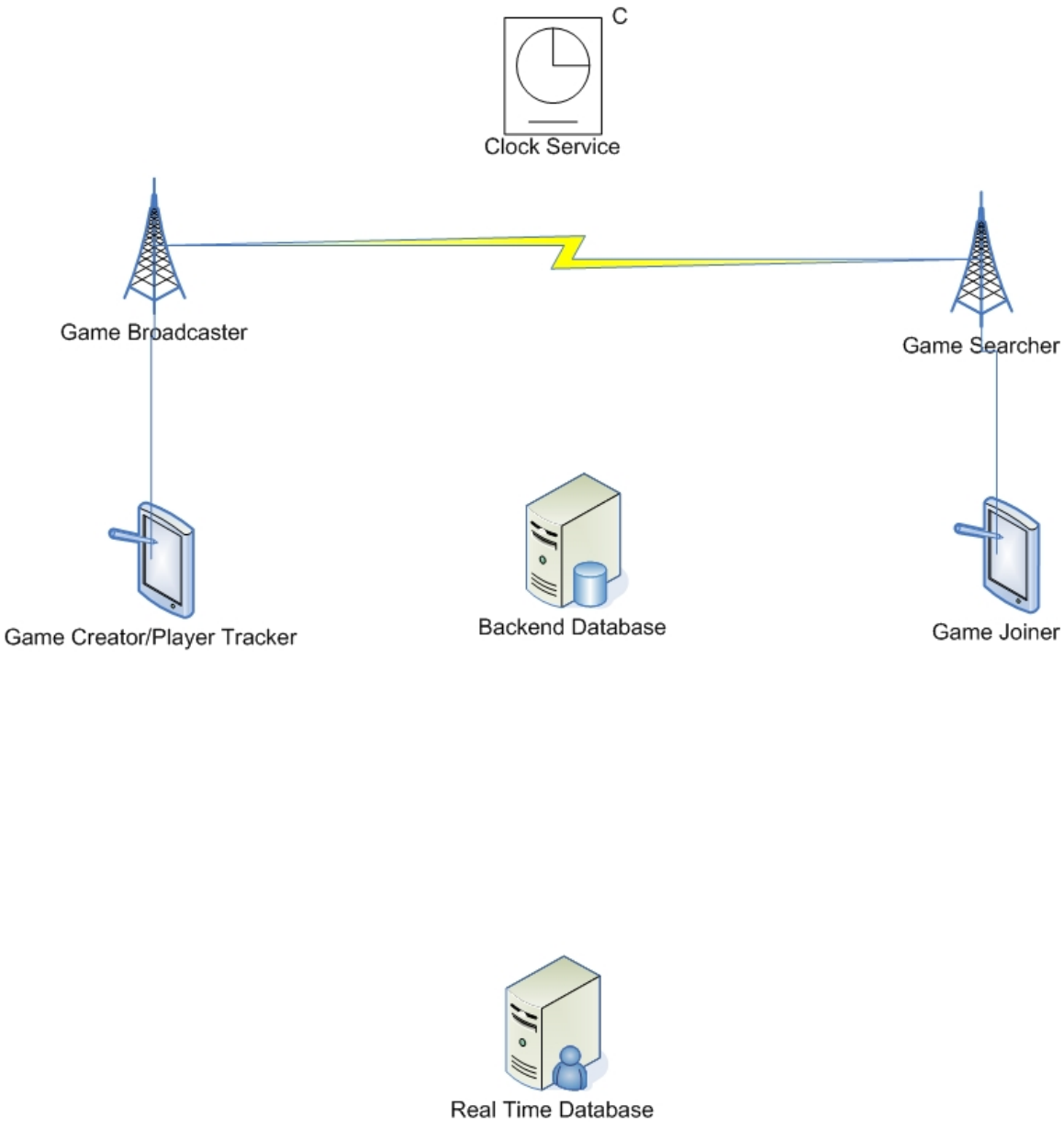
Common usage of this class involves a callback to an update even from a ClockListener object, and passing the gametime to the bool Update(long newTime) member of GameTime. This function will recalculate the interpretation of the game time if the supplied time is different than the previously supplied time, and return true. If the supplied time was not different, then it will return false.

Once update has been called, you may use the following members for game clock information.

int period:	Gets the current period
int minutes:	Gets the current minutes left in the period
int seconds:	Gets the current seconds left in the period
long SecondsInPeriod:	Gets the seconds in a normal period
long SecondsInOver:	Gets the seconds in an overtime period
int PeriodsInGame:	Gets the number of periods in a normal game
string ToString():	Returns a string that has the format MM:SS where MM is the minutes left in the period, and SS is the seconds left in the period

Note: A nice feature to implement in the future, would be an event along the lines of end_of_period, which would occur when the end of a period came about. Programs using this class could assign callbacks to this public event, and handle this situation.

Part II: Game Broadcaster & Searcher



Part II: Game Broadcaster & Searcher

Purpose: To allow computers on the local subnet to join a game and participate in the STATe process.

Namespaces: STATe.Net
STATe.Collections.Settings

Classes: GameSearcher
GameBroadcaster
GameSettings

Overview: GameBroadcaster is a definite misnomer, as it doesn't broadcast at all. Instead it takes the role of server, in a server-client relationship with GameSearcher playing the client role. Their entire purpose is to transmit a serialized GameSettings object from the server to the client. These two roles are defined below.

Note: GameSettings is a collection of all settings that a joiner might need to complete its function. Its scope is too large to cover here, but it is documented in the Namespace overview section of the technical specifications.

GameBroadcaster

The GameBroadcaster runs in a server mode, using asynchronous socket connections to give out a GameSettings object to any client (GameSearcher) that requests it. Once a connection is accepted, it sends the serialized GameSettings object immediately, and then closes the connection.

To use the GameBroadcaster you must first call the Constructor GameBroadcaster(STATe.Collections.Settings.GameSettings info) and pass it the GameSettings object to broadcast. Finally, you must call the BeginBroadcast(IPAddress ip, int port) member, where the address is the IP you wish to broadcast from (in case of a machine with two IP address and the port is the port to listen on. This will start the listening process, to hand out the GameSettings information.

Once you have called BeginBroadcast, the functions EndBroadcast() and RestartBroadcast() might be useful. Their purpose is self evident.

Note: It should be evident that the Game Creator should be the one who uses the game GameBroadcaster class. . This is shown by the connections between the Game Creator and GameBroadcaster parts in the diagram.

GameSearcher

The GameSearcher class acts as a client to the GameBroadcaster. It's main intent is to be ran as a separate thread (thus eliminating any blocking & waiting when attempting to connect to an IP address that does not have a GameBroadcaster.

GameSearcher is somewhat limited in its abilities, as it is tightly bound to the GameListView control class (A control to display GameSettings objects, see the Join Game Wizard Form). Upon successful receive from a GameBroadcaster, the add member of GameListView is called in order to display the found game. **THIS COUPLING SHOULD BE REMOVED IN A FUTURE VERSION.**

The usage of GameSearcher begins with its constructor, which is passed many arguments. Their names and purposes are:

GameListView glv:	A GameListView object to add a successfully retrieved GameSettings object.
IPAddress ip:	The IP Address to search for a GameBroadcaster.
int port:	The Port to search for a GameBroadcaster.
thread_end_signal end:	A special delegate that is called when the thread completes (see below).

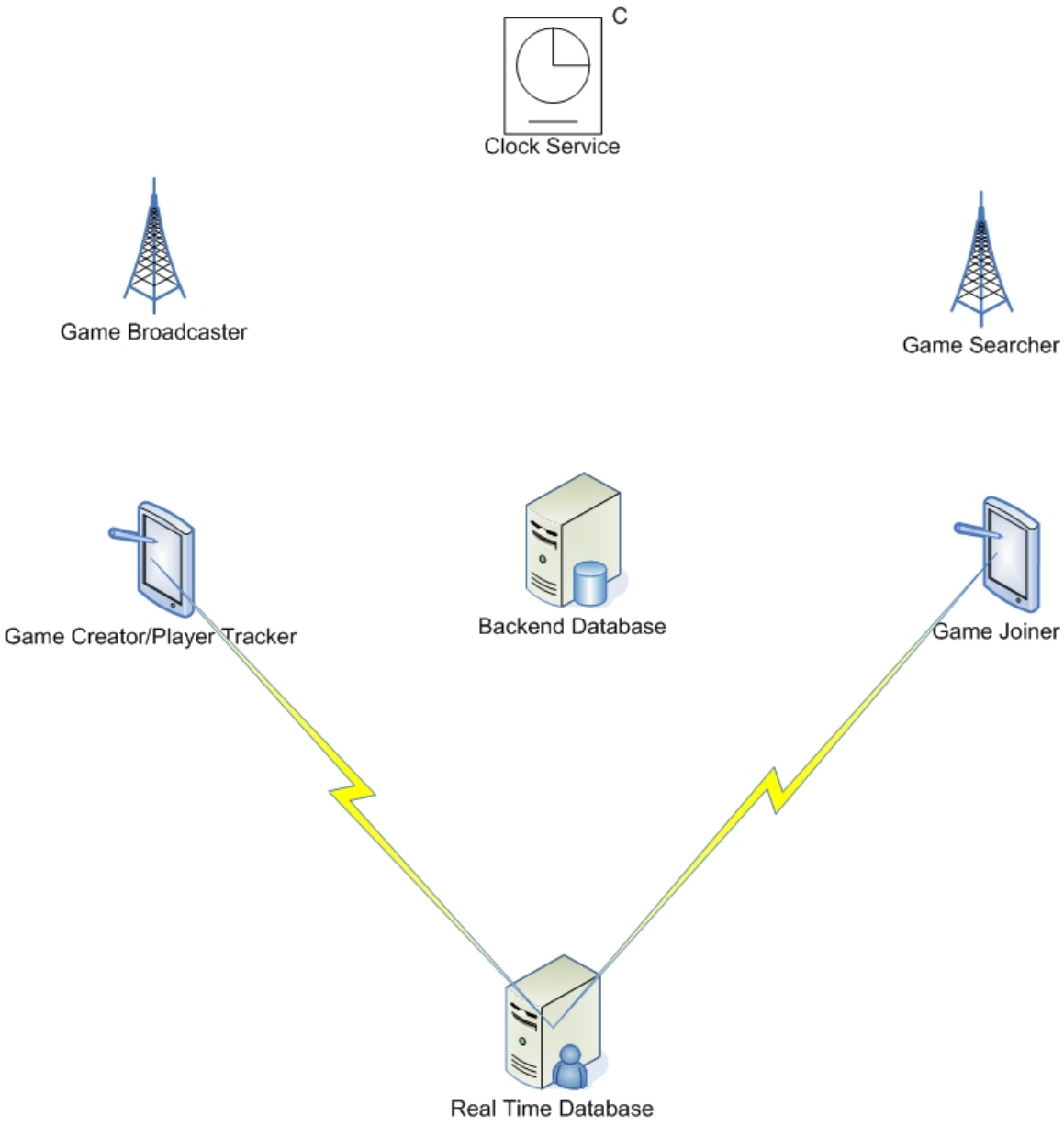
The member function void CheckIPForGame() does all of the work, and should be used as the main entry point of a thread.

