

# Bridgemate Data Connector developer's guide

## Table of contents

---

Revision history .....	3
General.....	4
Data exchange.....	6
Named Pipes.....	6
Overview of communication .....	8
Data Transfer Objects.....	17
Overview .....	20
InitDTO .....	21
ContinueDTO .....	23
SessionDTO .....	24
ScoringGroupDTO .....	26
SectionDTO .....	28
TableDTO .....	30
RoundDTO.....	31
ResultDTO .....	33
PlayerDataDTO .....	36
ParticipationDTO .....	37
Bridgemate2SettingsDTO.....	40
Bridgemate3SettingsDTO.....	45
BCSManagementRequestDTO .....	49
BCSManagementResponseDTO .....	50
HandrecordDTO .....	51
Procedures.....	53
Initialize an event.....	53
Continue an event.....	57
Update Bridgemate settings.....	59
Update the movement for a section.....	61
Delete a section .....	63
Update scoringgroups.....	64
Delete scoringgroups.....	65
Update player data.....	66
Update participations.....	67
Update handrecords .....	68
Manage BCS .....	69
Explanation of used terms.....	71

## Revision history

---

**Document version:** 0,1  
**Release date:** 2024-02-18  
**BCS.Net version:** 1.0.810  
**Changes:** Initial draft

## General

### Purpose of this document

This document describes how to use the Bridgemate Data Connector (the Data Connector for short) to exchange movement data, results, player data and handrecords between an external Bridge scoring program and the .Net version of the Bridgemate Control Software (BCS.Net, BCS for short). It is not a manual for how to use the Bridgemate. By reading this document and examining the sample files, you should be able to integrate the wireless Bridgemate in your own scoring software and make full use of its functionality. We strongly advise to read the English manual of Bridgemate as well in order to have full understanding of the Bridgemate scoring system.

It still is possible to exchange data between Bridgemate and an external program by writing to and reading from a database (.bws) file. For this consult the Bridgemate developer's guide.

Mind that the Bridgemate Data Connector only works with the .Net version of BCS.

### Data exchange between Bridgemate and external programs

When using the Bridgemate Data Connector data is exchanged by sending serialized data (as json) to and reading serialized data (as json) from an intermediary process using a named pipe. This intermediary process is called the Bridgemate Data Connector. External programs can write movement data, board results, player data and handrecords to the Data Connector, where BCS will retrieve it. Likewise BCS writes board results, player data and handrecords to the Data Connector process where the external program can retrieve it. The Data Connector retains the data from one side until it has been retrieved by the other side and until the other side has signalled that it has processed the data. It will always be possible to retrieve all data again, even if it has been processed before. Apart from data exchange the external program can pass commands to BCS through the Data Connector to start up BCS, to start up BCS and create a scoring file for its own use and to shut down BCS.

### The Scoringprogram pipe client

Bridge Systems provides the source code for a client dll, the BridgeSystems.Bridgemate.DataConnector.ScoringProgramClient.dll, that handles connect, disconnect, data exchange and management commands with the Bridgemate Data Connector. This client can either be accessed directly if the programming platform for the external program supports it, or its code can be used as examples how to implement communication with the Data Connector. The client comes with its own documentation, available at [the Bridgemate github space](#).

### Data exchange format

All data is exchanged using serialization in the json format. Please refer to the examples. All programming platforms support both writing and reading of json formatted data, making it an universal data exchange format. More information can be found at [JSON](#)

### Differences in architecture

The table below summarizes the differences in architecture between using data exchange using the classic database based method and the newer intermediary process based approach.

Topic	Database based approach	Data Connector based approach	Remarks
BCS data	BCS and external program both read and write to the same database (.bws) file	BCS has its own scoring file. No other program should read or write to it.	The Data Connector stores the exchange data in its own data tables

Consecutive actions	No new movement data, handrecords or settings updates can be sent before the previous batch has been processed by BCS as the new batch may overwrite the previous batch with instructions	The external program can send consecutive batches of instructions (with data). These batches will be processed by BCS in the same order.	The .bws Tables table has an UpdateFromRound field that contains instructions for updates to that table. This field should only be modified when its original value is zero. Failure to do so will lead to the loss of the previous instruction contained in this field.
Concurrency conflicts	Both BCS and the external program may have to deal with locked database files and concurrent writes to a table (or record).	All data sent is atomic: it is not visible to the other side before it has been completely sent. Conflicts are resolved by the Data Connector process.	
New versions	Newer functions for the Bridgemates may at some time no longer be supported.	New functions will always be supported. Effort will be put in maintaining backward compatibility.	The .bws file is a legacy, MS-Access 2000 based, database format. Using the Data Connector BCS.Net has its own database format, which will support all future functions of the Bridgemates.

## Data exchange

---

Data is exchanged by sending as JSON serialized data over a communication channel. Currently only communication using Named Pipes is supported.

### Named Pipes

Named pipes are used to send and receive data to and from the Data Connector. This section describes how to set up communication using named pipes.

### The Named Pipe server

The server side Named Pipe is installed when installing the Bridgemate Control Software. It runs as a Windows service as soon as the computer has been started up. Its name is

`"BridgeSystems.Bridgemate.DataConnectorService"`.

Mind that the pipe only accepts one connection.

### The Named Pipe client

The Named Pipe client must be created by the scoring program and should connect to the `"BridgeSystems.Bridgemate.DataConnectorService.ScoringProgram"` Named Pipe. Note that the pipe only accepts one connection. The pipe server will detect when then pipe client dies and will then accept a new connection. Alternatively the client can issue a Disconnect command to free up the pipe. A full `ScoringProgramPipeClient` class is provided in the `BridgeSystems.Bridgemate.DataConnector.ScoringProgram.dll` written by Bridge Systems. This class provides functions to connect, ping and exchange data with the pipe server. To use it the external scoring program must be written in .Net 4.5 or higher or it must be able to use an adapter for using .Net Standard 2.0 code for its programming platform.

### Code examples

A typical procedure to connect to the pipe server and test communication with it would be in .Net:

```
private NamedPipeClientStream _pipeClient;
public StreamWriter Writer {get;set;};
public StreamReader Reader {get;set;};

public async Task<bool> TryConnect()
{
    _pipeClient = new NamedPipeClientStream(".",
"BridgeSystems.Bridgemate.DataConnectorService.ScoringProgram",
    PipeDirection.InOut, PipeOptions.None,
    TokenImpersonationLevel.Impersonation);

    bool connected;
    try
    {
        await _pipeClient.ConnectAsync(TimeoutInMilliseconds);
        connected = true;
        Reader = new StreamReader(_middleManStream);
        Writer = new StreamWriter(_middleManStream);
        Writer.AutoFlush = true;

        string message="Hello world!";
        string serializedMessage=JsonSerializer.Serialize(message);
        await Writer.WriteLineAsync(serializedMessage);
        string response = await Reader.ReadLineAsync();
    }
}
```

```

        catch (TimeoutException ex)
        {
            connected=false;
        }
        return connected;
    }

```

In Java the code could be something like:

