**Multi-player architecture:**

**Initial thoughts:**

Can begin to include code prior to formal release, just don't access/run it, maybe comment out for earlier releases apart from Indy components on form.

Have one host interact with all clients.

Speed & TTClock controlled by host, TTClock time updated client TTClock at intervals

Only host can pause. Clients keep running even if out of focus, build in stoppers for points or routes flashing & try to keep stable when out of focus.

Recommend all open screens on discord so all can see

Have table of continuation links - each 2-way with 'railway1/ID' & 'railway2/ID'

Deal with trains entering at wrong links (can include in actions due as soon as have a time to the exit)

Calculate time to continuation for approaching train for receiving railway for floating window and for actions due - only provide when train has clear route or running to exit with nothing in the way

When train enters at receiving railway ensure receiver not penalised for late entry

Allow for trains to be received out of normal order

All railways to have timetables including all trains received/sent for that railway

Clients update host re all 'next' trains due at exits at intervals with 'time to exit' and 'railway/exitID' values, and host transmits that to relevant clients.

Probably best to use UDP but make sure that if an update missed then next will synchronise

Updates to be relatively frequent - say every 1/4 or 1/2 second if possible.

Allow for users to disconnect deliberately (maybe revert to normal timetable for relevant entries but allow for trains received early)

Allow for inadvertent disconnects - message to user/host/connected users + attempt to reconnect button

Possible to join session part way through? Maybe later

Bjorn is the internet expert so may well be able to help with network problems.

------------------

**Client/Server notes:**

Note that when a server has received data from a client it can send data back to that client using the Binding property in the event handler.

Need to catch connection errors, but just log them, don't stop the program

Binding refers to the local IP & port for the client/server object, becomes current when active

When testing on own computer use different port numbers for client & server or get clash

Setting 'Active' establishes the bindings and starts the server listening thread

Found this in http://www.fredshack.com/docs/indy.html

'**Note**: All Indy exceptions descend from EIdException. By default, Delphi 7 has at least one item in Tools > Debugger Options > Language Exceptions related to Indy, "Indy EIDSilentException Exceptions", and, possibly "Indy EIDSocketError Exceptions"... which do not prevent the IDE to fail when running in the by design exception EIdSocketError "Socket Error # 10004". The reason is that the actual exception is "EIdSocketError", not "Indy EIDSocketError Exceptions". Add the former through the Add button, don't forget to add IdException in the users section, and you won't be stopped by the IDE anymore.' Added EIdSocketError to the ignore list of Language Exceptions and after that TIdUDPServer->ReceiveString() didn't give this error when the program exited. Also added IdException and IdStack (from another search on this error) to the #inlcude list, but IdStack on its own didn't stop the error.

Probably best to use the read event as that seems to be recommended in many posts.

Keep ThreadedEvent false in the server (false by default) - this ensures the Read event is processed in the main thread so no synchronisation issues.

Often hear that there's no need to bind a client port, as the OS will allocate it. That's true for client initiated contact, as then the OS allocates a port and the client port is sent in the packet header (the IP address is sent in a lower layer of the TCP/IP hierarchy). The server then responds to the port that it receives from the client. Most contacts are of this nature. However if a server initiates a contact it *does* need a bound client socket so it knows which port to send the data to.

The client has no 'on receive message' event (like the servers OnUDPRead event), so need to poll it in the main loop. But that's ok because an internal buffer in the OS (in fact there's a stack of buffers for repeated unread messages) holds the received data until it is read by ReceiveString or ReceiveBuffer, so it doesn't matter if the read action follows long after the data were sent (tested this up to 20 seconds). This link provides more information https://stackoverflow.com/questions/12300008/where-is-data-sent-by-udp-stored

When poll the client buffer if there's nothing there the data are blank, no exception is raised, but best to use try... catch... for socket errors and take no action but note the error during development

---------

**Experimentation:**

/why doesn't ReadBuffer work? It's because TIdBytes is a Delphi dynamic array type, which can't be used directly in C++Builder. Use 'DynamicArray<Byte> buff' instead of 'TIdBytes buff', then set the length to something long enough, e.g. 'buff.Length = 100'.