The thread_end_signal delegate is of the form void thread_end_signal() and it is simply called when the thread completes, whether or not a GameSettings object was received. It is currently used to update the progress bar on the Join Game Wizard, but any function cal be passed.

Note: Be sure to use the same port in both the GameSearcher and GameBroadcaster.

Note: It should be evident that the Game Joiners should be the ones using the GameSearcher class. This is shown by the connections between the Game Joiner and GameSearcher parts in the diagram.

Part III: The Real Time Database



Part III: The Real Time Database

Purpose: To announce updates as the database receives them to any clients registered for those specific updates.

Namespaces: STATe.DB.RealTime

Classes: STATeRTDB
RTDBConnection
StatUpdate

Overview: The Real Time Database works in a Server-Client relationship, with the STATeRTDB class running as a system service performing the Server Role. The RTDBConnection class performs the client role, and serialized StatUpdate classes are the information they pass between each other. These roles are defined below.

StatUpdate

This class gets passed between the server and clients, and contains all information regarding a specific update. The members of the class and information they contain:

long gameid:	The id of the game, that the update occurs in
long playerid:	The id of the player that the update happens to
stat_type statType:	The type of stat that the update represents
bool increment:	True if the stat was incremented one, false if it was decremented
bool register:	True if this is a register request, false otherwise

The combination of gameid and playerid should be unique to any instance of any game, and therefore the member function string GetHashString() returns a string of the form 'p' + playerid + 'g' + gameid. This string can be used as a hash table key to store a specific player, in a specific game's stat data. See RTDBConnection for a more specific example of how to use this.

Note: byte [] GetByteArray() member function returns the serialized version of a StatUpdate class, which is ready to be sent through a socket.

STATeRTDB

This system service receives serialized StatUpdate objects and then forwards them to any clients registered to the unique gameid and playerid combo contained in that StatUpdate

object. It uses a hash table with the GetHastString() as it's key, and the object associated with key is an ArrayList of sockets that are listening for this type of update.

Much of the STATeRTDB code is similar to the clock service code.

Note: If a client disconnects, all its registrations are removed.

RTDBConnection

This class works as the client side to the Real Time Database (RTDB). You can connect to the server with the member function void Connect(STATe.Collections.Settings.RTDBSettings settings) where settings is a settings object that contains the IP Address of the RTDB server.

Once connected the following functions will begin to function:

RegisterForUpdate(long player_id, long game_id)

Lets the server know that you want to receive all updates for the player with the specified id, for the game with the specified id.

SendUpdate(STATe.DB.RealTime.StatUpdate update)

Send a raw StatUpdate object to the server.

AddStat(long player_id, long game_id, STATe.Collections.stat_type type)

Send an update to the server announcing the player with specified id, in the game with specified id, has gained 1 count of the specified stat.

RemoveStat(long player_id, long game_id, STATe.Collections.stat_type type)

Same as AddStat, but announces that the player has lost 1 count of the specified stat.

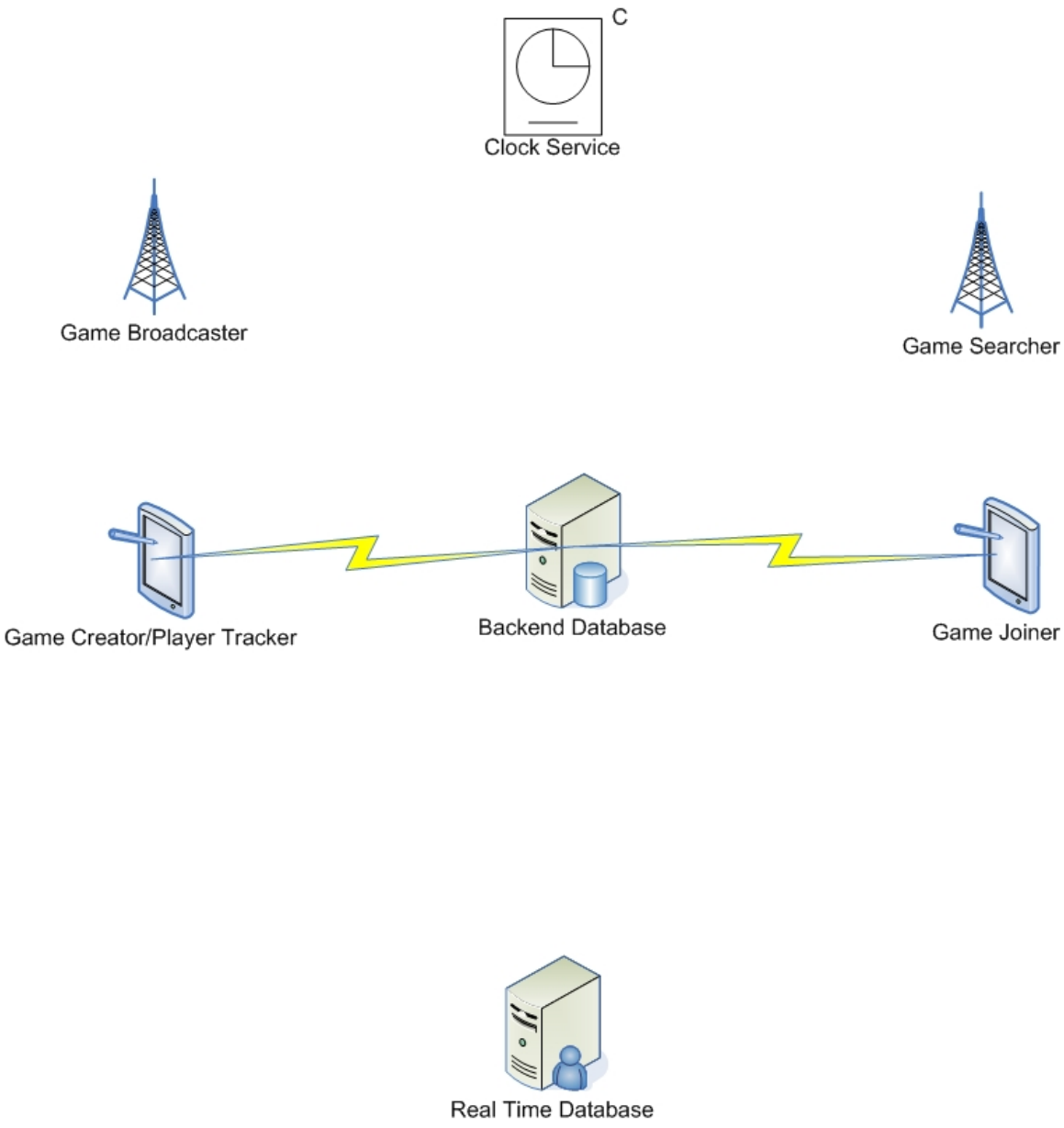
The updates that you are registered can be used by assigning a callback to the update event. The callback delegate is of the form RTDBHandeler(Object sender, STATe.DB.RealTime.StatUpdate data). It is expected that you use the data parameter and its members playerid & gameid to increment or decrement a stat on the corresponding player. It is SUGGESTED that you use a hash table to store your players and that you use the GetHashString() function of StatUpdate as a key for your hash table. This will allow simple code like the following to perform tasks quickly:

Example code of RTDBConnection update event:

```
Player p = playerTable[data.GetHashString()];

if(Player p != null){
	if(data.increment){
		p.stats[data.statType]++;
	}
	else{
		p.stats[data.statType]--;
	}
}
```

Part IV: The Backed Database



Part IV: The Backend Database

Purpose: To allow programs access to all the information about a game, its teams, and its team's players.

Namespaces: STATe.DB

Classes: DBConnection

Overview: DBConnection is the only class that talks directly to the Database. It is extremely large, with many functions that do many different things. To help with the complexity, this section has been broken down into the different types of database queries you can use.

Creating a Connection

The first important note regards the parameter passed to the constructor, DBConnection(TimeFunction tf). The Time Function delegate is a function that returns a long integer that represents the time elapsed in a game. DBConnection gets its own delegate for this info, because many of the functions depend on accurate to the second game time information. However, if you are not going to be doing game time tasks (EG only editing teams, or generating reports from previous data) then it is unnecessary to pass a true TimeFunction delegate. Instead you may pass it a null. Throughout this section, functions will be marked as requiring the time function, or not requiring the time function.