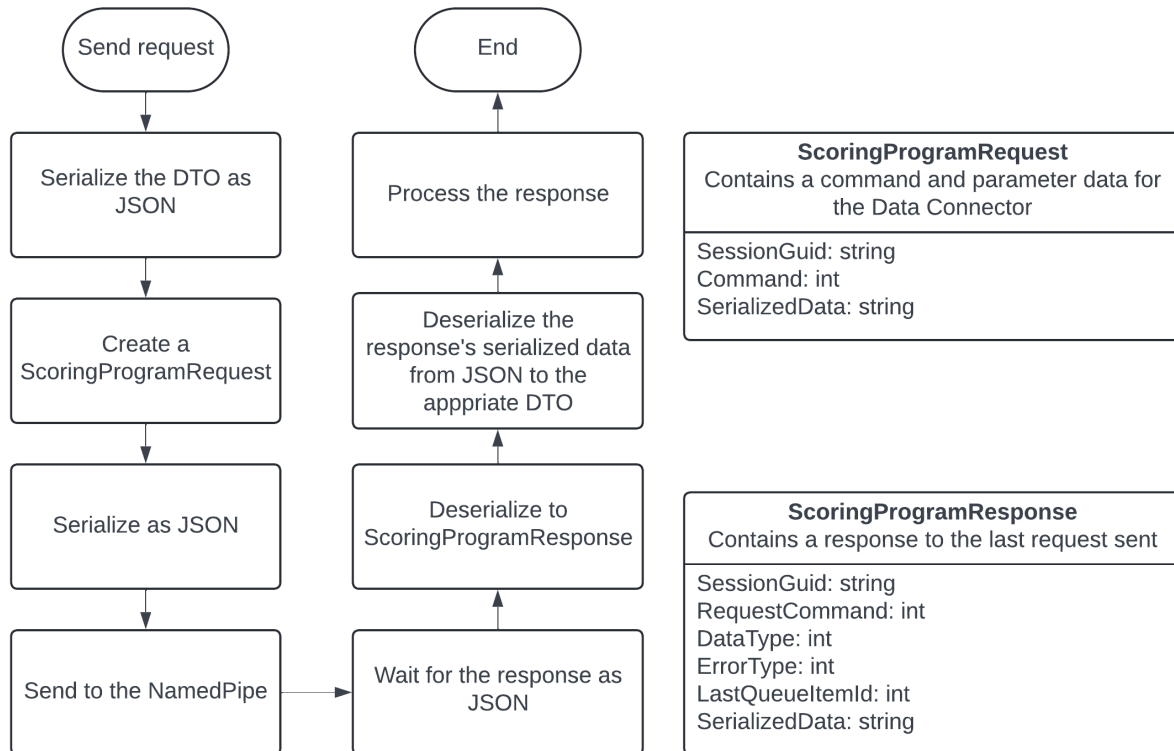
```

try {
    // Connect to the pipe
    RandomAccessFile pipe =
        new
RandomAccessFile("\\\\.\\pipe\\BridgeSystems.Bridgemate.DataConnectorService.ScoringProgram"
, "rw");
    String echoText = "Hello word\n";
    // write to pipe
    pipe.write ( echoText.getBytes() );
    // read response
    String echoResponse = pipe.readLine();
    System.out.println("Response: " + echoResponse );
    pipe.close();
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

```

## Overview of communication

This section describes how to exchange data with the Bridgemate Data Connector.



## General description

### ***Sending data***

The scoring program wants to send a command to the Data Connector with accompanying data. The scoring program constructs a DTO containing the data the Data Connector needs to work with. The type of DTO depends on the requested action as expressed in the command. The DTO is serialized as JSON and added to a `ScoringProgramRequest` class. This request contains a command that specifies what Data Connector should do. The request must contain the guid of the session that the request belongs to. The request is then serialized a second time and sent to the Data Connector over the communication channel. The Data Connector will always respond. The scoring program needs to wait for this response and consume it. Otherwise communication will stall. Mind that nearly all communications are scoped to a single session as expressed by the `SessionGuid` property of the request. So, say, a request for new board results must be done for each session that the scoring program currently administers.

### ***Receiving the response***

The Data Connector serializes its response data as JSON. Then it constructs a `ScoringProgramResponse` with this data. The response contains the original command from the request and the session guid that was sent with it in order to check if the response indeed is the answer to the request that was sent. The response has a `DataType` property that specifies to what DTO the JSON data can be deserialized. The scoring program waits for the response on the communication channel and deserializes it to a `ScoringProgramResponse`. Depending on the response's `DataType` the response's `SerializedData` can be deserialized from JSON to the appropriate DTO (or to a message).



## Overview of the commands for the ScoringProgramRequest

The tables below give a short description of the available commands for the ScoringProgramRequest and the associated DTO that must be sent with it.

### *Sending data to the Bridgemate Data Connector*

Command	Value	Description	DTO	Remarks
Connect	1	Connects to the communication channel	n/a	Does not transfer any data
Disconnect	3	Disconnects from the communication channel	n/a	Does not transfer any data. The communication channel will wait for a new connection.
Ping	4	Sends a string to the Data Connector and awaits its return	string	Serialize a custom string. On deserializing the response's SerializedData the same string should return. Can be used to check if the communication channel is alive.
InitializeEvent	5	Sends start-up parameters for BCS and initialization data	<a href="#">InitDTO</a>	The InitDTO contains both start-up parameters for BCS and the data for the sessions like movements, player data and hand records. With this data a new scoring file will be created for internal use by BCS.
ContinueEvent	30	Sends start-up parameters for BCS without initialization data	n/a	Instructs the Data Connector to start up BCS if it is not running and to continue processing data using an existing scoring file
ManageBCS	33	Can send a request asking BCS to shut down immediately and can query information on administered events	<a href="#">BCSManagementRequestDTO</a>	Does not transfer any data.
UpdateMovement	6	Instructs BCS to change the movement for the given section, or to delete it.	<a href="#">SectionDTO</a>	The SectionDTO holds Tables as an array of TableDTOs. Each TableDTO holds Rounds as an array of RoundDTOs. Each RoundDTO contains data on the opponents and the boards played. The SectionDTO holds a IsDeleted property that can be set to 'True' if the section must be deleted. If the section does not yet exist, it will be added.
UpdateScoringGroups	7	Instructs BCS to update its scoring groups	<a href="#">Array of ScoringGroupDTO</a>	All scoring groups for the session with their sections must be sent as all sections must have a parent scoring group. Mind to use this command before adding a new section using the UpdateMovement command that has a new scoring group.
PutResults	9	Sends board results to the Data Connector	<a href="#">Array of ResultDTO</a>	Using this command new or modified board results from the scoring program can be sent to the Bridgemates.

PutPlayerData	10	Sends player data to the Data Connector	<a href="#">Array of PlayerDataDTO</a>	Player data contains a person's first name, last name, organization id and playernumber. The player numbers must be known to BCS before players can make themselves be known by entering their number on the Bridgemate. The player data can be included in the InitDTO or sent separately. The first method is more performant.
PutParticipations	11	Sends participations to the Data Connector	<a href="#">Array of ParticipationDTO</a>	The participation contains information on the starting position of a player: section, table, round and seat direction. The player can either be identified by a player number, in which case the Data Connector will look up the first and last names or by first name and last name only. The participations can be included in the InitDTO or sent separately. The first method is more performant.
PutHandrecords	12	Sends handrecords to the Data Connector	<a href="#">Array of HandrecordDTO</a>	The handrecords must have a valid scoringgroup id. Make sure to add the scoringgroup first using the UpdateScoringGroup command if a new section was added. The handrecords can be included in the InitDTO or sent separately. The first method is more performant.
PutBridgemate2Settings	13	Sends the Bridgemate 2 settings	<a href="#">Array of Bridgemate2SettingsDTO</a>	The settings must be sent for each section separately. The settings can be included in the InitDTO or sent separately. The first method is more performant.
PutBridgemate3Settings	14	Sends the Bridgemate 2 settings	<a href="#">Array of Bridgemate3SettingsDTO</a>	The settings must be sent for each section separately. The settings can be included in the InitDTO or sent separately. The first method is more performant.

### ***Polling data from the Bridgemate Data Connector***

Command	Value	Description	DTO	Remarks
PollQueueForNewResults	18	Polls for new board results	returns: array of <a href="#">ResultDTO</a>	Returns the new board results since the last batch was accepted.
PollQueueForNewPlayerData	19	Polls for new player data	returns: array of <a href="#">PlayerDataDTO</a>	Returns the new player data since the last batch was accepted.
PollQueueForNewParticipations	20	Polls for new participations	returns: array of <a href="#">ParticipationDTO</a>	Returns the new participations since the last batch was accepted.
PollQueueForNewHandrecords	21	Polls for new handrecords	returns: array of <a href="#">HandrecordDTO</a>	Returns the new handrecords since the last batch was accepted.