This works, ReadBuffer accepts the DynamicArray<Byte> type in place of TIdBytes. Found this from the discussion at https://stackoverflow.com/questions/8378260/c-builder-tarraysystem-byte-not-compiling but added the Length setting myself to make it work.

++++++++++++++++

Note - these two versions work ok:

60135 is the local server port & 60000 the local client port

1) MPClient->SendBuffer("127.0.0.1", 60135, ToBytes("Greetings from your client"));

ServerReceiveMessageBox->Text = MPServer->ReceiveString(10);

MPServer->SendBuffer("127.0.0.1", 60000, ToBytes("Greetings from your server"));

ClientReceiveMessageBox->Text = MPClient->ReceiveString(10);

2) MPClient->SendBuffer("127.0.0.1", 60135, ToBytes("Greetings from your client"));

DynamicArray<System::Byte> buff;

buff.Length = 100;

MPServer->ReceiveBuffer(buff, 100);

ServerReceiveMessageBox->Text = BytesToString(buff, 0, 99);

MPServer->SendBuffer("127.0.0.1", 60000, ToBytes("Greetings from your server"));

ClientReceiveMessageBox->Text = MPClient->ReceiveString(10);

Also the server can respond to OnUDPRead events as in the UDPConnection project in the MultiPlayer folder.

**------------------**

**SimSig notes:**

SimSig note: Contrary to popular belief, it is **not** necessary to set up port forwarding on your router, or to open up ports on your firewall, if all you are interested in is joining as a client on a multiplayer game

SimSig uses TCP

-----------------

**Info transfer:**

Max safe payload is 508 bytes (576 inc 60 byte IPv4 header + 8 byte packet header

Only the host can start the session, or stop, change speed etc.

Server Out - sends TTClock to each in turn + game speed + paused or not, and stored times to exit for trains relevant to each user (avoid 'Broadcast' as routers often won't send these messages out)

Client in: updates TTClock, updates time to entry info for connected links (if not sent then use the timetable time)

Client out: responds to server with times to exit for relevant trains

Server in: stores the times to exit for this user ready for sending out next time.

The above doesn't require threads as all controlled by server, no probs with simultaneous requests from clients

Server packet: TTClock (4 bytes - secs up to 96 hours), game speed + paused or not (2 bytes max), list of train refs (8 bytes max or 5 bytes min - HC+marker) + connection IDs (4 bytes, H&V) + time to entry 1 byte (1/4mins up to 1 hr) = 4 + 13 \* trains min, 4 \* 10 trains max = 38 trains min, 50 max. Just use 38 trains with full reference.

Client packet: list of trains as above, 508 bytes available so use 38 trains as above.

Need table of values for linked connections, users, times to exit for each user (can have both where both send trains to each other), maintained by the server.

For each user a static table - static unless a user departs or their connection fails:

username; user's local IP + socket; list of linked connection IDs + username connected to

Dynamic table: - updated each time a client sends message

Username; list of train refs + connection ID + time to exit.

BUT - what about in-game messages that stop operation? - these will cause delays as the next TTClock update will move ahead whereas the trains won't have moved. Maybe suppress messages for multiplayer, or just give a very brief image flash at the mouse position different for each message type?

**Starting information:**

/ Need a static table on the host's computer with all the links/railways already established (prepared by the host or by someone else for them).

/ Implement at runtime as a map built with 2 pairs, each with railwayname and exit position (TExitPair), for each end of the connection

std:pair<AnsiString, TExitPair> TCouplingPair

std::map<TCouplingPair, TCouplingPair> CouplingMap

/ Maybe have each coupling listed twice, once for first railway and once for second railway, so can easily search on any railway name, but need to define the comparison routine

With this shouldn't need a static table on each user's computer, though they will need to know which continuations are connected so the timetable can be adjusted during operation - don't need to know what they are connected to as the host looks after that.