Note: The TimeFunction delegate and the get_game_time() member of ClockListener were designed to be used in tandem, and it is this function that you will often pass as a delegate to the DBConnection.

To create a connection to the database, you must call the void Connect(STATe.Collections.Settings.DBSettings settings) [no time function needed], where the settings object contains information like IP Address, username, & password. This connection will throw an exception on failed logon attempt. Also, this class assumes that a Database named "STATe" is accessible (Select, Insert, Update, Delete privileges) by the specified user.

One of the most useful functions available in the class is void create_tables() [no time function needed]. This function checks the system tables of the SQL server, and creates any tables that STATe needs that are preexisting. It does this by calling a series of sub-level create table function. It is a good idea to call this function on any connection that might be talking to a database that has not been set up yet. No harm will come from calling this function on a database that is already set up.

Game Independent Time Functions

Note: None of these functions require the Game Time delegate function.

void add_team(string team_name)
Adds a team to the database with the specified name.

ArrayList get_teams()
Returns an ArrayList of Team objects, representing all the Teams in the database. The team object's player member is not filled (that requires a separate call to get_players()).

PlayerList get_players(long team_id)
Returns a PlayerList (Strong Typed ArrayList) of Player objects that represent players on the team specified by the id.

void add_player(string last, string first, long team_id, short jersey, int rest_time, int active_time)
Adds a player to the database with the specified values. The last two are optional.

void delete_player(long player_id)
Deletes the player with the id specified from the database. His stats are not deleted.

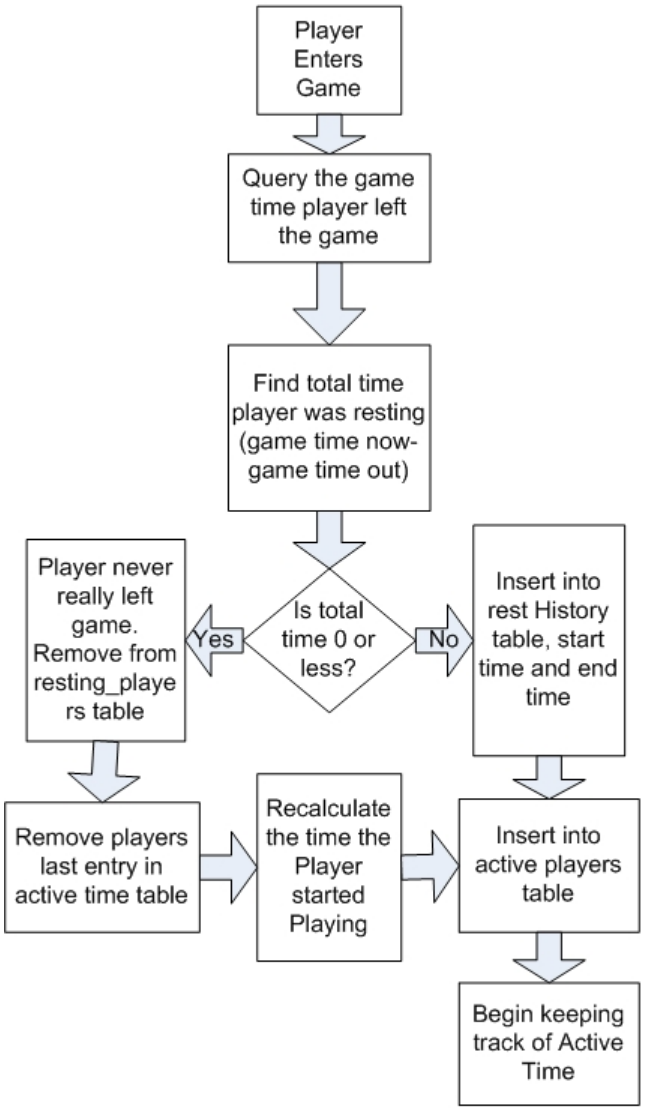
void delete_team(long team_id)
Deletes a team from the database, also deletes all player associated with that team.

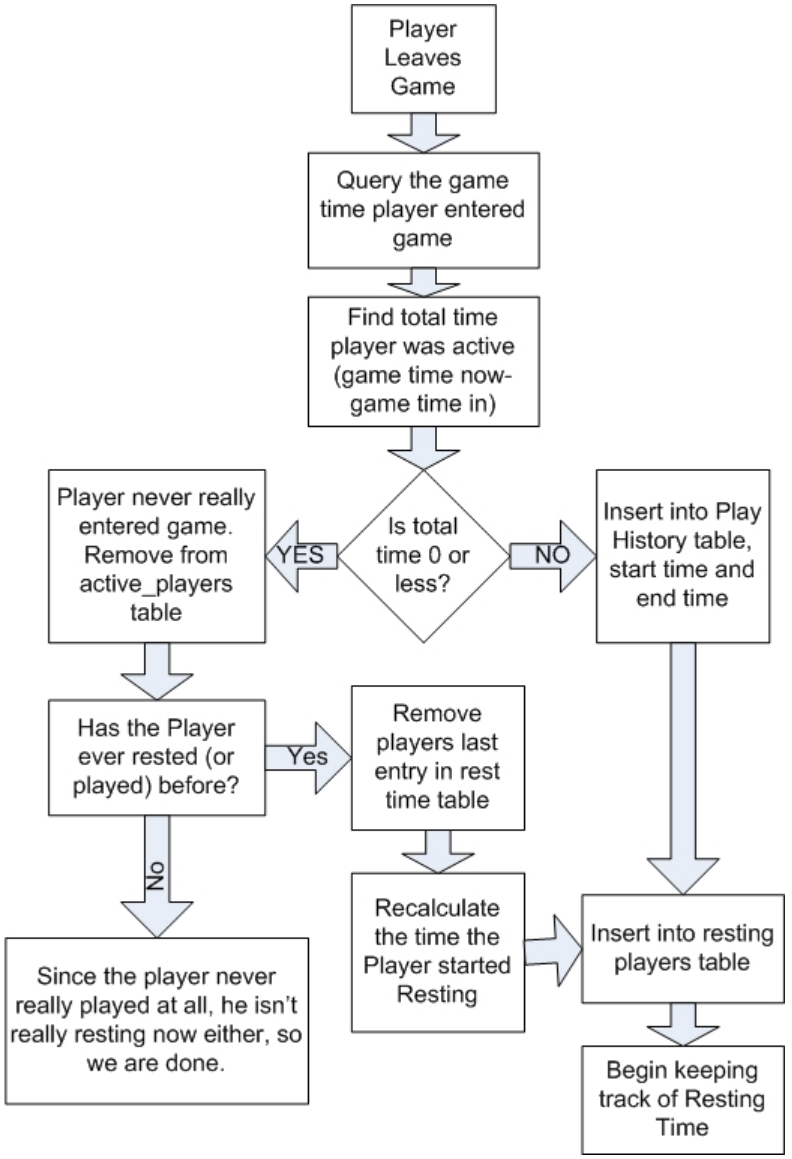
void update_player(Player p)
Updates a player's information in the database.

string getRealTimeString()
Returns a SqlDateTime string that represents the current time when the function was called. Can be used to set specific times that a player checked out of a game. (see check_out_player).

Game Dependent Time Functions

void check_in_player(long player_id, long game_id)
[Requires Game Time function]
Updates the database to reflect that a player is now active. This includes creating a resting log in the resting_times table, and inserting the player into the active_players table. In case a player was accidently checked out (the game time they checked in is the same as the game time they checked out) the function will fix the tables accordingly. See the figure on the next page.


Player Enters Game
Query the game time player left the game
Find total time player was resting (game time now-game time out)
Is total time 0 or less?
Yes
No
Player never really left game. Remove from resting_playe rs table
Insert into rest History table, start time and end time
Remove players last entry in active time table
Recalculate the time the Player started Playing
Insert into active players table
Begin keeping track of Active Time


Player Leaves Game
Query the game time player entered game
Find total time player was active (game time now-game time in)
Is total time 0 or less?
YES
NO
Player never really entered game. Remove from active_players table
Insert into Play History table, start time and end time
Has the Player ever rested (or played) before?
Yes
No
Remove players last entry in rest time table
Since the player never really played at all, he isn't really resting now either, so we are done.
Recalculate the time the Player started Resting
Insert into resting players table
Begin keeping track of Resting Time

void check_out_player(long player_id, long game_id, string real_time)
[Requires Game Time function]
Updates the database to reflect that a player is now inactive. This includes creating an active log in the active_times table, and inserting the player into the resting_players table. It uses the real_time string provided (a Sql DateTime string) as the time the player checked out. If this is not supplied, it uses the current time as the time the player checked out. Like check_in_player, if a player is checked out at the same game time that they checked in, it means they never really played, and the function fixes the database table accordingly. (see figure on previous page)

void add_stat(Player p, Game g, stat_type type)
[Requires Game Time function]
This adds a stat of the specified type to a Player during the specified game. It also records the game time that the stat occurred.

void remove_stat(long player_id, long game_id, stat_type type)
[Doesn't Require Game Time function]
This removes the specified stat from the player during the specified game. It removes the last stat of this type accredited to the player for the game by using the time the stat was accredited. If you try to remove a stat that does not exist (meaning if a player has 0 of that stat) then it will throw an exception.

Team get_home_team(int game_id)
[Doesn't Require Game Time function]
Returns a Team object that represents the home team for the specified game. It does not fill the list of players for the team.

Team get_away_team(int game_id)
[Doesn't Require Game Time function]
Returns a Team object that represents the away team for the specified game. It does not fill the list of players for the team.

long createGame(long home_team_id, long away_team_id, string description)
[Doesn't Require Game Time function]
Creates a game in the database, and returns the id of the game just created. This can then be used to let game joiners know what game to add information to.