PollQueueForAllResults	22	Requests all board results, irrespective of them having been accepted before.	returns: array of <a href="#">ResultDTO</a>	Returns all board results that have been created by BCS.
PollQueueForAllPlayerData	23	Requests all player data, irrespective of them having been accepted before.	returns: array of <a href="#">PlayerDataDTO</a>	Returns all playerdata that has been created by BCS
PollQueueForAllParticipations	24	Requests all participations, irrespective of them having been accepted before.	returns: array of <a href="#">ParticipationDTO</a>	Returns all participations that have been created by BCS
PollQueueForAllHandrecords	25	Requests all handrecords results, irrespective of them having been accepted before.	returns: array of <a href="#">HandrecordDTO</a>	Returns all handrecords that have been created by BCS.
AcceptResultQueueItems	26	Signals to the Data Connector to not send board results that have been sent before.	The id as serialized integer of the last processed result queue item	Signals to the Data Connector to not send board results that have been sent before.
AcceptPlayerDataQueueItems	27	Signals to the Data Connector to not send playerdata that has been sent before.	The id as serialized integer of the last processed player data queue item	Signals to the Data Connector to not send playerdata that has been sent before.
AcceptParticipationQueueItems	28	Signals to the Data Connector to not send participations that have been sent before.	The id as serialized integer of the last participation queue item	Signals to the Data Connector to not send participations that have been sent before.
AcceptHandrecordQueueItems	29	Signals to the Data Connector to not send handrecords that have been sent before.	The id as serialized integer of the last processed handrecord queue item	Signals to the Data Connector to not send handrecords that have been sent before.
GetMovement	31	Requests the movement for a specific section	sends: <a href="#">sectionDTO</a> with the desired SessionGuid and Letters. returns: sectionDTO	The sectionDTO has Tables as an array of TableDTO, each table has Rounds as an array of RoundDTO. The RoundDTO contains the opponents and the board numbers that they will play.
GetAllMovements	32	Requests all movements for a session.	sends: n/a returns: an array of <a href="#">SectionDTO</a>	Each sectionDTO has Tables as an array of TableDTO, each table has Rounds as an array of RoundDTO. The RoundDTO contains the opponents and the board numbers that they will play.

## Data types

The ScoringProgramResponse has a DataType property which specifies which type of data can be

expected to be in its SerializedData property.

Value	Name	Description
1	OK	The scoring program request was handled correctly. No serialized data is included in the response
2	Not in use	
3	Not in use	
4	Error	There was an error in processing the scoring program request. Examine the ErrorType property and deserialize the serialized data as a string for detailed information
5	InitiData	Not in use for the scoring program client
6	ContinueData	Not in use for the scoring program client
7	SectionData	Not in use for the scoring program client
8	Bridgemate2Settings	Not in use for the scoring program client
9	Bridgemate3Settings	Not in use for the scoring program client
10	PlayerData	The serialized data is an array of <a href="#">PlayerDataDTO</a>
11	Participations	The serialized data is an array of <a href="#">ParticipationDTO</a>
12	Results	The serialized data is an array of <a href="#">ResultDTO</a>
13	Handrecords	The serialized data is an array of <a href="#">HandrecordDTO</a>
14	Movement	The serialized data is a <a href="#">SectionDTO</a>
15	Sessions	The serialized data is an array of <a href="#">SectionDTO</a>
16	EventInfo	The serialized data is BCSManagementResponseDTO containing an array of SessionInfoDTO
17	AllSessionsInfo	The serialized data is BCSManagementResponseDTO containing an array of SessionInfoDTO
18	ScoringFileLocation	The serialized data is BCSManagementResponseDTO
19	ShutDownRequest	Not in use for the scoring program client

## Error codes

When the ScoringProgramResponse.DataType property is "Error" (4) then the ScoringProgramResponse.ErrorType property will hold a value further explaining what went wrong. Moreover the SerializedData property can be deserialized to a string to obtain detailed debugging information. This information is not meant to be shown to the end users of the external scoring program.

Value	Name	Description
0	None	The ScoringProgramRequest was handled successfully
1	Busy	The ScoringProgramRequest could not be processed because a previous request has not yet been completed. Try again.
2	NoData	The ScoringProgramRequest.Command requires data to be sent with it, but there is none.
3	NoUpdates	The ScoringProgramRequest.Command included data to be updated, but the said data is already present.
4	Movement	The included data did not comply to a known section, table in a section or round on a table.
5	Validation	The sent data did not pass validation. Deserialize the SerializedData to a string for details.
6	EntryUnknown	The provided data has a (composite) primary key that can not be computed. Deserialize the SerializedData to a string for details.
7	Exception	An error occurred while processing the data. Study the DataConnector.log to find details.
8	NotImplemented	The requested command is not implemented
9	EmptyResponse	The Data Connector did not respond to the request.
10	NoConnection	The communication with the Data Connector is broken
11	TimeOut	The request was blocked by a previously sent long running operation on the Data Connector.

12	WrongDataType	The datatype of the dtos did not conform to the request command.
13	UnexpectedCommand	The response command did not conform to the request command.
14	Unknown	Unknown error. Currently not in use.

## Code examples

Below a typical piece of code to send a request to the Data Connector and awaiting the response: using the Ping command

```

    /// <summary>
    /// Communicates to the DataConnector to see if it is responsive.
    /// The Ping command returns the exact data that was sent with it.
    /// </summary>
    /// <returns></returns>
    public async Task<ScoringProgramResponse> Ping()
    {
        var requestTicks = DateTime.Now.Ticks.ToString();
        var serializedTicks=JsonSerializer.Serialize(requestTicks);
        var response = await SendDataAsync(sessionGuid: string.Empty,
        ScoringProgramMiddleManCommands.Ping, serializedTicks);
        if (response.RequestCommand != ScoringProgramMiddleManCommands.Ping)
        {
            return new ScoringProgramResponse
            {
                RequestCommand = ScoringProgramMiddleManCommands.Ping,
                DataType = MiddleManResponseData.Error,
                ErrorType = ErrorType.Unknown,
                SerializedData = JsonSerializer.Serialize($"Invalid command in reponse to
{nameof(ScoringProgramMiddleManCommands.Ping)}: " +
                    $"{response.RequestCommand}")
            };
        }
        if (response.DataType != MiddleManResponseData.OK)
            return response;

        var responseTicks = JsonSerializer.Deserialize<string>(response.SerializedData);
        var error = responseTicks != requestTicks;
        return new ScoringProgramResponse
        {
            RequestCommand = ScoringProgramMiddleManCommands.Ping,
            DataType = requestTicks == responseTicks ? MiddleManResponseData.OK :
MiddleManResponseData.Error,
            ErrorType = error ? ErrorType.Validation : ErrorType.None,
            SerializedData = response.SerializedData
        };
    }

    /// <summary>
    /// The code that handles the actual sending of requests and reading their reponses.
    /// </summary>
    /// <param name="sessionGuid">Specifies which session the request targets (if
any)</param>
    /// <param name="command">The command to the middleman</param>
    /// <param name="serializedData">The data to send to the middleman as json data. (If
any)</param>
    /// <returns></returns>
    private async Task<ScoringProgramResponse> SendDataAsync(string sessionGuid,
    ScoringProgramMiddleManCommands command, string serializedData)
    {
        //Construct the request to the Middleman.
        var request = new ScoringProgramRequest
        {
            Command = command,
            SessionGuid = sessionGuid,
            SerializedData = serializedData
        };

        //Serialize it.
        var requestSerialized = JsonSerializer.Serialize(request);

        //Do not proceed if sending is already in progress (for an other request). There
can be only on request be sent at the same time.
        if (!_isSending)

```

```

{
    return new ScoringProgramResponse
    {
        RequestCommand = command,
        SessionGuid = sessionGuid,
        DataType = MiddleManResponseData.Error,
        ErrorType = ErrorType.Busy,
        SerializedData = JsonSerializer.Serialize($"Client is busy, please retry
later.")
    };
}
try
{
    _isSending = true;

    //Do not continue if the connection has been broken. Call the Connect method
again then resend.
    var errorResponse = CheckConnection(command);
    if (errorResponse != null)
    {
        return errorResponse;
    }

    //Reconnect to the Middleman if needed.
    if (!MiddleManStream.IsConnected)
    {
        await MiddleManStream.ConnectAsync(5000);
    }

    //Send the request to the Middleman. Mind: as it is written now this is a
blocking call.
    //However, in .Net an exception will be thrown if the connection has gone dead
for whatever reason.
    await MiddleManWriter.WriteLineAsync(requestSerialized);

    //Wait for the response. This too is a blocking call. But in .Net a broken
connection will throw an exception.
    string response = await MiddleManReader.ReadLineAsync();
    if (response != null)
    {
        var responseDeserialized =
JsonSerializer.Deserialize<ScoringProgramResponse>(response);
        return responseDeserialized ??
            new ScoringProgramResponse
            {
                RequestCommand = command,
                DataType = MiddleManResponseData.Error,
                SerializedData = JsonSerializer.Serialize("Empty response")
            };
    }
    else
    {
        return new ScoringProgramResponse
        {
            RequestCommand = command,
            DataType = MiddleManResponseData.Error,
            SerializedData = JsonSerializer.Serialize("Empty response")
        };
    }
}
catch (IOException)
{
    CloseConnection();
    return
    new ScoringProgramResponse
    {
        RequestCommand = command,
        DataType = MiddleManResponseData.Error,
        SerializedData = JsonSerializer.Serialize("Pipe broken")
    }
}

