DarkRift

Unity API Reference

Introduction

The DarkRift Unity API is your game's gateway to DarkRift. It manages the connection to the server and passes data back and forth to your game.

The API is designed to be very small and very lightweight but still provide all the functionality you need to make it easy to incorporate, this reference will tell you what everything does.

There are 2 parts to the API: DarkRiftConnection which allows object orientated connections to the server and thus supports multiple connections, plus DarkRiftAPI which is a static interface for a single DarkRiftConnection – it's mainly for legacy purposes but it is quite useful when you only require a single connection. Both can be used at the same time.

Since V1.3 DarkRift is able to perform most of the serialisation work in the background to reduce the time DarkRift requires on the main thread. For example when you send data it is queued onto a separate thread then, whilst your application is doing other work, it is encoded and then transmitted and when data is received it will be decoded, passed to the main thread and then your application will be alerted. This is now the default behaviour and means that there is no guarantee that data will arrive in the same order it is sent at the server or other clients, if you require this behaviour you can disable it with DarkRiftAPI.workInBackground = false or DarkRiftConnection.workInBackground = false.

Also since V1.3, you can specify your own distribution modes, which can be later handled by a server plugin, using SendMessage(...). When specifying custom distribution modes a byte is passed to indicate it; the value passed should be greater than 4 as 0 – 4 (inclusive) are used for All, Others, ID, etc and will result in the default behaviour (usually).

Reference

DarkRiftAPI

This is the static way to connect to servers, this can easily be called from any script. This should really be used by default.

Variables

public static DarkRiftConnection connection

Desc: The connection used by DarkRiftAPI.

This is the underlying connection instance that DarkRiftAPI exposes staticly so that you can pass it around as you would with a DarkRiftConnection instance.

public static ushort id

Desc: This is the id of the client as assigned by the server.

The unique id we have been assigned by the server.

public static bool isConnected

Desc: Are we connected to a server?

Determines if the API has connected to a server or not, use Connect() to connect.

public static bool workInBackground

Desc: Should threading be used to send/receive messages?

Setting this true (default) causes DarkRift to do any heavy work in the background so that your application can run smoother. This may, however, cause data to arrive out of order.

Methods

public static bool Connect(string ip)

Desc: Connect to the specified IP.

Tries to connect to the sever at IP on port 4296 (the default) returning true if sucessful or false if not.

public static bool Connect(string ip, int port)

Desc: Connect to the specified I through the specified port.

Tries to connect to the sever at IP on the specified port returning true if successful or false if not.

public static void Receive()

Desc: Processes all the data received and fires the events.

This is the function that processes, decodes and distributes any messages received from the server. Usually this would be called from the Update routine or from the DarkRiftReceiver.cs script but if you need it, it's here.

public static void SendMessageToServer(byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to the server.

Sends a message to the server only with tag, subject and data. This is used to talk to plugins on the server.

public static void SendMessageToID(ushort targetID, byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to a specific ID.

Sends a message to the client with the specified ID with tag, subject and data.

public static void SendMessageToAll(byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to all clients and the server.

Sends a message everyone (including the server) with tag, subject and data.

public static void SendMessageToOthers(byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to all other clients.

Sends a message everyone but the sender (and the server) with tag, subject and data.

public static void SendMessage(DistributionType distributionType, byte tag, ushort subject, object data, ushort id = 0)

Desc: Sends a message to clients/server.

This is the method each of the SendMessageTo... methods call with the corresponding DistributionType to transmit the message. id is an optional parameter for DistributionTye.ID.

public static void SendMessage(byte distributionMode, byte tag, ushort subject, object data, ushort id = 0)

Desc: Sends a message to clients/server.

Similar to the other SendMessage but allows a custom distribution type to be sent, be aware that 0-4 (inclusive) are already used for All, Server, Others etc. id is an option parameter for use in some circumstances.

public static void Disconnect()

Desc: Disconnect from the server.