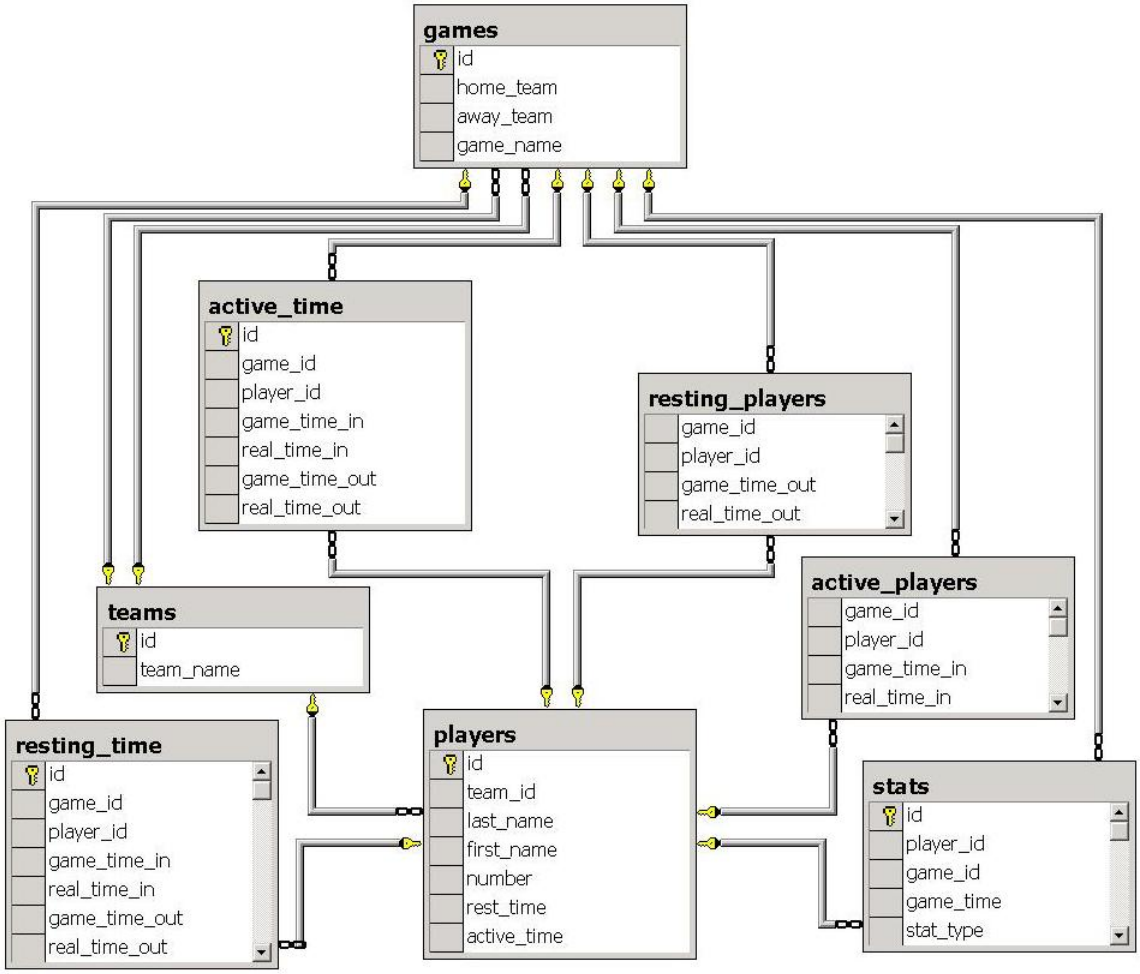
ArrayList getActivePlayers(long game_id, long team_id)
[Doesn't Require Game Time function]
Returns an Array List of PlayerTimeInfo objects (which contain the id of the player, and the game time they entered the game) for a specific team in a specific game. This information can then be used as a basis for each player active time. For example, if a player checked in at game time 5, then his current active time (in seconds) would be the current game time minus 5.

ArrayList getRestingPlayers(long game_id, long team_id))
[Doesn't Require Game Time function]
This returns an ArrayList of PlayerTimeInfo objects (id of the player, and current real time of resting) for a specific team in a specific game. This time number can be use by incrementing it for every second of real time that passes.

PlayerList getPlayersOnTeamWithStats(long team_id, long game_id)
[Doesn't Require Game Time function]
This function works the same as get_players, but it fills the stats member of each player object with the total stats for that game. This is useful when a game joiner joins into an already running game (as opposed to the beginning of the game) because he can get all of the statistical information for that game. It fills the stats member with every stat_type (even if more stat_types are added), but be warned this query is much larger than all others and actually takes a noticeable amount of time (about half a second to a second), so this function should not be called often.

Note: the stat's member of the Player class is dynamic in the sense that it is as large as stat_type.Size value is. In fact, it is simply a unsigned integer array that uses a casting of stat_type to integer as an index in the array. Notice that there is no stat_type.TOTAL_POINTS, but a Player object does have a TOTAL_POINTS member. This is because TOTAL_POINTS is an algorithm based off of number of shots made of certain type. Simply put, the stats member of the Player class keeps track of only the most primitive stats.

Finally, we have included the layout of the database, so that custom functions can be added, and current functions expanded.



Part V: Overview of Current Forms
Player Statistical GUI

Purpose: The purpose of this GUI is to record the statistics during the game.

Namespaces: STATe.forms

Overview: The statistical GUI was designed to keep track of individual and team stats during a game. The statistical GUI is also a clock listener. It receives the game time from this service. Stats can be added at anytime. The reason this was implemented this way is to account for human error. A stat does not have to be recorded at the time it happens. However, the game time the stat was recorded is kept in the database. This could be used in future versions to compute a timeline of the game. This GUI was also designed to be as user friendly as possible. In the fast paced game a user does not have time for many complex features. Simplicity was a major factor in its design.

The constructor for this class takes in a game settings object along with a role. The role is weather this instance of the GUI will track the home or away team. Then the constructor connects to the game clock so it can receive the game time. After the clock is initialized a connection to the database is established and the players in the database are loaded into the GUI. If there are more slots in the GUI then players, the empty slots are disabled so that stats cannot be tracked if there is not a player associated with that slot. If there are more players the slots the first 15 are loaded.

Another useful function in this GUI is the Change Curser function. This function takes in a Boolean variable that symbolizes adding stats or decreasing stats. True is decrement a stat and false is increment a stat. The function changes the way the cursor looks when the minus stat button is pushed. This is to give a visual message to the user that clicking on a stat button will now decrement a stat instead of increment a stat. This allows a user to correct a mistake made in recording statistics throughout the game.

The Populate Line function takes in an index and a player object. The function first checks to see if the player has 4 fouls, if the player does a visual warning is displayed for that player (the foul statistic background color changes to yellow). If a player acquires 5 fouls the color changes to red to symbolize the player fouled out. If a player has less the 4 fouls the background color is white. Then the function populates the stats of that player into the appropriate fields. These fields include the players name, number, shooting statistics, and personal fouls.

The Update Team is also another useful function takes in no arguments. This function updates the stats for the entire team. It checks the number of team fouls and displays a warning if the team is about to enter the single or double bonus foul shots. Then the function computes the stats for each type of shot made or missed and computes totals for the entire team. The updated information is then displayed in the appropriate fields in the GUI.

Sometimes a user might want to distinguish certain stats from other stats. The Change Color function can be used. This function takes in an index to denote which

button was pushed. Then the function changes the background color to highlight specific stats. This enables easy visualization of stats.

The Player Activation function takes in an index into a player list. The function then disables or enables a user to update stats for that player. This can be useful if a player fouls out and no more stats can be recorded for that player.

The functions Personal fouls, change one pointers, change two pointers and change three pointers all take in an index into the slot to denote the stat to update and also a player object to distinguish the player. A slot is made up of arrays of buttons, dropdown boxes, and text boxes. It is a container class developed to hold the stat objects that are recorded as one object. The function then updates the stats in the GUI and the GUI updates the database.

The mouse down functions takes in a sender object. This is the button that was pressed. It then changes the background color to denote that the button was pressed. This gives the user a visual assurance the button pressed event has occurred.

The personal fouls checked function takes in a sender object also. This function updates a player's personal fouls. It finds the index of the player and writes a record to the database for that player. It then updates the team stats to reflect this change.

The player click functions update the shooting statistics. They also take in a sender object that represents the button clicked. The function finds what button was pushed, this denotes the stat that needs to be updated, and indexes into the array. It then finds the player that needs to be updated and writes the record to the database. Again the update team function is called to show the changes.

The selected index changed function allows users to switch the order player appear in the GUI. This takes in a button object also. This is done by a dropdown menu where the player selects the player they wish to appear at the current location. If a player being swapped is already displayed in the GUI the two players swap locations otherwise the new player takes the place of the old player.

The subtract click function again takes in a button object. This function sets a flag that changes the statistic buttons from increasing stats to decreasing stats. This allows a user to undo accidental clicks to player stats.

Player Tracker GUI

Purpose: The purpose of this GUI is to record a player's active and resting time during the game.

Namespaces: STATe.forms

Overview: This GUI was developed to keep track of a player's active and resting time. This GUI also acts as the clock manager. It is responsible for starting and stopping the game time. (See clock service). Five players must be checked in for the clock to start. This ensures a full team is in the game. For ease of use a user can have players who are about to enter or leave the game in pending status (flashing). When the clock is stopped these players will automatically be subbed in or out of the game. This GUI also displays visual warnings about a player's active and resting time. This allows the user to manage player time more efficiently.

