DarkRift

Server Plugin Reference

Introduction

Plugins are key to DarkRift servers, they allow the simple, non-authorative server to stop cheating, stream levels, access databases and much more. Without plugins clients need to do all the work which, on poor hardware, can slow the game and makes ‘hacking’ and ‘modding’ very easy therefore you can write plugins to take some of the load off the clients and make sure games are fair. You can also add new dynamics to your games like Non Player Characters (NPCs) or dynamic maps that are reflected across all the clients.

Plugins are designed to be easy to write and easy to run; they are written in C# and simply built to a .dll meaning they can be shared and used by many of the community. All it takes to use a plugins is to place it in the Plugins folder and then boot the server.

This document is a reference to provide you with all the information you need on every method, variable or event accessible to your plugins.

DarkRift Namespace

Command

The command struct represents an instruction the user can call at runtime, and controls what methods are called on execution.

Constructors

public Command(string name, string description, Action<string[]> callback)

Desc: Initializes a new instance of a command.

This can be used to easily create commands with a name, description and a callback function for when the command is executed (see callback variable).

Variables

public string name;

Desc: The name of the command.

This is the name of the command and thus what the user types to execute this command.

public string description;

Desc: The description of the command.

This is a description of the command for use in the help functions.

public Action<string[]> callback;

Desc: The function called when the command is executed.

This allows you to specify the function that will be executed when the command is called, it must take an array of strings as its only parameter which will contain the parameters the user specified on execution.

ConnectionService

Each instance of this class handles a single connection to a client; it is where all data is processed and handled.

Variables

public ushort id;

Desc: The id of the client that this ConnectionService handles.

The ID of the client this instance represents/handles.

Methods

public bool SendNetworkMessage(NetworkMessage message)

Desc: Encodes and transmits a network event across this connection.

This method will send the NetworkMessage passed to it to this ConnectionService's associated client. It will return true of the data sent successfully.

public bool SendReply(byte tag, ushort subject, object data)

Desc: Sends a reply as the server.

This sends a reply to the ConnectionServices's client as the server (ID 0) with specified data. This basically sends a NetworkMessage with sender 0 and DistributionType Reply.

public void SetData(string pluginName, string fieldName, object data)

Desc: Sets a player data item on this connection.

This allows you to store data regarding this ConnectionService in a central area that any plugin can access and/or change.

public object GetData(string pluginName, string fieldName)

Desc: Get a player data item on this connection.

Allows you to retrieve information about this ConnectionService set using the SetData method.

public void Close()

Desc: Close this connection.

This method safely closes the connection and alerts the client that they have been disconnected.

Delegates

public delegate void PlayerConnectEvent(ConnectionService con);

Desc:

The PlayerConnectEvent is the delegate used in onPlayerConnect. The ConnectionService supplied is that for the new connection.

public delegate void DataEvent(ConnectionService con, ref NetworkMessage data);

Desc:

DataEvent delegates are used in onData and onDataDecoded events. They represent an event which can read and write to the message via the reference passed to them.

public delegate void ReadOnlyEvent(ConnectionService con, NetworkMessage data);

Desc:

ReadOnlyEvent delegates are used in onServerMessage events. They cannot be written to unlike normal DataEvents.

public delegate void PlayerDisconnectEvent(ConnectionService con);

Desc:

PlayerDisconnectEvent delegates are used only for the onPlayerDisconnect event. Data cannot be sent to the client at this point as the streams have been closed, it is purely to let your plugins know the player has disconnected.

public delegate bool DistributorEvent(ConnectionService con, NetworkMessage data);

Desc:

DistributorEvent delegates are used only for the onDistribute event. They are used to override the server's normal distribution system.