Then need dynamic maps (2 for each user - inbound and outbound) during operation, one for info received from server about trains due and one for info to be sent out to other users.

Each of form:

<TExitPair, float, TrainHC> for exit position, service h/c and time to exit

When info received it is used to populate the outbound map, *in fact maybe don't need the inbound map as that info is contained in the data packet, just use that to compile the outbound map data in a form that is needed for the packet (prob don't need any maps at all)*

Maybe have a panel for host with boxes completed by clients as they set up - giving username, railway name, IP address & port number + (say) 5 character identifier. User only sends username & railway, IP & port come over with the packet.

As the information is received the server replies giving the continuation IDs that are connected. When received by the client the server is informed that the IDs have been registered (maybe by allocating a non zero number to the attribute).

When all set up the host starts the game and all clients start in response to the hosts start signal via the server.

/ Maybe a marked exit link where coupled - yes, so know which exits are & aren't coupled

Clients first contact server with times to exit (so server has IP & port) [have ID first in case port changes, but deal with that later]. When server receives datagram it updates its dynamic multimap then sends out the new times to exit to the other railways in turn including self. Maybe server only responds to clients - better that way.

Prerequisites: Clients asked to load railway & timetable and click 'Operate' to enter pre-start mode. Set up any routes (incl autoroutes) required BUT DON'T START!

At that point the client can click 'Send local information' [as soon as that done local timer control is suspended] and complete the boxes [Username (up to 4 characters), Railway name (with or without .rly)]. These boxes are checked before acceptance for railway name same as railway loaded and valid username. The panel includes a red light for 'server not ready' which turns green when it is with label 'Server received information'. When the box information has been checked the client sends the information (say) every 5 seconds, until an 'info received' response is received and an 'OK' button becomes enabled and also the 'Ready to begin' menu item becomes enabled. Along with the response is a list of Exit IDs for the railway, which have their attributes marked with an integer and their appearance changes to indicate 'coupled'. (8 in all & all available for blue name overlay - keep existing name overlay if there is one).

The server first loads the coupling file, the railway and timetable, clicks operate and sets up routes like the clients do, BUT DOESN'T START YET! Then the 'receive client info' menu item becomes enabled and when clicked [here the timer controls are suspended for the time being] it starts listening for information from each client and loads a panel with edit boxes for each username, IP, Port, railway & tick box for when client ready to start - the list of these is available from the coupling file. It also loads its own information. The info is all stored on server and when all the boxes are ticked an 'OK' button is enabled that removes the panel, and the 'begin simulation' option is enabled after any more checks that are needed. When the client info is received a response 'Server received information' is sent to the client along with the exit IDs for that railway.

The host also has a railway, username 'Host' (this name not allowed for clients), and that treated like the others wrt information transfer but internet not needed in that case. The information is stored along with client info during 'receive client info' process.



Useful for server to be able to send messages to players without affecting the tt clock, and for players to message server and each other via server.

NB allow for user cancelling join after click 'send' - won't matter, won't be able to send 'ready to begin'

**Timetable adjustment for coupled exits**

During play any train due to enter at a coupling isn't started at its due time, it is started when the relevant train leaves the coupled exit, which may be before or after the due time. This event is signalled to the server by the other client and the server signals it to the receiving client. Note entered on TTrainController::Operate where this to be added. OK if a later repeat arrives first as the earlier repeat still has NotStarted for its running entry so can arrive later