The constructor for this GUI takes in game settings and a role. This role again is home team or away team. It then initializes the slots it will use for each player. Then the number of players checked in is set to 0. The constructor then gets the current time from the database to use as a base time. The base time is used to compute resting time of inactive players. The clock service is then configured and the GUI also becomes a listener to this service so it can display the game time. A connection to the database is created and a roster is loaded into the GUI. The real time counter, which is initialized to the base time, is then started. This is used to keep track of a players resting time.

The check in out click function takes an index into the player array. This function checks to see if there currently is a player in this slot. This function then calls the cmdcheckin or cmdcheckout depending on the status of the player when the clock was stopped.

The cmdcheck in and cmdcheck out functions take the same index into the player array. This identifies the player. The first thing the functions check is if the player is already checked in or out. If the player is nothing needs to be done and the function returns otherwise the function checks if the clock is started. If the clock is stared and a player is trying to check in or out the player is put in pending status and the player will be substituted at the next clock stop. If the clock is stopped the and player is checked out the number of players in the game is decremented. The function then checks that there are 5 players checked in. If there are not 5 players checked in the clock start button is disabled until 5 players are checked in. The function then checks that the player actually played and was not just subbed in then out with out actually playing. This was a tricky situation to try to handle. Records for this player do not need to be kept. This allows the coaches to change there minds about subbing players in and out. The next step is to compute the active/rest time depending if the player was checked in or out. The player resting/active labels are then updated to show the current status of the player. This also includes the active and resting time and the arrows that display if a player is checked in or out. The last part of this function calls the warnings function, which displays visual warnings about a player's time.

The cmdinout function takes a button object as input and calls the cmdcheckin or cmdcheckout depending if a player checks in or out. This is the function that allows the cmdcheckin/out function to know which player to check in or out. This is done using a switch statement.

The format time function takes in an int and formats it into a string that represents game time in the form of mm:ss this is used to display the time remaining in the game.

The next function is the issue check in out warnings. This function insures that 5 players are always checked in before the clock can be started.

The load roster function connects to the database and loads the players into the gui.

The check out all function checks out all the players that are currently checked in.

The start stop click function starts or stops the clock by calling the stop or start functions.

The activate warnings function enables or disables visual warnings

The timer tick function takes in a game time and updates the playing time and rest time of each player. This function also updates visual warnings if they are needed.

The clock up and down functions do what they say increment or decrement the clock. This allows the game clock in the program to sync with the actual game clock at the stadium.

The period end function checks all the players out.

The 8-ball function was in Dr Dyksen's original program and we kept it for historic reasons.

Bench View GUI

Purpose: The purpose of this GUI is to display the information collected by the other GUIs.

Namespaces: STATe.forms

Overview: The bench view was designed as a simple way for the coaches to see up to date statistics for their players. There is no user interaction with this GUI. It is strictly to display information. The data is kept in the database and sent to the GUI through a TCP/IP socket. There are visual warnings on the GUI about players playing time and fouls.

The constructor for this GUI also takes in a game settings object and a role. The role again is also to distinguish between home and away team. The constructor then sets up slots to hold active and inactive players. Active players are displayed above inactive players. The GUI then connects in with the clock service to receive the game time. It also connects in with the database to load the current data into the view and also so it is able to receive the updated data as it is recorded.

The update clock function receives the updates from the clock service to display the game time on the GUI.

The start clock function clears out all the players from the GUI slots and recomputes weather the player is active or inactive. This is to make sure that it is displaying the current players in the game as being active and the current-resting players as resting. This also forces the statistics to be updated also.

The timer tick function is used to update the resting time of the players resting. Each time this function is called resting players resting time is recomputed from the base time that was initialized when the started to rest. This is the offset used to compute the current time the player has rested.

The rtdb_update function receives the updated statistics and updates the player's stats in the view. This function uses the enumerated stat types to increment or decrement the player's stats. The function receives a stat update object. Inside this object is the data that needs to be updated along with the player id. The function then gets the id of the player to update out of this object and increments or decrements the stat depending on the weather the object sent in is for an increment or a decrement.

Part VI: The Appendix

What follows, is a more Class level oriented technical specification, as opposed to the previous designer oriented specification. This sections is more like schematics as opposed to actual use, however it gives a good approximation to the complexness of the entire project.

STATe FORMS

Class AddTeam

Public Members

teamName	name of team to add.

Public Member Functions

public AddTeam()

Private Members

Button	cancel_button
Button	ok_button
Label label2	Team Name
TextBox teamname	used to insert new team name

Private Member Functions

void label2_Click(object sender, System.EventArgs e)

ClassBenchGui

Public Member Functions

BenchGui(STATe.Collections.Settings.GameSettings gs, STATe.Collections.roles role)

constructor for bench GUI set up active and in active players connect to database and clock service

Private Members

long myTeamId	Team id
Button b_Exit	exit
Label label13	Active Players
Label label14	3 points
Label label15	2 points
Label label16	1 point
Label label17	personal fouls
Label label18	1 point shooting percent
Label label19	2 point shooting percent
Label label20	3 point shooting percent
Label label21	current time
Label label22	Resting Players
Label label23	total points
Label label31	total time
Label label32	PLAYER STATS------→
GroupBox groupBox1	field labels

// Container components

BenchGUIPlayerPanel panel0

…

BenchGUIPlayerPanel panel12 player stats panel

BenchGUIPlayerPanel activePanel1	
…	
BenchGUIPlayerPanel activePanel5	active players stat panel
BenchGUIPlayerPanel[] myRestingPanels	array of panels for resting players
BenchGUIPlayerPanel[] myActivePanels	array of panels for active players
Timer timer1	local timer
ClockListener myCL	Listener for updates from clock service
GameSettings myGameSettings	game settings
DBConnection dbc	database connection
RTDBConnection rtdb	real time database connection
Hashtable myPlayers	hash of players
Label	teamname
GameClock myGameClock	game clock

Private Member Functions

void Clock_Update(STATe.Clock.ClockListener sender, long gametime)
update game time

void Clock_Start(STATe.Clock.ClockListener sender, long gametime)
start game time go through active and inactive players update appropriately

void timer1_Tick(object sender, System.EventArgs e)
increment resting time for inactive players

void rtdb_update(object sender, STATe.DB.RealTime.StatUpdate data)
update database with new data

Class ClockSettingsView

Public Member Functions

ClockSettingsView(STATe.Collections.Settings.ClockSettings Csettings)
used to see if there are already any clock services running also passwords for the service can be set here

Private Members

Button Button1	ok
Button Button2	cancel
Label label1	clock server
Label label2	confirm
Label label3	password
Combobox Isp	Drop down box with available game clock server ip addresses
Textbox passA	password for game clock server
Textbox passB	confirm password for game clock server
GroupBox groupBox1	server settings
GroupBox groupBox2	game clock settings
Label label4	periods per game
Label label5	time in periods
Label label6	overtime
Label label7	“” between minutes and seconds
Label label8	“” between minutes and seconds
Label label9	clock type
TextBox periodsPerGame	enter the number of periods per game
TextBox MinPerPeriod	the number of minutes per period
TextBox MinPerOver	the number of minutes per overtime
TextBox SecsPerPeriod	the number of seconds per period

TextBox SecsPerOver	the number of seconds per overtime
ComboBox ClockTiming	

Private Member Functions

void button1_Click(object, System.EventArgs)

checks passwords match

void comboBox1_SelectedIndexChanged(object, System.EventArgs)

convert inputs to strings

Class DatabaseSettings

Private Members

button_	ok
Button	button_cancel
Label label1	Database Server
TextBox db_server_address	allows location of server to change
Label label2	tells what server ip resolves to
Label	db_server_ip
Label label3	User Name
TextBox db_user_name	user name for connecting to database
Label label4	Password
Label label5	Confirm
TextBox db_pass_a	password for database
TextBox db_pass_b	confirm password for database
Button button1	set to Local set database server to local computer

Private Member Functions

private void button_ok_Click(object sender, System.EventArgs e)

Checks both passwords match

private void db_server_address_Validated(object sender, System.EventArgs e)

Validates that the database servers ip address can be resolved

private void button1_Click(object sender, System.EventArgs e)

set local computer as database server

Public Member Functions

DatabaseSettings(STATe.Collections.Settings.DBSettings settings)

Creates a form of the current database settings.

Class EditPlayer

Private Members

Button	ok
Button	cancel
Label label1	First Name
Label label2	Last Name
Label label3	Number
Label label4	Rest Time
Label label5	Active Time
TextBox first	players first name
TextBox last	players last name
TextBox number	players number
TextBox activeTime	playing time before red warning appears
TextBox restTime	required rest time before player can return
Label label6	seconds
Label label7	seconds

Protected Member Functions

void populate()

assign values from textboxes to variables

Private Member Functions

void number_TextValidated(object sender, System.EventArgs e)

check that player number is between 0 and 99

void restTime_Validated(object sender, System.EventArgs e)

check that rest time is a positive value

Class GameInfoView

Public Members

GameInfoView(STATe.Collections.Settings.GameSettings info)

Constructor displays current settings about the game
Home team, Away Team, Clock settings, Database info

Private Members

GroupBox groupBox1	Game Specific Info
Label label1	Descripton
Label	game_description
Label label2	Home Team
Label label3	Away Team
Label	game_away_team
Label	game_home_team
Button button1	ok
GroupBox groupBox2	Clock Info
Label label4	Clock IP
GroupBox groupBox3	Database Info
Label label5	Database IP
Label label6	Username
Label	clock_ip
Label	db_ip
Label	db_user
GroupBox groupBox4	Real Time Database Info
Label label7	Database IP
Label	RTDB_ip
Label label8	Description

Label label9	Period
Label label10	Period Length
Label label11	Overtime
Label	Clock_description
Label	clock_period
Label	clock_periodLength
Label	clock_overtime