Disconnects form the specified server, this should always be called in OnApplicationQuit() to stop the server trying to sending data to it.

Delegates

public delegate void DataEvent(byte tag, ushort subject, object data);

Desc:

This is used in the onData event, it is used for basic data transmission.

public delegate void DetailedDataEvent(ushort sender, byte tag, ushort subject, object data);

Desc:

This is used in the onDataDetailed event, it's similar to the DataEvent delegate but also passes the sender's ID

public delegate void ConnectionEvent(ushort id);

Desc:

This is used in the onPlayerDisconnected event to give details about which ID disconnected. It will also be used in the onPlayerConnected event [future].

Events

public static event DataEvent onData;

Desc: Occurs when data is recieved but only gives tag, subject and data.

This event is fired when data is received; it passes the tag, subject and data to the function.

public static event DetailedDataEvent onDataDetailed;

Desc: Occurs when data is recieved but also passes the sender ID.

Like onData this is called when data received but also passes the ID of the sending client.

public static event ConnectionEvent onPlayerDisconnected;

Desc: Occurs when a player has disconnected.

This is called when a player disconnects from the server allowing your game to remove their objects/data/etc; you will be passed their ID.

DarkRiftConnection

This object represents a connection to the server instead of using the DarkRift.DarkRiftAPI static system. By default you should use DarkRift.DarkRiftAPI but if you need more than one connection you should use this.

Variables

public ushort id

Desc: This is the id of the client as assigned by the server.

The unique id we have been assigned by the server.

public bool isConnected

Desc: Are we connected to a server?

Determines if the API has connected to a server or not, use Connect() to connect.

public bool workInBackground

Desc: Should threading be used to send/receive messages?

Setting this true (default) causes DarkRift to do any heavy work in the background so that your application can run smoother. This may, however, cause data to arrive out of order.

Constructors

public DarkRiftConnection()

Desc:

A blank constructor, use Connect() to connect.

public DarkRiftConnection(string ip)

Desc:

A constructor that connects to the server at IP.

public DarkRiftConnection(string ip, int port)

Desc:

A constructor that connects to the server at IP using port port.

Methods

public bool Connect(string ip)

Desc: Connect to the specified IP.

Tries to connect to the sever at IP on port 4296 (the default) returning true if sucessful or false if not.

public bool Connect(string ip, int port)

Desc: Connect to the specified I through the specified port.

Tries to connect to the sever at IP on the specified port returning true if successful or false if not.

public void Receive()

Desc: Receives all data from the server and sends it out.

This is the function that processes, decodes and distributes any messages received from the server. Usually this would be called from the Update routine.

public void SendMessageToServer(byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to the server.

Sends a message to the server only with tag, subject and data. This is used to talk to plugins on the server.

public void SendMessageToID(ushort targetID, byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to a specific ID.

Sends a message to the client with the specified ID with tag, subject and data.

public void SendMessageToAll(byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to all clients and the server.

Sends a message everyone (including the server) with tag, subject and data.

public void SendMessageToOthers(byte tag, ushort subject, object data)

Desc: Sends data, tag and subject to all other clients.

Sends a message everyone but the sender (and the server) with tag, subject and data.

public void SendMessage(DistributionType distributionType, byte tag, ushort subject, object data, ushort id = 0)

Desc: Sends a message to clients/server.

This is the method each of the SendMessageTo... methods call with the corresponding DistributionType to transmit the message. id is an optional parameter for DistributionTye.ID.

public void SendMessage(byte distributionMode, byte tag, ushort subject, object data, ushort id = 0)

Desc: Sends a message to clients/server.

Similar to the other SendMessage but allows a custom distribution type to be sent, be aware that 0-4 (inclusive) are already used for All, Server, Others etc. id is an option parameter for use in some circumstances.

public void Disconnect()

Desc: Disconnect from the server.