Events

public static event PlayerConnectEvent onPlayerConnect;

Desc: Occurs when a player connects to the server.

onPlayerConnect is fired everytime a client connects to the server. See PlayerConnectEvent for parameters and returns.

public static event DataEvent onData;

Desc: Occurs when data is recieved.

onDataRecieve is called every time data is received from a client. At this point the data is not decoded and you only have the metadata about the NetworkMessage populated but you can edit the data that is there via the reference passed. See DataEvent for parameters and returns.

public static event DataEvent onDataDecoded;

Desc: Occurs when data is recieved and decoded.

onDataDecoded is called when data is has been decoded and can be used. This is similar to the onData event but the data will be decoded for you to use first, everything is editable via the reference passed to you. See DataEvent for parameters and returns.

public static event PlayerDisconnectEvent onPlayerDisconnect;

Desc: Occurs when a client disconnects.

onPlayerDisconnect is called when a client sends the disconnect message to the server, at this point the client will have already gone so you can't send data but you should free up any resources they had on the server here. See PlayerDisconnectEvent for parameters and returns.

public static event ReadOnlyDataEvent onServerMessage;

Desc: Occurs when the server is sent a message, with the data decoded.

onServerMessage is called when the server is sent a message or when a message is sent to all. These messages include system messages like the disconnect message on tag 255. See ReadOnlyDataEvent for parameters and returns.

public static event DistributorEvent onDistribute;

Desc: Occurs when distributing data.

onDistribute is used to override the builtin distribution methods by returning true, if the return value is false the server will distribute the data. There should never be more than one plugin connected to this event as this will cause some really bad effects like data being sent twice etc. See DistributorEvent for parameters and returns.

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.

DarkRiftServer

The DarkRiftServer Class is the base class for the server and holds the connections as well as a few generics.

Variables

public static ConfigReader settings;

Desc: The config reader for the main settings.cnf file (Read only).

This is a reference to the ConfigReader used to read the basic settings of the server as set in the settings.cnf file. You can use this to decide whether to output debug data etc but you should use your own .cnf file for each plugin if you need to add extra (plugin independent) settings.

public const int CONNECTION_LIMIT;

Desc: The maximum number of concurrent connections this server is licensed for.

This is the maximum number of connections that the server is allowed to have.

public static Database database{ get; internal set; }

Desc: The database.

A reference to the database plugin currently active, if there is one available.

Methods

public static ConnectionService GetConnectionServiceByID(ushort id)

Desc: Gets the connection service for the specified client.

Returns the connection service connected to the client of ID id.

public static ConnectionService[] GetAllConnections()

Desc: Gets a list of all connections to this server.

Returns a list of all the ConnectionsServices this server has.

public static ushort GetConnectionLimit()

Desc: Gets the maximum number of connections allowed.

This returns the lower of CONNECTION_LIMIT and the value defined in settings.cnf. This should be used rather than just CONNECTION_LIMIT.

public static int GetNumberOfConnections()

Desc: Gets the number of connections to this server.

Returns the number of clients connected to this server.

public static void Close(bool quit)

Desc: Safely shutdown the server.

Closes the server correctly to ensure resources are disposed of correctly and connections are closed. If quit is set to true the program will exit.

Delegates:

public delegate void ServerCloseEvent();

Desc:

This is the delegate for the onServerCloseEvent.

Events:

public static event ServerCloseEvent onServerClose

Desc: Occurs when the server is closing.

This occurs when the server has been told to close, you should close any files etc here.

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

Interface

A generic wrapper for the GUI or console to give a standard way of inputting/outputting data. Use this instead of Console.WriteLine etc.

Methods

public static void LogTrace(string message)

Desc: Logs a message to the log file only.

Writes a message to file only with no output on the console.

public static void Log(string message)

Desc: Logs a message to the console.

This will log a standard message to the console/GUI/Logs.

public static void LogWarning(string message)

Desc: Logs a warning to the console.

This will log a warning to the console/GUI/Logs. Warnings should be used when the server has detected a problem but has resolved it like defaulting to a value if specified value is invalid.

public static void LogError(string message)

Desc: Logs an error to the console.