Class JoinGameWizard

////////**Private Members**

Button	cancel
Button	back
Button	next
Button	finish
Panel panel1	Pick a Role to Use
Button advanced	
Panel advance_panel	
Label label2	
Label label11	
ImageList imageList1	preview of each role
Button get_games	
ComboBox ips	
IContainer components	
private int position	
Panel panel2	
Label label1	
GameListView games	
private int threadcount	number of threads searching subnet
ProgressBar searchprogress	display search progress
Label search_progress_label	
Panel panel3	
Label label3	
private bool first	
Label label4	
Label label5 STATs Input	
RadioButton role_bench_home	choose bench role home team
RadioButton role_stats_home	choose stats role home team
RadioButton role_bench_away	choose bench view away team
RadioButton role_stats_away	choose stats role away team
PictureBox pictureBox1	box to display preview
private bool finished	is wizard finished

Public Member Functions

public JoinGameWizard()

public STATe.Collections.Settings.GameSettings myGameSettings
public STATe.Collections.roles myRole

Private Member Functions

private void search_for_games()

Search subnet for available games

private void end_search()

private void refresh_click(object sender, System.EventArgs e)

Search subnet again for available games

private void advanced_Click(object sender, System.EventArgs e)

Show or hide advanced options

private void populate_ip()

Add valid ips

private void games_VisibleChanged(object sender, System.EventArgs e)

private void games_Click(object sender, System.EventArgs e)

Make sure a game was selected then enable next button

private void next_Click(object sender, System.EventArgs e)

Choose to track home or away team

private bool check_next()

private void back_Click(object sender, System.EventArgs e)

Allows user to back track though menus for setting up a game

private void role_Click(object sender, System.EventArgs e)

Makes sure a role was selected

private void games_DoubleClick(object sender, System.EventArgs e)

private void finish_Click(object sender, System.EventArgs e)

Finish setup of game

Class MainWindow

Public Members

Public Member Functions

public MainWindow()

Constructor calls SetUp() and loadSettings()

public void MenuNewGame_Click(object sender, System.EventArgs e)

Quits current game starts new one

public void MenuJoinGame_Click(object sender, System.EventArgs e)

Join an existing game

Private Members

DBSettings myDBSettings	database settings
ClockSettings myClockSettings	clock settings
RTDBSettings myRTDBSettings	real time database settings
MainMenu mainMenu1	menu bar
MenuItem menuItem1	menu item Database
MenuItem menuItem3	menu item STATe
MenuItem MenuNewGame	new game in menu
MenuItem MenuJoinGame	join game in menu
MenuItem menuItem4	menu item Game
StatusBar statusBar1	status bar for broadcast
StatusBarPanel BroadCastingStatus	status of broadcast
MenuItem menuItem5	
MenuItem menuItem7	menu item settings
MenuItem menuBroadcasting	menu for broadcast options
MenuItem menuDBSettingsA	menu for database settings
MenuItem menuEditTeam	menu for edit teams
MenuItem menuDBSettingsB	menu for database settings
MenuItem menuClockSettings	menu for clock settings
ServiceController ClockServiceController	clock controler
ServiceController RTDBServiceController	real time database controler
MenuItem menuRTDBSettingsA	real time database menu
MenuItem menuRTDBSettingsB	real time database menu
SetUp mySetUp	
StatusBarPanel ClockStatus	clock status
StatusBarPanel RTDBStatus	real time database status

Private Member Functions

private void loadSettings()

Default connect to wonderwoman.cse.msu.edu server for this project)

private void MainWindow_Closing(object sender, System.ComponentModel.CancelEventArgs e)

Close main window

private void menuItem2_Click(object sender, System.EventArgs e)

Opens a new team player editor

private void menuBroadcasting_Click(object sender, System.EventArgs e)

Used to start or stop a broadcast

private void menuDBSettingsA_Click(object sender, System.EventArgs e)

Create a database settings form

private void menuClockSettings_Click(object sender, System.EventArgs e)

Create clock settings form

private void menuRTDBSettingsB_Click(object sender, System.EventArgs e)

Create a real time database settings form

private void Tracker_Closed(object sender, EventArgs e)

exit and shutdown database and clock service also enable new game wizard

Class NewGameWizard

Public Members

public STATe.Collections.roles myRole

role to play player tracker, stat tracker, bench view

public STATe.Collections.Settings.GameSettings GameSettings

information needed to start a game teams, game id, database settings ect.

Public Member Functions

public NewGameWizard(STATe.Collections.Settings.DBSettings dbSettings, STATe.Collections.Settings.ClockSettings.Settings,STATe.Collections.Settings.RTDBSettings rtdb)

Set up a new game

private void next_Click(object sender, System.EventArgs e)

move to next window in the game wizard

private void back_Click(object sender, System.EventArgs e)

move back a window in game wizard

private void update_teams()

Update the teams in the combo box of the game wizard

private void update_players(long teamid, PlayerListView plv)

Update the players listed in the game wizard

private void cancel_Click(object sender, System.EventArgs e)

cancel wizard

private void finish_Click(object sender, System.EventArgs e)

Finish wizard

private void HomeTeamCombo_SelectedIndexChanged(object sender, System.EventArgs e)

If the home team is changed update_players is called

private void AwayTeamCombo_SelectedIndexChanged(object sender, System.EventArgs e)

If the away team is called update_players is called

private bool checkNext()
checks required info is selected before next is enabled

private void panel2_validate(object sender, System.EventArgs e)
enables next

private void advanced_Click(object sender, System.EventArgs e)
hide or show advanced options

private void panel3_Paint(object sender, System.Windows.Forms.PaintEventArgs e)

private void db_settings_Click(object sender, System.EventArgs e)

private void clock_settings_Click(object sender, System.EventArgs e)

private void rtdb_settings_Click(object sender, System.EventArgs e)

Private Members

private RTDBSettings myRTDBSettings	current settings for real time database
private DBSettings myDBSettings	current setting of database
private ClockSettings myClockSettings	settings for clock
private bool finished = false	
private long gameID	game id
Button finish	finish new game wizard
Button next	go to next screen in game wizard
Button back	go back a screen in game wizard
Button cancel	cancel game wizard
Panel panel1	
private int position	position in game wizard (screen num)
private System.Collections.ArrayList panels	
Label label1	
Label label2	
Label label3	
Label label4	
Label label5	
ComboBox AwayTeamCombo	select away team
ComboBox HomeTeamCombo	select home team
Label label6	
Panel panel2	
Label label7	
private DBConnection dbc	database connection info
private PlayerListView awayplayers	away players
private PlayerListView homeplayers	home players
Label label8	
TextBox description	
Button advanced	show/hide advanced options
Panel advancedpanel	advanced options
Panel panel3	
Label label11	
Label label12	
Label label13	
Label label14	
Label description_label	
Label away_label	

Label home_label
Button db_settings — show database settings
Button clock_settings — show clock settings
Label label9
RadioButton roleHome — select home team
RadioButton roleAway — select away team
Button rtdb_settings — show real time database settings

Private Member Functions

Class PlayerStatistics

Public Members

Public Member Functions

Private Members

Private Member Functions

Class PlayerTracker

Public Members

Public Member Functions

Private Members

Private Member Functions

Class RTDBSettingsView

Public Members

Public Member Functions

public STATe.Collections.Settings.RTDBSettings myRTDBSettings
    return current database settings

public RTDBSettingsView(STATe.Collections.Settings.RTDBSettings RTDBsettings)

Private Members

Button button1
Button button2
Label label1
ComboBox ips

Private Member Functions

private void button1_Click(object sender, System.EventArgs e)

check a clock service is selected

Class SetUp

Public Members

Public Member Functions
public SetUp()
Private Members
Button newgame
Button joingame
Private Member Functions
private void joingame_Click(object sender, System.EventArgs e)

join a game

private void newgame_Click(object sender, System.EventArgs e)

create a game

Class TeamPlayerEditor
Public Members

Public Member Functions
public TeamPlayerEditor(STATe.Collections.Settings.DBSettings dbs)
connect to the database and update the teams
Private Members
ComboBox teams	select team
Label label1	
Label label2	
Button addteam	add team button
Button deleteteam	delete team button
Private DBConnection dbc	database connection info
private PlayerListView players	list of players
ContextMenu PlayersMenu	
MenuItem menuItem4	
MenuItem menu_newplayer	
MenuItem menu_editplayer	
MenuItem menu_deleteplayer	

Private Member Functions
private void update_teams()

connects to the database and updates the teams

private void update_players(long teamid)

Connects to the database and updates players on team teamid

private void teams_SelectedIndexChanged(object sender, System.EventArgs e)

Selected team changed update to players on new team

private void addteam_Click(object sender, System.EventArgs e)

Add a new team to the database

private void deleteteam_Click(object sender, System.EventArgs e)

Delete a team from the database

private void PlayersMenu_Popup(object sender, System.EventArgs e)

Enable or disable options depending on what has been selected

private void menu_newplayer_Click(object sender, System.EventArgs e)

Add a new player to database

private void menu_editplayer_Click(object sender, System.EventArgs e)

Edit a player in the database

private void menu_deleteplayer_Click(object sender, System.EventArgs e)

Delete a player in the database

Protected Member Functions

protected override void OnClosing(CancelEventArgs e)

disconnect form database

CLOCK CLASSES

Class ClockConnector

Public Members

public System.Net.IPAddress myIP — IP address of clock service

public int myPort — port of the socket

Public Member Functions

public ClockConnector()

constructor initialize socket to NULL

public bool Connected()

is the socket connected

public void Disconnect()

disconnect the socket

Protected Members

protected int CLOCK_PORT = 400	Port listening on
protected Socket mySocket	TCP/IP Socket to communicate
protected byte [] CLOCKRUNNING	query server is clock running