Disconnects form the specified server, this should always be called in OnApplicationQuit() to stop the server trying to sending data to it.

Delegates

public delegate void DataEvent(byte tag, ushort subject, object data);

Desc:

This is used in the onData event, it is used for basic data transmission.

public delegate void DetailedDataEvent(ushort sender, byte tag, ushort subject, object data);

Desc:

This is used in the onDataDetailed event, it's similar to the DataEvent delegate but also passes the sender's ID

public delegate void ConnectionEvent(ushort id);

Desc:

This is used in the onPlayerDisconnected event to give details about which ID disconnected. It will also be used in the onPlayerConnected event [future].

Events

public event DataEvent onData;

Desc: Occurs when data is recieved but only gives tag, subject and data.

This event is fired when data is received; it passes the tag, subject and data to the function.

public event DetailedDataEvent onDataDetailed;

Desc: Occurs when data is recieved but also passes the sender ID.

Like onData this is called when data received but also passes the ID of the sending client.

public event ConnectionEvent onPlayerDisconnected;

Desc: Occurs when a player has disconnected.

This is called when a player disconnects from the server allowing your game to remove their objects/data/etc; you will be passed their ID.

DarkRiftReader : BinaryReader

DarkRiftReaders are used to unpack manually serialised objects when they're received. They inherit from the .NET BinaryReader class and so must be disposed of properly or placed in a using statement. The underlying stream of the BinaryReader is a MemoryStream.

For the purposes of my sanity, I'm just going to give a general explanation of all the methods and then list them.

Methods

public virtual byte[] ReadBytes()
public virtual char[] ReadChars()
public virtual bool[] ReadBooleans()
public virtual decimal[] ReadDecimals()
public virtual double[] ReadDoubles()
public virtual short[] ReadInt16s()
public virtual int[] ReadInt32s()
public virtual long[] ReadInt64s()
public virtual sbyte[] ReadSBytes()
public virtual float[] ReadSingles()
public virtual string[] ReadStrings()
public virtual ushort[] ReadUInt16s()
public virtual uint[] ReadUInt32s()
public virtual ulong[] ReadUInt64s()

Reads an array of the specified type off the data.

DarkRiftWriter : BinaryWriter

DarkRiftWriters are used to pack manually serialised objects to be sent across the network. They inherit from the .NET BinaryWriter class and so must be disposed of properly or placed in a using statement. The underlying stream of the BinaryWriter is a MemoryStream.

For the purposes of my sanity, I'm just going to give a general explanation of all the methods and then list them.

Methods

public virtual void Write(bool[] value)
public virtual void Write(decimal[] value)
public virtual void Write(double[] value)
public virtual void Write(short[] value)
public virtual void Write(int[] value)
public virtual void Write(long[] value)
public virtual void Write(sbyte[] value)
public virtual void Write(float[] value)
public virtual void Write(string[] value)
public virtual void Write(ushort[] value)
public virtual void Write(uint[] value)
public virtual void Write(ulong[] value)

Writes an array of the specified type onto the data

DistributionType (enum)

DistributionType is an enum of the commonly used distribution types, though it is possible to use other values.

All = 0

Desc: Send to all connections.

Server = 1

Desc: Only send to the server.

Others = 2

Desc: Send to all connections but the one that sent it.

ID = 3

Desc: Send to a specific ID specified.

Reply = 4

Desc: Send to a specific ID as a reply (internal, probably don't use).

Custom = 5

Desc: Send using a custom mode (internal use, any value higher than 4 counts as custom anyway).

Exceptions

ConnectionFailedException

The server couldn’t be connected to, possibly because of a wrong IP or the server rejected us. See details in the error for more information. Check innerException as it may contain details about the cause.

NotConnectedException

You’re trying to do something that requires you to be connected but you’re not connected.

InvalidDataException

Data can’t be sent because it was invalid. Usually because you are using 255 as a tag (it’s reserved for inside use).