```

```

        };
    }
    catch (Exception ex)
    {
        CloseConnection();
        DebugLogger.Error(ex);
        ErrorLogger.Error(ex);
        return
            new ScoringProgramResponse
            {
                RequestCommand = command,
                DataType = MiddleManResponseData.Error,
                SerializedData = JsonSerializer.Serialize(ex.Message)
            };
    }

    finally
    {
        //Always signal that the client is free for the next items to send.
        //Otherwise after an exception further communication will be blocked.
        _isSending = false;
    }
}

```



## Data Transfer Objects

### Introduction

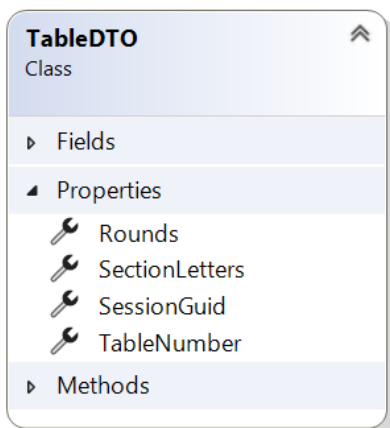
All data exchange between the external scoring program and the Data Connector is done using DTOs: Data Transfer Objects. How a DTO is structured will be dependent on the programming language used. This document will use UML to show the structure of the DTOs. A DTO must be serialized as json when sent to the Data Connector. When the Data Connector responds with data, this data will also be a json-serialized DTO.

Obviously a DTO can only have public properties and have no functions.

Mind: the requests to and the responsesn from the Data Connector are layered and have two levels of json data. See the section on how to exchange data for a detailed explanation.

### Example

This is a class diagram of the TableDTO, representing a table in the session's movement.



In C# the TableDTO class or struct would be something like:

```

public class TableDTO
{
    public string SessionGuid
    {
        get; set;
    }

    public string SectionLetters
    {
        get; set;
    }

    public int TableNumber
    {
        get; set;
    }

    public RoundDTO[] Rounds
    {
        get; set;
    }
}
  
```

while in Java it would be something like:

```
public class TableDTO {
    private String sessionGuid;
    private String sectionLetters;
    private int tableNumber;
    private RoundDTO[] rounds;

    public String getSessionGuid() {
        return sessionGuid;
    }

    public void setSessionGuid(String sessionGuid) {
        this.sessionGuid = sessionGuid;
    }

    public String getSectionLetters() {
        return sectionLetters;
    }

    public void setSectionLetters(String sectionLetters) {
        this.sectionLetters = sectionLetters;
    }

    public int getTableNumber() {
        return tableNumber;
    }

    public void setTableNumber(int tableNumber) {
        this.tableNumber = tableNumber;
    }

    public RoundDTO[] getRounds() {
        return rounds;
    }

    public void setRounds(RoundDTO[] rounds) {
        this.rounds = rounds;
    }
}
```

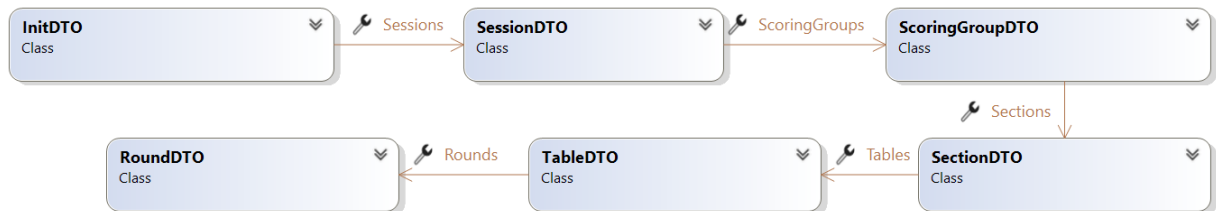
The json serialized data for a TableDTO representing table A3 with two rounds would be something like:

```
{
  "SessionGuid": "4f24d8a2-2b6c-4a72-9e6a-8901a5a8b3c1",
  "SectionLetters": "A",
  "TableNumber": 3,
  "Rounds": [
    {
      "SessionGuid": "4f24d8a2-2b6c-4a72-9e6a-8901a5a8b3c1",
      "SectionLetters": "A",
      "TableNumber": 3,
      "RoundNumber": 1,
      "LowBoardNumber": 9,
      "HighBoardNumber": 12,
      "PairNS": 5,
      "PairEW": 6,
    },
    {
      "SessionGuid": "4f24d8a2-2b6c-4a72-9e6a-8901a5a8b3c1",
      "SectionLetters": "A",
      "TableNumber": 3,
      "RoundNumber": 2,
      "LowBoardNumber": 9,
      "HighBoardNumber": 12,
      "PairNS": 4,
      "PairEW": 8,
    }
  ]
}
```

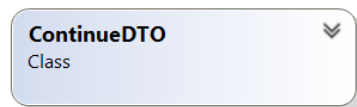
## Overview

Below you find all Data Transfer Objects (DTOs) and their relation according to the desired action towards BCS.

### Initialization



### Continuation



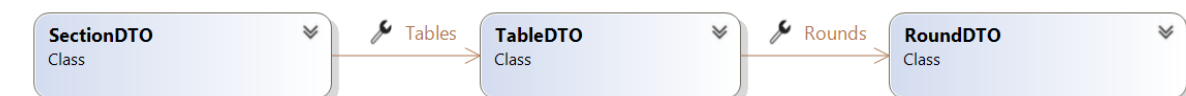
### Update Bridgemate settings



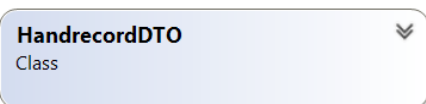
### Update scoringgroups



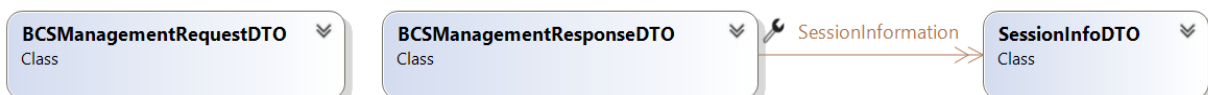
### Update movement



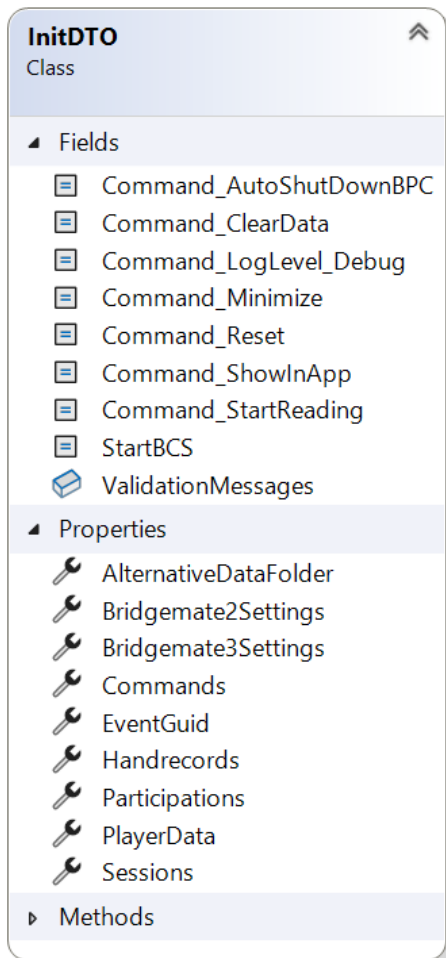
### Update handrecords



### BCS Management



## InitDTO



The InitDTO is used with the [InitializeEvent](#) Command.

### Commands property

The command property is the sum of actions that BCS should perform:

Value	Description	Remarks
1	Start BCS	Instructs Bridgemate Control Software (BCS) to launch. Always include.
2	Reset	Instructs BCS to create a new scoring file based on the provided data, clear the Bridgemate servers and upload new data to them. Always include. If the data in the Bridgemates should not be cleared then use the <a href="#">ContinueDTO</a> with the <a href="#">ContinueEvent</a> command.
4	Start reading	Instructs BCS to start reading from the Bridgemates and the Bridgemate Data Connector. Optional
8	Show in App	Instructs BCS to upload all session to the App back-end. Alternatively this can be specified on the SessionDTOs seperately
16	Minimize	BCS will start minimized.
32	Auto shutdown	Instructs BCS to shut down after the last result has been processed.
64	Loglevel debug	Lowens the Log level from "Info" to "Debug".
128	Clear data	Instructs the Bridgemate Data Connector to clear all data. This prevents stale data of the same sessions that are contained in the InitDTO from being processed. On the other hand it will also delete data from other sessions that may await further processing. Use with caution in situations where multiple events may be ongoing.