**New Graphic Names**

New name Equivalent continuation name

CouplingExit1 GL84

CouplingExit2 GL83

CouplingExit3 GL85

CouplingExit4 GL80

CouplingExit6 GL81

CouplingExit7 GL86

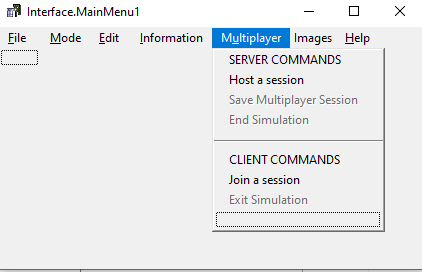
CouplingExit8 GL82

CouplingExit9 GL87

**State when put into abeyance**

Above plans changed quite a bit during coding. Now have two panels with buttons for host and external player rather than having multiple menu items. Still need to sort out the multiplayer modes - done some of this wrt buttons - so can't access features that shouldn't be available until multiplayer exited. Still need to assemble the datagrams and to find out how to deal with multiple sent datagrams (are they each read individually [yes they are] or does the OS look after that?), though may be able to avoid multiple datagrams.

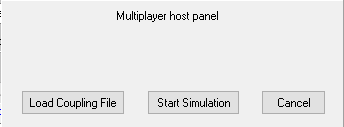
**Current description (16/09/21)**



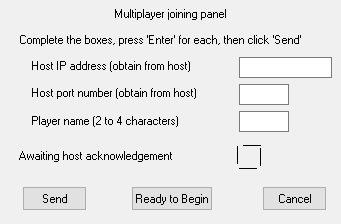
Multiplayer initiated from the main menu as shown.

The menu item will only be available in pre-start mode (i.e. with a railway loaded and any routes set) or when operating in multiplayer mode.

The host panel is shown on clicking 'Host a session', with the 'Start Simulation' button disabled. Clicking the 'Load Coupling File' button starts a dialog (defaulting to the railway.exe directory - 'CurDir'), allowing an appropriately formatted .csv file to be loaded. Loading errors give a message then revert to the same state as before the button is clicked. After a successful load a string grid appears with entries for player name, railway and 'ready to start'.



The first entry is 'Host', others have temporary names 'Player2, Player3 etc. for as many as are needed (determined from the coupling file). These temporary names are changed for user-defined names when contact is made. A label appears in the panel to say 'Awaiting player connections'. When all players are connected the 'Start Simulation' button is enabled. The host is connected as soon as the load is successful.



The client panel is shown on clicking 'Join a session', with the edit panels enabled and just the 'Cancel' button enabled. When each edit box is completed (by pressing 'Enter', the contents are checked and an error message given if appropriate. All being well when all are completed the 'Send' button is enabled, and when clicked sends a datagram to the host containing the username, the railway title, and times to exit for any trains listed in the time to exit map. The 'Send' button is then disabled. After that the 'Awaiting host acknowledgement' label is shown along with a red circle, which changes to green when acknowledged, and the 'Ready to begin' button is enabled. If there is an error with the railway - not listed in coupling file or already taken, the host sends information to indicate this and an appropriate message is given.

**Datagram structures:**

class TRailwayNameUserName

AnsiString RailwayName;

AnsiString UserName;

//-----------------

typedef std::list<TRailwayNameUserName> TRailwayNameUserNameList;

typedef std::pair<TRailwayNameUserName, THVPair> TCouplingPair; //railway/user name + exit HV pair

typedef std::pair<TCouplingPair, TCouplingPair> TCouplingMapEntry;

TRailwayNameUserNameList RailwayNameUserNameList;

//-----------------

typedef std::map<TCouplingPair, TCouplingPair, TStaticMapComp> TStaticMap; //Static map

TStaticMap CouplingMap;

// ----------------

class TServiceInfo

AnsiString ServiceRef;

Word RepeatNumber; //unsigned short integer (2 bytes)

Word TimeToExitSecs; //time to exit in seconds, use 4000 for don't list (> 3600 so won't be)

//-----------------

typedef std::map<THVPair, TServiceInfo> TDynamicMapEntry;

typedef std::map<THVPair, TServiceInfo, TDynamicMapComp> TDynamicMap;