Class ClockControler

Public Member Functions

public ClockController()

public void Connect(System.Net.IPAddress ip, string password)

connect to clock service

public void Start()

send message to start clock

public void Stop()

send message to stop clock

public void Increment()

send message to increment clock timer game time decreases

public void Decrement()

send message to decrease clock timer game time increases

public delegate void ClockHandeler(ClockListener sender, long gametime)

Private Members

private byte [] CLOCKSTOP	info to sent to clock sevice
private byte [] CLOCKSTART	
private byte [] CLOCKINCREMENT	
private byte [] CLOCKDECREMENT	

Class ClockListener

Public Members

public event ClockHandeler stop stop clock event

public event ClockHandeler start start clock event

public event ClockHandeler update update clock event

Public Member Functions

public ClockListener()

create clock listener set current time to 0 if query before everything set

public void Connect(System.Net.IPAddress ip)

connect to clock service

public void SetupRecieveCallback(Socket sock)

public long get_game_time()

return game time

Protected Member Functions

protected void onRecieve(IAsyncResult ar)

when game time comes in start stop or update the clock

Private Members

private long gametime game time

private byte []m_byBuff buffer for socket

Class GameTime

Public Members

public long SecondsInPeriod number of seconds in period

public long SecondsInOver number of seconds in overtime

public long PeriodsInGame number of periods in a game

public int time current time

public int period current period

public int minutes minutes

public int seconds seconds

Public Member Functions

public bool Update(long newTime)

set the current game time

public GameTime(STATe.Collections.Settings.ClockTimingSettings settings)

create a game time object

public override string ToString()

write a string in mm:ss

Private Members

private long seconds_in_period	seconds in period
private long seconds_in_over	seconds in overtime
private int periods_in_game	periods in game
private int mySeconds	current seconds
private int myPeriod	current period
private int myMinutes	current minutes
private long myTime	current time

STATe COLLECTIONS

Class PlayerTimeInfo

Public Members

public long id

public long time

Public Member Functions

public PlayerTimeInfo(long player_id, long Time)

Class BenchGUIPlayerData

Public Members

public Player player	player
public bool active	is player active
public bool hasPlayed	has player played (helps with player checks in then out without playing)
public long current_time	current time
public string current_time_string	current time as string
public long cumulative_time	time played so far

public string cumulative_time_string	time so far as a sting
public enum roles : int	role bench view player tracker stat tracker for home or away team

Public Member Functions

public BenchGUIPlayerData(Player p, bool Active, long baseTime, long totaltime, STATe.DB.TimeFunction time_now)

constructor for player data in bench gui

public void setActive(long base_time)

player is playing bench view shows active players at top of screen

public void setInActive(long baseTime)

player is no longer playing bench view shows inactive players at bottom of screen

public void increment_base()

Private Members

private bool isActive	is player active
private long base_time	base time (used resting time)
private long total_time	total time
private TimeFunction timeNow	current time

class PlayerStats

Public Members

public uint TWO_MAKE

public uint TOTAL_POINTS	total points
public uint ONE_TAKEN	free throw shot taken
public uint TWO_TAKEN	2 point shot taken
public uint THREE_TAKEN	3 point shot taken
public float THREE_PERCENT	3 point % made
public float TWO_PERCENT	2 point % made
public float ONE_PERCENT	free throw % made
public uint TWO_MISS	2 point miss
public uint THREE_MAKE	3 point make

public uint THREE_MISS	3 point miss
public uint ONE_MAKE	free throw make
public uint ONE_MISS	free throw miss
public uint ONE_POINTS	points off free throws
public uint TWO_POINTS	points off two point shots
public uint THREE_POINTS	points off 3 point shots
public uint FOUL	number of fouls
public uint ASSIST	number of assists(currently not tracked)
public uint STEAL	number of steals (currently not tracked)
public uint OFFENSIVE_REBOUND	offensive rebound (currently not tracked)
public uint DEFENSIVE_REBOUND	defensive rebound(currently not tracked)
public uint BLOCK	block(currently not tracked)
public uint TURNOVER	turn over(currently not tracked)
public uint this[stat_type type]	current stat to use

Public Member Functions

public PlayerStats()

creates a stat table

Protected Members

protected uint [] stat_table	enumerated array of stats

class Player

Public Members

public string lastName	player last name
public string firstName	player first name
public long id	player id (for database)
public long teamId	team id (for database)
public short jeresyNumber	player jersey number
public int restTime	required rest time
public int activeTime	max active time
public PlayerStats stats	player stats

Public Member Functions

public Player(string last, string first, long player_id, long team_id, short jersey, int rest_time, int active_time)

create a player

public override string ToString()

override writing a name as a string “first last”

class PlayerList

Public Members

public int count — number of players in the list

Public Member Functions

public PlayerList()

make array list of players

public void Add(Player p)

add player to list

public void Remove(Player p)

remove player from list

public Player this [int index]

access player out of list

Private Members

private ArrayList myPlayers — list of current players

class PlayerListEnumerator

Public Member Functions

public PlayerListEnumerator(PlayerList coll)

create a collection of players

public bool MoveNext()

move through player list

public void Reset()

go back to beginning of list

object System.Collections.IEnumerator.Current

for iterating through list

Private Members

private int nIndex	location in the collection
private PlayerList collection	the collection of players

class Team

Public Members

public string name	team name
public long id	team id (for database)
public PlayerList players	list of players on team

Public Member Functions

public Team(string team_name, long team_id, PlayerList team_players)

create a team with players

public Team(string team_name, long team_id)

create a team with out players

public override string ToString()

return name of team

class Game

Public Members

public long id	game id
public Team homeTeam	home team
public Team awayTeam	away team
public string Description	description of game (home vs. away)

Public Member Functions

public Game(long game_id, Team home_team, Team away_team, string game_description)

create a game class (not to be confused with a game)

class ClockTimingSettings

Public Members

public string ClockDescription	type of clock college NBA high school ect
public long SecondsPerPeriod	number of seconds per period
public long SecondsPerOver	number of seconds per overtime
public int PeriodsPerGame	number of periods per game

Public Member Functions

public ClockTimingSettings(string description, long SecsInPeriod, long SecsInOver, int PeriodsInGame)

create an object for clock timing settings

public void GetObjectData(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

public override string ToString()

return game description

class ClockSettings

Public Members

public IPAddress ip	ip address of clock
public string password	password for clock
public ClockTimingSettings TimeSettings	time settings for clock (college NBA high school ect)

Public Member Functions

public ClockSettings(System.Net.IPAddress ip_address, string pass, STATe.Collections.Settings.ClockTimingSettings settings)

settings for clock

public void GetObjectData(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

public ClockSettings(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

class DBSettings

Public Members

public string Host	database host name
public string password	database password
public string username	database user name
public System.Net.IPAddress ip	ip address of database

Public Member Functions

public DBSettings(string HostName, string UserName, string Password)

create a database settings object

public void GetObjectData(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

public DBSettings(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

Private Members

private string myHost	database host name
private string myPass	database password
private string myUserName	database user name

class RTDBSettings

Public Members

public System.Net.IPAddress ipAddress	ip address of real time database

Public Member Functions

public RTDBSettings(System.Net.IPAddress ip)

create object of real time database settings

public void GetObjectData(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

public RTDBSettings(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

Private Members

private System.Net.IPAddress myIP	ip address of real time database

class GameSettings

Public Members

public string HomeTeamName	home team name
public string Description	description of game settings
public long HomeTeamID	home team id (for database)
public string AwayTeamName	away team name
public long AwayTeamID	away team id (for database)
public long id	game id (for database)
public roles myRole	role to be played (bench view player/stat tracker)

public DBSettings myDBSettings settings for database
public ClockSettings myClockSettings settings for clock
public RTDBSettings myRTDBSettings settings for real time database

Public Member Functions

public GameSettings(long gameid, string description, string home, long homeid, string away, long awayid, STATe.Collections.roles role, STATe.Collections.Settings.ClockSettings CSettings, STATe.Collections.Settings.RTDBSettings RealTimeSettings, STATe.Collections.Settings.DBSettings DataBaseSettings)

Create object with current game settings desired

Public GameSettings(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext ctxt)

public void GetObjectData(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

class ButtonArray

Public Member Functions

public void AddNewButton(System.Windows.Forms.Button aButton)

add a button

public System.Windows.Forms.Button this [int Index]

return button at index

public ButtonArray(System.Windows.Forms.Form host)

sets the form to the hose

Private Members

private readonly System.Windows.Forms.Form HostForm;

class ComboBoxArray

Public Members

public void AddNewComboBox(System.Windows.Forms.ComboBox aComboBox)

add a combo box

public System.Windows.Forms.ComboBox this [int Index]

return combo box at index

public ComboBoxArray(System.Windows.Forms.Form host)

Private Members

private readonly System.Windows.Forms.Form HostForm;

class CheckBoxArray

Public Members

public System.Windows.Forms.CheckBox this [int Index]
add a check box

public CheckBoxArray(System.Windows.Forms.Form host)

Public Member Functions

public void AddNewCheckBox(System.Windows.Forms.CheckBox aCheckBox)

Private Members

private readonly System.Windows.Forms.Form HostForm;

class LabelArray

Public Members

public System.Windows.Forms.Label this [int Index]

return a label at index

Public Member Functions

public void AddNewButton(System.Windows.Forms.Label aLabel)

add a new button to the list

public LabelArray(System.Windows.Forms.Form host)

Private Members

private readonly System.Windows.Forms.Form HostForm;

class PanelArray

Public Members

public void AddNewPanel(System.Windows.Forms.Panel aPanel)

add a new panel

Public Member Functions

public System.Windows.Forms.Panel this [int Index]

return a panel at index

public PanelArray(System.Windows.Forms.Form host)