Typically the Commands property will be  $1+2+4+128=135$ . The other values are optional.

***EventGuid property***

This is required when there is more than one session. When there is one session, its SessionGuid property will be reused as the event guid. This property will help discern scoring groups with the same scoring group number between events.

***Sessions property***

At least one is required. See the details on the [SessionDTO](#) for further details. The SessionDTO must contain the movement data.

***PlayerData property***

Optional. Can be sent separately using the [PutPlayerData](#) command as well. The first name, last name of each player that could participate in the event, uniquely defined by the combination of the SessionGuid of one of the event's sessions and a PlayerNumber defined by the organization, usually the Bridge league the player is a member of.

***Participations property***

Optional. Can be sent separately using the [PutParticipations](#) command as well. Starting positions for each player. The combination of SessionGuid and PlayerNumber must be present in the PlayerData.

***Handrecords property***

Optional. Can be sent separately using the [PutHandrecords](#) command as well.

***Bridgemate2Settings property***

Optional. The settings for the Bridgemate II's. Can be sent separately using the [PutBridgemate2Settings](#) command as well.

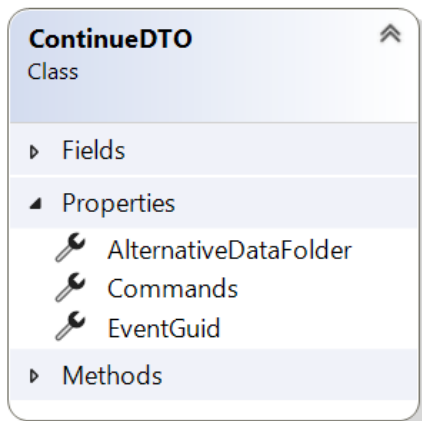
***Bridgemate3Settings property***

Optional. The settings for the Bridgemate III's. Can be sent separately using the [PutBridgemate3Settings](#) command as well.

***AlternativeDataFolder***

Only used in advanced scenario's. Not documented here.

## ContinueDTO



The ContinueDTO is used with the [ContinueEvent](#) command

### **EventGuid property**

The guid of the [event](#) that BCS should continue administering. If the [event](#) has one [session](#) the session's SessionGuid property can be used if this property was also used in the [event initialization](#).

### **Commands property**

The command property is the sum of actions that BCS should perform after start-up:

Value	Description	Remarks
1	Start BCS	Instructs Bridgemate Control Software (BCS) to launch. Always include.
4	Start reading	Instructs BCS to start reading from the Bridgemates and the Bridgemate Data Connector. Optional
16	Minimize	BCS will start minimized. Optional
32	Auto shutdown	Instructs BCS to shut down after the last result has been processed. Optional
128	Clear data	Instructs the Bridgemate Data Connector to clear all data. This prevents stale data of the same sessions that are contained in the InitDTO from being processed. On the other hand it will also delete data from other sessions that may await further processing. Use with caution in situations where multiple events may be ongoing.

Typically the Commands property will be 1+4=5. The other values are optional.

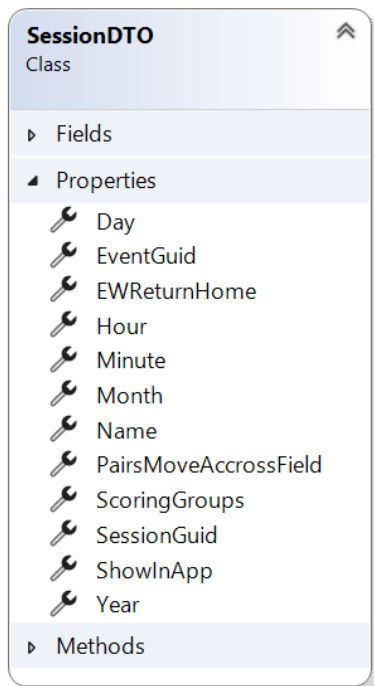
### **Note:**

The other command values as specified for the [InitDTO](#) are invalid and if used will lead to the [ContinueEvent](#) command not being executed.

### **AlternativeDataFolder**

Only used in advanced scenarios. Not documented here.

## SessionDTO



The SessionDTO can be used with the InitializeEvent command and will then be added to the InitDTO.Sessions property.

### EventGuid property

Optional. Must be present if there is more than one session.

### SessionGuid property

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### Name property

Optional, but recommended.

### Year property

Required. Must be at least 2000.

### Month property

Required. Must be between 1 and 12

### Day property

Required. Must be between 1 and 31. Must match with the highest day for the month

### Hour property

Optional. Must be between 0 and 23

### Minute property

Optional. Must be between 0 and 59

### ShowInApp property

Signals that the session should be uploaded to the Bridgemate App.



***ScoringGroups property***

Required. Array of ScoringGroupDTOs. At least one must be present. See [ScoringGroupDTO](#) for details.

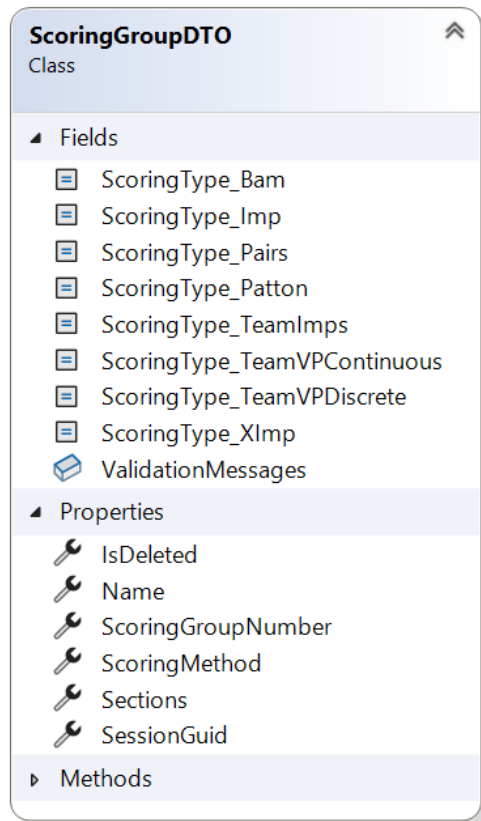
***EWReturnHome property***

Currently not supported.

***PairsMoveAccrossField property***

Currently not supported.

## ScoringGroupDTO



The ScoringGroupDTO defines the scoring method for its sections and is used for the score calculation: it will group the results of all the participants of its sections together. It will be part of the a SessionDTO in an InitDTO, or it can be sent separately using the [UpdateScoringGroups](#) command.

### ***SessionGuid property***

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### ***ScoringGroupNumber property***

Required. An integer value uniquely defining the scoring group within the event. Must be greater than zero.

### ***ScoringMethod property***

Required. Values can be:

- 10 for Matchpoints
- 20 for Imps
- 30 for Cross Imps
- 40 for Team impes
- 50 for Discrete Victory Points
- 51 for Continuous Victory Points
- 60 for Board-a-Match
- 70 for Patton

### ***Name property***

Optional

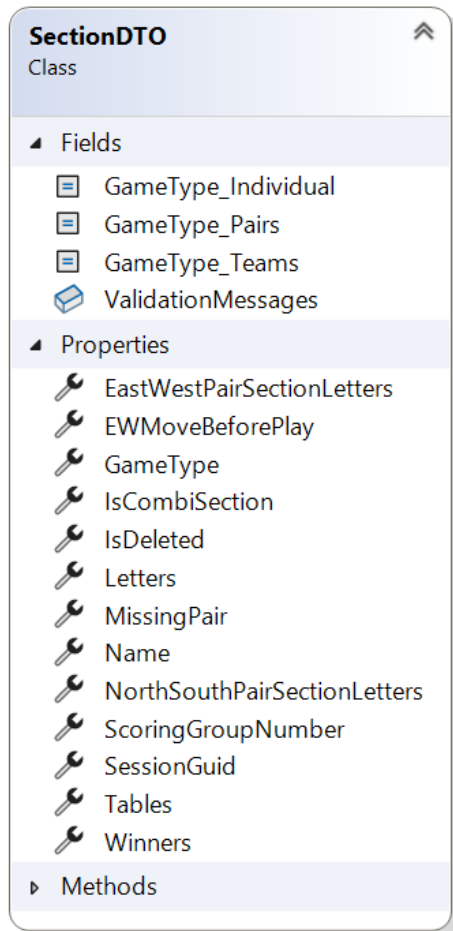
### ***Sections property***

Required. An array of [SectionDTOs](#) whose results will be calculated together as if they were one section.