This will log an error to the console/GUI/Logs. Errors should be used when a component failed but operation has not be interrupted permanently.

public static void LogFatal(string message)

Desc: Logs a fatal error to the console.

This will log a fatal error to the console/GUI/Logs. Fatal errors should be used when the server has encountered an error and cannot continue running.

PerformanceMonitor

The performance monitor provides your plugins with stats about the server's performance. Currently there's not much information available here but more will be added with updates.

Methods

public static void ResetCounters

Desc: Resets the counters to 0.

This resets all the counters in the performance monitor.

Variables

public static long totalExecutionCounts

Desc: Gets the total number messages that have been processed.

This holds the number of messages that the server has received and processed.

public static double totalExecutionTime

Desc: Gets the total time the server has been processing data.

Returns the total time the server has spent processing data sent to it (including in events) in milliseconds.

public static double averageExecutionTime

Desc: Gets the average time it takes to process a single message.

Returns the totalExecutionTime divided by the totalExecutionCounts to get an average of how long it takes to execute a message in milliseconds.

public static long totalConnectionReadCounts

Desc: Gets the total number of connection reads.

Returns the total number of times data has been taken off of a socket onto the internal buffers.

public static double totalConnectionReadTime

Desc: Gets the total time the server has been reading data.

Returns the total time the server has been reading data from the sockets to the internal buffers.

public static double averageConnectionReadTime

Desc: Gets the average time it takes to read a single message.

Returns the total time the server has been reading data from the sockets divided by the number of times it has read data.

Plugin

This is the base class for plugins and identifies a .dll as a plugin. They must implement all the abstract fields below.

Variables

public abstract string name{get;}

Desc: The name of the plugin.

This should be set to the name of your plugin when inheriting from Plugin.

public abstract string version{get;}

Desc: The version of the plugin.

This should be set to the version of your plugin in dot format eg. "1.2.10".

public abstract string[] commands{get;}

Desc: A list of commands this plugin uses.

This should be a list of commands your plugins uses so that the GUI can create options etc.

public abstract string author{get;}

Desc: The author of the plugin.

This should be set tothe name of whoever wrote the plugin.

public abstract string supportEmail{get;}

Desc: An email at which users can get support.

This should be a valid email address that users can get plugins related help at.

Methods

public virtual void Update()

Desc: Any overriding member will be called at a user set repetition rate.

If you override the Update method your method will be called repeatedly at a rate defined in settings.cnf. This allows you to move heavy processing to this method which can reduce the time it takes to process data.

protected bool IsInstalled()

Desc: Determines whether the plugin is installed.

Determines whether this plugin is 'installed', currently whether this plugin already has a subdirectory within the plugins folder.

protected void InstallSubdirectory(Dictionary<string, byte[]> fileContents)

Desc: Creates the plugin's subdirectory and writes the default files to it.

Creates a new directory and unpacks the files specified in each KeyValuePair of the dictionary where key is the filename and extension and the value is a byte array of file contents.

public string GetSubdirectory()

Desc: Gets the address of the plugin's subdirectory.

Returns the path to your plugin’s subdirectory.

PluginManager

The plugin manager is responsible for loading plugins and managing them.

Variables

public static Dictionary<string,Plugin> *plugins* {get;}

Desc: A list of plugins which you can access by name to communicate with.

The references to the plugins loaded allowing you to communicate between plugins. The dictionary is accessed by the exact name of the plugin (as specified inside the plugin not the .dll name).

DarkRift.ConfigTools Namespace

ConfigReader

Config readers allow you to access .cnf config files easily, they allow you to access them using indexes to make life easier.

Constructors

public ConfigReader(string filename)

Desc: Loads a config file for access.

This loads a config file from the specified path relative to the server executable's path, if the file isn't there it'll throw a FileNotFoundException.

Methods

public string this[string key]

Desc: Gets the value of the config option with the specified key.