TDynamicMap HostOutBoundMap, HostInBoundMap, ClientOutBoundMap, ClientInBoundMap;

StaticMap (pair): RlyName1, UserName1; HLoc1, VLoc1; -- RlyName2, UserName2; HLoc2, VLoc2

DynamicMapEntry size is 8 bytes for HVPair & 12 bytes ( pad front of service ref with spaces up to 8 chars) for service info = 20 bytes per train, i.e. 25 trains max - allows for username in datagram

DynamicMapOutBound (pair): Exitpair, Service ref, repeat number, time to exit (several)

DynamicMapInBound (pair): Exitpair, Service ref, repeat number, time to enter (several)

NB: Multiplayer menu not enabled until in prestart mode & all buttons disabled while host or client panels visible

ClientInitialContact: sends railwayname, username, receives error message if railway not listed or already taken, else sends list of ExitPairs (may be more than one datagram - keep reading till empty)

ClientRegularContact: sends user (once) + Exitpair, Service ref, repeat number, time to exit (up to 25)

set up the binding when ip & port entered for host

don't enable load coupling file button until above accepted

need a better description of what is happening, separately for host & client

e.g.

Host:

multiplayer menu only enabled in prestart mode

all buttons disabled initially

bring up host panel

all other buttons disabled

have to enter IP & port & have accepted & bound to enable load coupling file button

load coupling file & await client contact

\*\*advise ip & port to clients & say ready for contact - outside game\*\*

------------------------------

**Status at 08/10/21 with comms apparently working and main structures defined:**

Host:

Host loads a pre-start railway

Multiplayer menu enabled

Selects 'Host a session'

Completes boxes with own IP & Port

Clicks 'Load coupling file'

File loads (or error message)

String grid loads all railways & sets host's own railway + ready

Creates new InfoVector & loads RailwayNames and RlyUserNumbers ("Host" as user for own railway)

Message 'Awaiting players'

CouplingFileLoadedAwaitingPlayers = true

1<-Receives player initial message - marker = 1 +username + railwayname

Checks if player has right railway & not already allocated - returns message if not

Host asked if will allow player to join - rejection message if not

Accepts player & updates string grid/InfoVector with all information

1->sends series of: OwnRlyUserNumber + shortHVs then other number + coupled short HVs (10 bytes total per exit/entry), no separators

(50 exit/entries for min datagram)

2<-receives DynMapToHost from player, marker = 2, updates HostCombinedDynamicMap, updates string grid & info vector

2->Sends "Await simulation start"

---> to here

when all players ready 'Start simulation' button enabled

click 'Start simulation'

Host panel & string grid removed

3->'Start' message sent to all players

HostMultiplayerInSession set true

Clock starts & sim begins

4<-receives DynMapToHost from each player

updates InfoVector with player IP & port

updates HostCombinedDynamicMap

Player:

Loads a pre-start railway (allow for multiplayer saved sessions later)

Multiplayer menu enabled

Selects 'Joins a session'

Completes boxes with host IP & Port, and username

Red image

Clicks 'Send'

Yellow image

1->Initial message sent at 5 sec intervals until response received, marker 1 + username + railwayname

Error message if anything wrong or rejected - cancels join request

1<- receive OwnRlyUserNumber + shortHVs then other number + coupled short HVs (10 bytes total per exit/entry), no separators.

Stores own UserNumber, compiles DynMapFromHost and DynMapToHost with blank service info

Ready to begin button enabled

Green image & label changes - 'click ready to begin when ready'

Ready to begin button clicked

2->Sends DynMapToHost, marker = 2

2<-Receives " Await simulation start"

Clears join panel apart from message saying wait for host to start

No message sent out

---> to here

3-<Start message sent from host

Join panel removed

PlayerMultiplayerInSession set true

4->at 5 sec intervals send DynMapToHost updated with latest service info, marker '4' + username art start