### ***IsDeleted property***

Only used for the Update Scoringgroups command. Indicates that the scoring group has no more sections and can be deleted. Before deleting a scoring group make sure that its former sections have been assigned to a different scoring group.

## SectionDTO



The SectionDTO contains the movement for a group of participants for the duration of the session. It will be part of a [ScoringGroupDTO](#) which is part of a [SessionDTO](#) which is part of an [InitDTO](#), or it can be sent separately with the [UpdateMovement](#) command.

### **SessionGuid property**

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### **ScoringGroupNumber**

Required. The number of the ScoringGroupDTO that the section belongs to. Must be greater than zero.

### **Letters property**

Required. Uniquely defines the section within [the event](#).

### **Winners property**

Required. Values can be 1 or 2. In the case of 2 winners the pair numbers in the section can be the same for North-South and East-West. Otherwise the pairnumbers in the section must be unique.

### **GameType property**

Required. Values can be 10 for "Pairs", 20 for "Individual" and 30 for "Teams".

### **Name property**

Optional.

***EWMoveBeforePlay property***

Currently not supported.

***MissingPair property***

Optional. If specified it will indicate the number for the pair that is not playing. Its opponents will have a sit-out when they are scheduled to play against this pair. This value can be omitted as a sit-out can also be specified on the RoundDTOs. However, if used the graphic representation of sit-out tables in BCS will be improved.

***IsCombiSection property***

Optional. If "true" the section will host the two pairs that would otherwise have a sit-out in their own sections. Specify the section that will provide the NorthSouth pair and the section that will provide the EastWest pair.

***NorthSouthPairSectionLetters property***

Required if IsCombiSection is "true". The letters for the section where the NorthSouth pair for each round comes from.

***EastWestPairSectionLetters property***

Required if IsCombiSection is "true". The letters for the section where the EastWest pair for each round comes from.

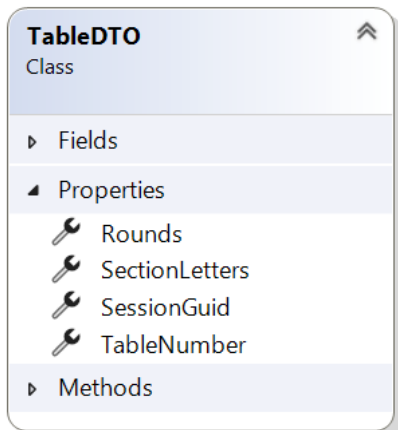
***IsDeleted property***

Can be used together with the [UpdateMovement](#) command to indicate that the section should be removed from BCS and the Bridgemates. Will be ignored otherwise.

***Tables property***

Must be present if IsDeleted is "false" as the SectionDTO contains the movement or a movement update for the section,  
Array of [TableDTO](#).

## TableDTO



The TableDTO represents the location where each round two pairs will meet to play boards against each other. A table automatically has a Bridgemate associated with it.

### ***SessionGuid property***

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### ***SectionLetters property***

Required. Refers to the section the table is part of.

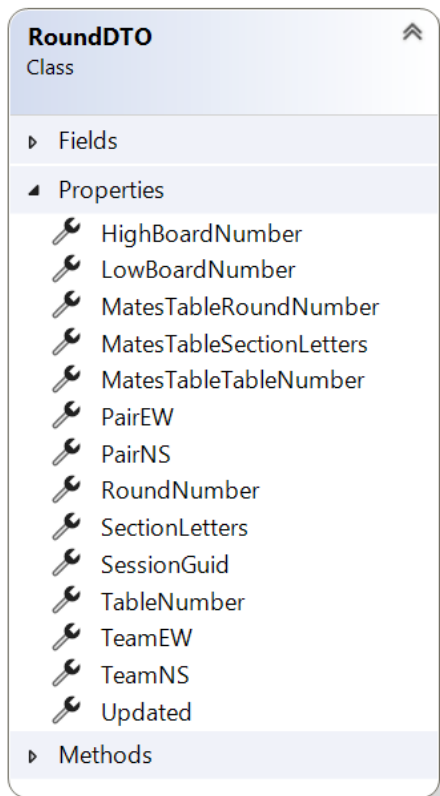
### ***TableNumber property***

Required. Uniquely defines the table in the section.

### ***Rounds property***

Required if the TableDTO is part of an InitDTO. If the TableDTO is part of a movement update, zero rounds means that the table should be deleted. An array of [RoundDTOs](#).

## RoundDTO



The RoundDTO defines which pairs play on its table in the specified round and which boards they will play. Currently only consecutive boards are supported.

### ***SessionGuid property***

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### ***SectionLetters property***

Required. Refers to the section the table is part of.

### ***TableNumber property***

Required. Uniquely defines the table in the section

### ***RoundNumber property***

Required. Uniquely defines the round on its table.

### ***PairNS property***

The number of the pair that takes the North-South position. If the value is greater than zero and if the section has one [winner](#) the number must be unique within all rounds with the same number in the section. If the section has two winners the number must be unique within all PairNS values within all rounds with the same number in the section. A value of zero will indicate a sit-out if the PairEW value is greater than zero or an empty table if PairEW is zero too.

### ***PairEW property***

The number of the pair that takes the East-West position. If the value is greater than zero and if the section has one [winner](#) the number must be unique within all rounds with the same number in the section. If the section has two winners the number must be unique within all PairEW values within all rounds with the same number in the section. A value of zero will indicate a sit-out if the PairNS value is greater than zero or an empty table if PairNS is zero too.

***LowBoardNumber property***

The number of the lowest board that the pairs will play. Together with the high board number property the value defines all boards that the pairs will play against each other.

***HighBoardNumber property***

The number of the highest board that the pairs will play. Together with the low board number property the value defines all boards that the pairs will play against each other.

***TeamNS property***

Optional. The number of the team for the North-South pair.

***TeamEW property***

Optional. The number of the team for the East-West pair.

***MatesTableSectionLetters property***

Optional. The letter of the section where the two other pairs of the teams match will play the same boards.

***MatesTableTableNumber property***

Optional. The number of the table where the two other pairs of the teams match will play the same boards.

***MatesTableRoundNumber property***

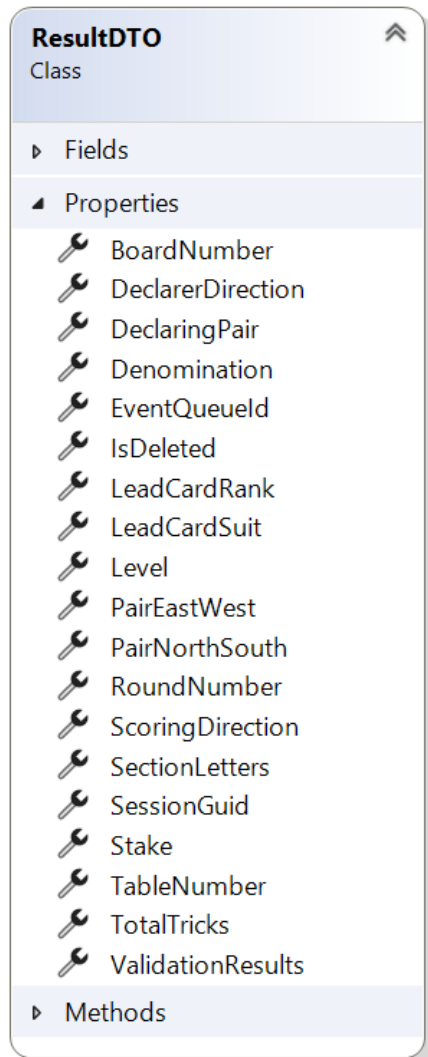
Optional. The number of the round where the two other pairs of the teams match will play the same boards.

***Updated***

Not in use for external scoring programs.



## ResultDTO



Represents the result on a board. Both natural and artificial results can be expressed, It is possible to send separate results for the North-South scoring side and East-West scoring side, but currently Bridgemate Control Software, the Bridgemate App and the Bridgemate servers do not support this.

### ***SessionGuid property***

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### ***SectionLetters property***

Required. Specifies the section in which the result was obtained.

### ***TableNumber property***

Required. Specifies the table on which the result was obtained.