Private Members

private readonly System.Windows.Forms.Form HostForm;

class DBConnection

Public Members

public STATe.Collections.PlayerList list of players

Public Member Functions

public bool is_connected()

is the database connection active

public DBConnection(TimeFunction tf)

return time

public void Connect(STATe.Collections.Settings.DBSettings settings)

connect to database

public void check_in_player(long player_id, long game_id)

check player in

public void check_out_player(long player_id, long game_id)

check player out

public void check_out_player(long player_id, long game_id, string real_time)

check out player

public void add_stat(Player p, Game g, stat_type type)

add a stat by player

public void add_stat(long player_id, long game_id, stat_type type)

add a stat by player id

public void remove_stat(long player_id, long game_id, stat_type type)

remove stat by player id

public void remove_stat(Player p, Game g, stat_type type)

remove stat by player

Public void add_team(Team t)

Add a team by a team

public void add_team(string team_name)

add a tean name

public ArrayList get_teams()

get teams

public PlayerList get_players(long team_id)

get players on a team

public void get_players(ref Team t)

get players

public void add_player(string last, string first, long team_id, short jersey)

add player to team by name team and jersey number

public void add_player(string last, string first, long team_id, short jersey, int rest_time, int active_time)

add player to team with specific active and rest times (used for visual warnings)

public void add_player(Player p)

add a player

public void delete_player(Player p)

delete a player by player

public void delete_player(long player_id)

delete a player by player id

public void delete_team(Team t)

delete a team by team

public void delete_team(long team_id)

delete a team by id

public Team get_home_team(int game_id)

get home team by id

public Team get_away_team(int game_id)

get away team by id

public void update_player(Player p)

update player by player

public long createGame(long home_team_id, long away_team_id, string description)

create a createGame object

public ArrayList getActivePlayers(long game_id, long team_id)

get active players

public ArrayList getRestingPlayers(long game_id, long team_id)

get resting players

public string getRealTimeString()

get time off sql server

getPlayersOnTeamWithStats(long team_id, long game_id)

get players with stats out of database (used if stat tracker or bench view joins late)

public void disconnect()

disconnect from DB

Protected Members

protected System.Data.SqlClient.SqlConnection myConnection;	connection to SQL server
protected bool connected;	connected
protected System.Collections.ArrayList myTables	tables in the DB
protected TimeFunction get_time;	time used insert into DB

Private Members

private System.Data.SqlClient.SqlCommand myCommand;	SQL command

class StatUpdate

Public Members

public long gameid;	game id
public long playerid;	player id
public STATe.Collections.stat_type statType;	type of stat
public bool increment;	stat increase or decrease

Public Member Functions

public StatUpdate(long game_id, long player_id, stat_type stat, bool inc)

create stat update object

public bool register;

register with real time database

public void GetObjectData(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context)

serialize so data can be sent

public StatUpdate(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext ctxt)

serialize so data can be sent

public string GetHashString()

create unique hash to use as a key

public byte [] GetByteArray()

public delegate void RTDBHandeler(Object sender, STATe.DB.RealTime.StatUpdate data);

class RTDBConnection

Public Member Functions

public bool Connected()

is the database connected

public void Disconnect()

disconnect from database

public RTDBConnection()

constructor for connection object to database

public void Connect(STATe.Collections.Settings.RTDBSettings settings)

connect to d atabase

public event RTDBHandeler update;

update event triggered

public void RegisterForUpdate(long player_id, long game_id)

register for updates

public void SendUpdate(STATe.DB.RealTime.StatUpdate update)

send updates

public void AddStat(long player_id, long game_id, STATe.Collections.stat_type type)
add a stat

public void RemoveStat(long player_id, long game_id, STATe.Collections.stat_type type)
remove a stat

Protected Members

protected Socket mySocket; socket for sending data

Protected Member Functions

protected void onRecieve(IAsyncResult ar) what to do when data is recieved

Private Members

private byte [] m_byBuff; buffer for socket

private System.Net.IPAddress myIP; database ip
private int myPort; port listening on

Private Member Functions

private void SetupRecieveCallback(Socket sock)

setup callback if data is received and connection is lost

class Slot

Public Members

Public System.Windows.Forms.CheckBoxWarningIsActiveCheckBox;

public System.Windows.Forms.Panel myPanel; panel for stats

public int ActiveTimeMax max active time

public int RestTimeMax rest time needed

public string PlayerName player name

public string PlayerNumber player number

public long PlayerID player id

Public Member Functions

public Slot(System.Windows.Forms.CheckBox ChkWarnings,
System.Windows.Forms.Label CINumber,
System.Windows.Forms.Label CIName,
System.Windows.Forms.Label CONumber,
System.Windows.Forms.Label COName,
System.Windows.Forms.Label RestTime,
System.Windows.Forms.Label RestTimeRemaining,
System.Windows.Forms.Label ActiveTime,
System.Windows.Forms.Label ActiveTimeRemaining,
System.Windows.Forms.Button InOut,
System.Windows.Forms.Panel panel)

Create a slot

public void LoadSlot(STATe.Collections.Player player)
load a player into a slod

public void SetIsResting(bool B)
is a player resting (active players are in top slots)

public void SetCheckedIn(bool B)
set a player as active

public void SetCheckInPending(bool B)
set a player to waiting to be checked in

public void SetPlayedAtLeastOnceThisPeriod(bool B)
player has played this period

public void SetCheckOutPending(bool B)
player is waiting to be checked out

public void SetPlayedWhenClockLastStopped(bool B)
set a player played when the clock stopped

public void SetGameTimeCheckedOut(int T)

record game time player checked out in database

public void SetRealTimeCheckedOut(int T)
record actual time player checked out

public void SetGameTimeCheckedIn(int T)
record game time player checked in

public void SetRealTimeCheckedIn(int T)
record real time player checked in

public System.Windows.Forms.Label CIPlayerNumber()
return player number player checking in

public System.Windows.Forms.Label COPlayerNumber()
return player number player checking out

public System.Windows.Forms.Label CIPlayerName()
return name player checking in

public System.Windows.Forms.Label COPlayerName()
return name player checing out

public System.Windows.Forms.Label RestTime()
return rest time for player

public System.Windows.Forms.Label RestTimeRemaining()
return rest time remaining for player

public System.Windows.Forms.Button ChkInOut()
check players in or out

public bool IsResting()
see if player is resting

public bool CheckedIn()
see if player is checked in

public bool CheckInPending()
is player waiting to be checked in

public bool CheckOutPending()
is a player waiting to be checked out

public bool PlayedAtLeastOnceThisPeriod()
has a player played yet this period

public System.Windows.Forms.Label ActiveTime()
players active time

public System.Windows.Forms.Label ActiveTimeRemaining()
players playing time remaining

public bool PlayedWhenClockLastStopped()
when clock is stopped has the player played helps with player is checked in but never actually plays (sub in and out before clock is started)

public int GameTimeCheckedOut()
game time a player checks out

public int RealTimeCheckedOut()
real time player checked out

public int GameTimeCheckedIn()
game time player checked in

public int RealTimeCheckedIn()
real time player checked in

Private Members

private STATe.Collections.Player myPlayer; player object

private System.Windows.Forms.Label lCIPlayerNumber; player number checking in

private System.Windows.Forms.Label lCOPlayerNumber; player number player checking out
private System.Windows.Forms.Label lCIPlayerName; check in player name
private System.Windows.Forms.Label lCOPlayerName; checkout player name
private System.Windows.Forms.Label lRestTime; resting time
private System.Windows.Forms.Label lRestTimeRemaining; rest time remaining
private System.Windows.Forms.Button cmdChkInOut; check player in/out
private bool blnIsResting; player is resting
private bool blnCheckedIn; player is checked in
private bool blnCheckInPending; player is waiting to be checked in
private bool blnCheckOutPending; player is waiting to be checked out
private bool blnPlayedAtLeastOnceThisPeriod; player played this period
private System.Windows.Forms.Label ActiveTimeLabel; active time played
private System.Windows.Forms.Label ActiveTimeRemainingLabel; remaining playing time
private bool blnPlayedWhenClockLastStopped; played (not sub in sub out)
private int intGameTimeCheckedOut; game time checked out
private int intRealTimeCheckedOut; real time checked out
private int intGameTimeCheckedIn; game time checked in
private int intRealTimeCheckedIn; real time checked in

class GameSearcher

Public Member Functions

public void CheckIPForGame()

public GameSearcher(STATe.Forms.Controls.GameListView glv, IPAddress ip, int port, thread_end_signal end)
search for active games

Private Members

private IPAddress myIP; ip address
private STATe.Forms.Controls.GameListView myGLV; show available games
private int myPort; port for game
private thread_end_signal myEnd; end the thread

class GameBroadcaster

Public Member Functions

public GameBroadcaster(STATe.Collections.Settings.GameSettings info)

send info about games upon requests

public void RestartBroadcast()

refresh the info about the game

public void BeginBroadcast(IPAddress ip, int port)

send the requested information

public void EndBroadcast()

end the connection

Private Members

private byte[] myGameInfo;	buffer
private System.Net.Sockets.Socket listener	listen on socket
private IPAddress myIP;	ip address listen
private int myPort;	port listen

Private Member Functions

private void OnConnectRequest(IAsyncResult ar)

accept requests to join listen for more connections

private void NewConnection(Socket sockClient)

send the game info close the connection

class ipList

Public Members

public int count	number of ips in the list

Public Member Functions

public ipList()	ip list
public IPAddress this[int x]	return specific ip out of list

Private Members

private IPAddress [] myAddresses;	ip address

class ipEnumerator

Public Members

public object Current

Public Member Functions

public ipEnumerator(ipList coll)

iterate through ips

public void Reset()

reset index to beginning

public bool MoveNext()

move to next index

public System.Collections.IEnumerator GetEnumerator()

get enumerator works similar to for each

Private Members

private ipList collection;	collection of ip addresses
private int index;	index into collection