This, in a similar way to a dictionary does, accesses the value of the specified key. If the key doesn't exist it'll return null.

public bool IsTrue(string key)

Desc: Does the specified key have the value "True"?

Checks to see if the specified key has the value of "True", if it has anything else (excluding the wrong capitalisation) or doesn't exist it will return false.

public void Load(string filename)

Desc: Load the specified filename.

This allows you to reload a configuration file or change the file the reader refers to.

Variables

public bool loaded

Desc: True if the reader was able to load the file.

This is true if the reader successfully loaded the keys and values to memory.

DarkRift.Storage Namespace

Database

The Database class is a type of Plugin used for communicating with an external database. There should only be one plugin of this type which can be accessed through DarkRiftServer.database.

Methods

public abstract string name{get;}

Desc: The name of the plugin.

This should be set to the name of your plugin when inheriting from Plugin.

public abstract string version{get;}

Desc: The version of the plugin.

This should be set to the version of your plugin in dot format eg. “1.2.10”.

public abstract string[] commands{get;}

Desc: A list of commands this plugin uses.

This should be a list of commands your plugins uses so that the GUI can create options etc.

public abstract string author{get;}

Desc: The author of the plugin.

This should be set tothe name of whoever wrote the plugin.

public abstract string supportEmail{get;}

Desc: An email at which users can get support.

This should be a valid email address that users can get plugins related help at.

public abstract string databaseName{get;}

Desc: Gets the name of the database this plugin operates with.

This is the name of the database this plugin works alongside.

public abstract DatabaseRow[] ExecuteQuery (string query, params QueryParameter parameters);

Desc: Executes a query on the database returning an array of rows.

Executes a query on the database which returns an array of database rows (ie a SELECT query).

public abstract object ExecuteScalar (string query, params QueryParameter parameters);

Desc: Executes a query on the database with a scalar return.

Executes a query on the database that returns only a single piece of data.

public abstract void ExecuteNonQuery (string query, params QueryParameter parameters);

Desc: Execute the specified query on the database.

Executes a query on the database where nothing is returned.

public virtual string EscapeString (string s)

Desc: Removes any characters that could allow SQL injection.

Escapes a string and returns it.

public abstract void Dispose();

Desc: Releases all resource used by the Database object.

This should be implemented so that the server connection is closed correctly and not left open and using memory.

public virtual void Update()

Desc: Any overriding member will be called at a user set repetition rate.

If you override the Update method your method will be called repeatedly at a rate defined in settings.cnf. This allows you to move heavy processing to this method which can reduce the time it takes to process data.

protected bool IsInstalled()

Desc: Determines whether the plugin is installed.

Determines whether this plugin is 'installed', currently whether this plugin already has a subdirectory within the plugins folder.

protected void InstallSubdirectory(Dictionary<string, byte[]> fileContents)

Desc: Creates the plugin's subdirectory and writes the default files to it.

Creates a new directory and unpacks the files specified in each KeyValuePair of the dictionary where key is the filename and extension and the value is a byte array of file contents.

public string GetSubdirectory()

Desc: Gets the address of the plugin's subdirectory.

Returns the path to your plugin’s subdirectory.

DatabaseException

This is a wrapper exception for any database errors so that you can easily catch them. Inner exception may contain the source error depending on the database connector used.

DatabaseRow

The DatabaseRow class is used to return data from a database query and marks a single row of a table. It is simply a Dictionary<string, object> and therefore you should refer to the MSDN Dictionary for this class. Note: some methods/constructors/variables may be missing from this implementation of Dictionary.

QueryParameter

This is used to pass parameters into your SQL queries to avoid SQL injection.

Variables

public string name;

Desc: The name of the parameter.

The name of the parameter in the query.

public object obj;

Desc: The object for the parameter.

What to place in the query.

Constructors

public QueryParameter(string name, object obj)

Desc:

Creates aparamater with the given details.