### ***RoundNumber property***

Required. Specifies the round in which the result was obtained.

### ***BoardNumber property***

Required. Specifies the board on which the result was obtained.

**ScoringDirection property**

Required (cannot be zero). 1 for North-South, 2 for East-West, 3 for both, i.e.: no split score. As split scores for North-South and East-West currently are not supported use 3 always.

**PairNorthSouth property**

The number of the pair in the North-South position.

**PairEastWest property**

The number of the pair in the East-West position.

**DeclaringPair property**

Either the value of the PairNorthSouth property or the PairEastWest property.

**DeclarerDirection property**

Specifies the declarer on the board. Possible values are:

- 0: NA
- 1: North
- 2: East
- 3: South
- 4: West

**Note**

When the declarer direction property is North or South while the de declaring pair is the East-West pair the result is marked as being a switched seating. Likewise for East or West and the North-South pair.

**Level property**

Represent the level of the contract: 1 to 7 for natural scores and:

Value	Meaning	Comment
0	Pass	
-1	Avg minus/Avg minus	Both sides get Average minus
-2	Avg minus/ Avg	Average minus for the scoringdirection, Average for the opponents. When scoring direction is 3 it is handled as 1 (North-South)
-3	Avg minus/Avg plus	Average minus for the scoringdirection, Average plus for the opponents. When scoring direction is 3 it is handled as 1 (North-South)
-4	Avg/Avg minus	Average for the scoringdirection, Average minus for the opponents. When scoring direction is 3 it is handled as 1 (North-South)
-5	Avg/Avg	Average for both sides.
-6	Avg/Avg plus	Average for the scoringdirection, Average plus for the opponents. When scoring direction is 3 it is handled as 1 (North-South)
-7	Avg plus/Avg minus	Average plus for the scoringdirection, Average minus for the opponents. When scoring direction is 3 it is handled as 1 (North-South)
-8	Avg plus/Avg	Average plus for the scoringdirection, Average for the opponents. When scoring direction is 3 it is handled as 1 (North-South)
-9	Avg plus/Avg plus	Average plus for both sides
-10	No play	The board was cancelled.

**Denomination property**

The denomination for the contract, if applicable. Possible values are:

- 0: N/A (Pass, No play, artificial score)
- 1: Clubs

- 2: Diamonds
- 3: Hearts
- 4: Spades
- 5: No Trump

***Stake property***

Specifies if the contract was doubled or redoubled. Possible values are:

- 0: Not doubled
- 1: Doubled
- 2: Redoubled

***TotalTricks property***

Specifies the total number of tricks that were obtained by the declaring side. Zero if N/A.

***LeadCardRank property***

Optional. The rank of the lead card, if specified. Possible values are:

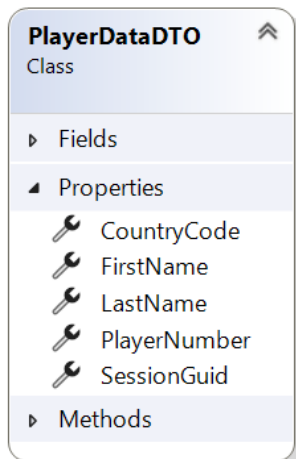
- 0: N/A
- 2-10: the card value
- 11: Jack
- 12: Queen
- 13: King
- 14: Ace

***LeadCardSuit property***

Optional, required if the lead card rank is other than zero. The suit of the leadcard. Possible values are:

- 0: N/A
- 1: Clubs
- 2: Diamonds
- 3: Hearts
- 4: Spades

## PlayerDataDTO



The PlayerDataDTO contains the name and identification data for a player that may compete in a specific session.

The player is uniquely defined by the combination of SessionGuid and PlayerNumber. Both values are therefore required.

### Note

Because of the above a player must be sent to Bridgemate Data Connector for each session that it may compete in.

If players can make themselves known at their first table by entering their player number on the Bridgemate make sure that all for all possible players a PlayerDataDTO has been sent for that session.

The PlayerDataDTO can be used as part of the [InitDTO](#) for the [event initialization](#), or it can be sent as data for the [PutPlayerData command](#).

### **SessionGuid property**

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### **PlayerNumber property**

Required. Uniquely defines the player (for this session).

### **FirstName property**

Optional. The first name of the player.

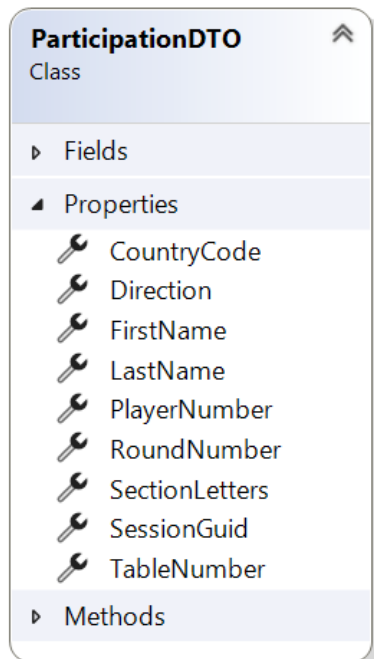
### **LastName property**

Required. The last name of the player.

### **CountryCode property**

Currently not supported.

## ParticipationDTO



The ParticipationDTO specifies for each round on a table which players occupy the four seats, Only specify participation for known players.

The ParticipationDTO can be used in two ways:

1. By specifying the SessionGuid and PlayerNumber. Make sure that a corresponding [PlayerDataDTO](#) with the same SessionGuid and PlayerNumber has been sent before sending the participation. Do not include first name or last name of the player.
2. By specifying the SessionGuid and at least the PlayerLastName. Internally Bridgemate Data Connector will make a registration of this player. Do not include the player number of the player.

In theory you could send all participations for all rounds. However, currently this is not supported. Bridgemate Control Software will determine the participations for round two and higher from the movement as sent with the [SectionDTO](#). So leave the RoundNumber at zero, or set it to 1.

You can send the participations as part of the [InitDTO](#) when [initializing a new event](#). Or they can be sent separately as data for the [PutParticipations command](#).

### ***SessionGuid property***

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### ***SectionLetters property***

Required. Specifies the section for the participation.

### ***TableNumber property***

Required. Specifies the table for the participation.

### ***RoundNumber property***

Currently not supported.

### ***PlayerNumber property***

Optional. Together with the SessionGuid uniquely defines the player. The player must have been sent to Bridgemate Data Connector using a [PlayerDataDTO](#) beforehand. Do not include first name or last name.

***FirstName property***

Optional .Leave empty when a player number has been specified.

***LastName property***

Optional. Required if no player number has been specified. Leave empty when a player number has been specified.

***CountryCode property***














Currently not supported.



## Bridgemate2SettingsDTO





<b>Bm2AutoBoardNumberOption</b>  Enum NoAutoBoardNumber AutoBoardNumber FirstBoardManually	<b>Bm2ContractRepresentationOption</b>  Enum UseSymbols UseLetter	<b>Bm2EnterHandrecordWhenOption</b>  Enum AtEndOfRound AtEndOfBoard DoNotEnter
<b>Bm2NameSource</b>  Enum PlayerNamesTable BMPlayerDB NoExternalSource PlayerNamesTableThenBMPlayerDB	<b>Bm2PairNumberEntryOption</b>  Enum NoEntry Optional Required	<b>Bm2PointScorePerspectiveOption</b>  Enum FromNorthSouth FromDeclarer
<b>Bm2ResultsOverviewOption</b>  Enum FrequencyList6lines1column FrequencyList6lines2columns FrequencyList4lines1column Traveler6lines1column Traveler6lines2columns Traveler4lines1column	<b>Bm2ScoreRecapOption</b>  Enum NoScoreRecap UseScoreRecap AutomaticScoreRecap	<b>BM2ResultEntryMethod</b>  Enum UpDownTricks NumberOfTricks AmericanStyle
<b>Bm2SummaryPointOption</b>  Enum NoSummary Matchpoints Percentage	<b>Bm2NumberValidationOption</b>  Enum NoValidation ACBL FFB ABF JCBL	<b>Bm2ShowPlayerNamesOption</b>  Enum DoNotShow ShowInAllRounds ShowInFirstRoundOnly
		<b>Bm2ShowRankingOption</b>  Enum DoNotShow ShowAllRounds ShowAfterLastRound

The Bridgemate2SettingsDTO contains all settings for the Bridgemate 2. It can be used as part of the [InitDTO](#) for [the initialization of an event](#), or it can be used as data for the [PutBridgemate2Settings command](#) to update settings. When used all settings must be provided. For readability many of the settings are expressed as enum values. This is not necessary, the use of integer values is allowed as well. In the image above the enum values will always be consecutively numbered starting with zero. So Bm2ShowRankingOption.DoNotShow equals to zero and Bm2ShowRankingOption.ShowAfterLastRound equals to two. The settings will be applied to all Bridgemate 2s for a section. The settings must be provided for each section, even if they are the same.

#### Note

In the diagram the Bridgemate2SettingsDTO is depicted as a child of the abstract BridgemateSettingsDTO. The latter's properties can be considered to be part of the Bridgemate2SettingsDTO.

### SessionGuid property

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### SectionLetters property

Required. Identifies the section that the settings pertain to.

### AutopoweroffTime property

Byte. Specifies the number of seconds after which the display will power off.

***AutoShutDownBPC property***

Boolean. Specifies if Bridgemate Control Software should shut down after the last result has been processed.

***BM2AutoBoardNumber property***

Integer/Enum (Bm2AutoBoardNumberOption). Specifies if the Bridgemate should fill in the boardnumber automatically. Also see the BM2FirstBoardManually setting.

***BM2AutoShowScoreRecap property***

Boolean. Specifies if the scores obtained should be displayed for review after the last result has been entered for each round. All obtained results will be shown as well after the play of the last board.

***BM2ConfirmNP property***

Boolean. Specifies if the TD must come to the table to confirm No play on the Bridgemate.

***BM2EnterHandrecord property***

Boolean. Specifies if the players can enter the handrecord for a board. If "true" then the BM2EnterHandrecordWhen setting determines when the handrecord may be entered.

***BM2EnterHandrecordWhen property***

Integer/Enum (Bm2EnterHandrecordWhenOption). Specifies when the players can enter the handrecord on the Bridgemate. The default value is zero: after all boards for the round have been played.

***BM2FirstBoardManually property***

Boolean. If the BM2AutoBoardNumber setting is "True" this setting further specifies if the board number for the first board played should be entered manually.

***BM2GameSummary property***

Boolean. If "True" shows the summary of the session after the last board has been entered.

***BM2NameSource property***

Currently not supported. Player names for at least all competing players should be sent to the Data Connector using [PlayerDataDTOs](#).

***BM2NextSeating property***

Boolean. Specifies if the Bridgemate should show the table for the next round after the last result has been entered for the current round.

***BM2NumberEntryEachRound property***

Boolean. If "True" the players must enter their player number at the start of each round.

***BM2NumberEntryPreloadValues property***

Boolean. If player numbers are entered each round, preload known player numbers.

***BM2PairNumberEntry property***

Integer/Enum (Bm2PairNumberEntryOption). Specifies if when entering a result the declaring pair must be entered.

***BM2PINcode property***

String of four digits. Optional. Defaults to "0000".

***BM2Ranking property***

Integer/Enum (Bm2ShowRankingOption). Specifies if and when the current ranking for the pairs should be shown.

***BM2RecordBidding property***

Boolean. Currently not supported.

***BM2RecordPlay property***

Boolean. Currently not supported.

***BM2RemainingBoards property***

Boolean. If "True" the Bridgemate will show how many boards remain to be played.

***BM2ResetFunctionKey property***

Boolean. If "True" the reset function key will be available from the TD menu without having to enter the PIN code first.

***BM2ResultsOverview property***

Integer/Enum (Bm2ResultsOverviewOption). Specifies how the previous results on the board should be shown. Either as frequency list (show number of times a specific contract was played) or as traveler (show the result for each pair).

***BM2ScoreCorrection property***

Boolean. If "True" allows the players to erase a result in the round on their table and to enter it again.

***BM2ScoreRecap property***

Boolean. If "True" the Bridgemate will show a "Scores" button to check the entered results for the round.

***BM2ShowHands property***

Boolean. Currently not supported.

***BM2ShowPlayerNames property***

Integer/Enum (Bm2ShowPlayerNamesOption). Specifies if and when the playernames should be shown on the Bridgemate at the start of a round.

***BM2SummaryPoints property***

Integer/Enum (Bm2SummaryPointsOption). If the BM2GameSummary is "True" specifies whether the obtained results per board should be shown as matchpoints or as a percentag. For this to work the BM2RankingProperty must be either ShowAllRounds (1) or ShowAfterLastRound (2).

***BM2TDCall property***

Boolean. If "True" specifies that the players can call the Tournament Director from the Bridgemate.

***BM2TextBasedNumber property***

Currently not supported. Player numbers can be set using the [PlayerDataDTO](#). The player number is stored as a string.

***BM2ValidateLeadCard property***

Boolean. If "True" The entered lead card will be checked against the handrecord.

***BM2ValidateRecording property***

Currently not supported.

***BM2ViewHandRecord property***

Boolean. If "true" the players get the option to view the handrecord after entering the result.

***BoardOrderVerification property***

Boolean. If "True" the Bridgemate will check the order of entry of the boardnumbers.

***EnterResultsMethod property***

Integer/Enum (Bm2ResultsEntryOption). Specifies how the resulting tricks after play must be entered.

***GroupSections property***

Not supported. Use the [ScoringGroupDTO](#) to indicate that sections should be scored together.

***HandRecordValidation property***

Boolean. Currently not supported.

***Intermediate results property***

Boolean. Currently not supported.

***LeadCard property***

Boolean. If "True" specifies that the leadcard must be entered along with the contract.

***MaximumResults property***

Integer (0-127). Specifies the maximum number of results to show. Zero means "unlimited".

***MemberNumber property***

Boolean. If "True" the Bridgemate will ask for playernumbers at the start of a round when for that round there are participations without a player number.

***MemberNumbersNoBlankEntry property***

Boolean If "True" entry of player numbers cannot be skipped,

***NumberValidation property***

Currently not supported.

***RepeatResults property***

Boolean. If "True" allows the players to review the results of a round again.

***ScorePoints property***

Integer/Enum (Bm2PointScorePerspectiveOption). Show score points from perspective of North-South or from declarer.

***ShowContract property***

Integer/Enum (Bm2ContractRepresentationOption). Specifies if the denomination of the contract should be shown with letters or symbols.

***ShowOwnResult property***

Boolean. If "True" specifies that the own result should be included in the result overview.

***ShowPairNumbers property***

Boolean. If "True" the pair numbers for the current round will be displayed on the Bridgemate.

***ShowPercentage property***

Boolean. If "True" show the percentage obtained on the board just played.

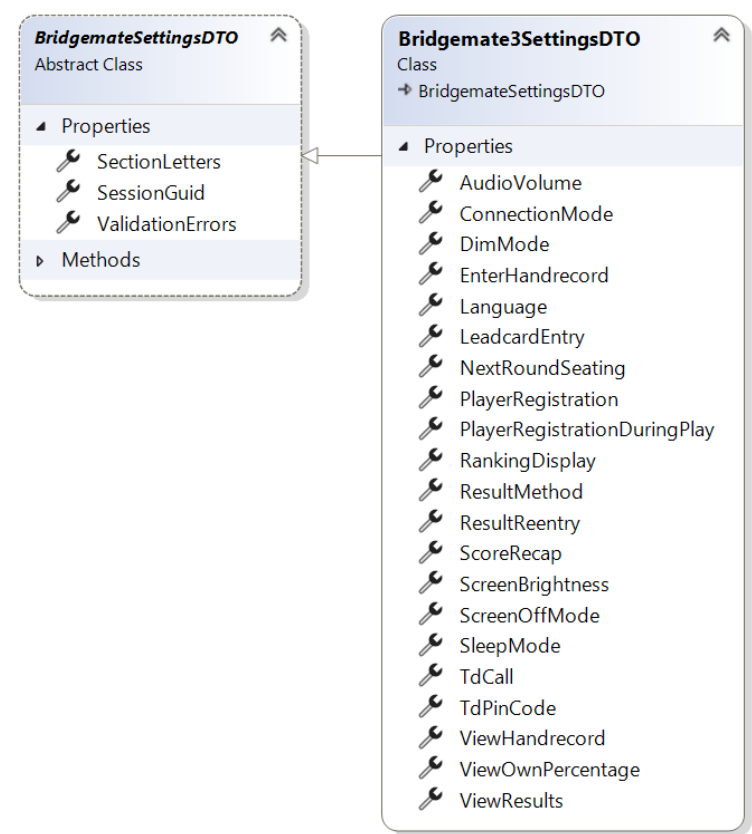
***ShowResults property***

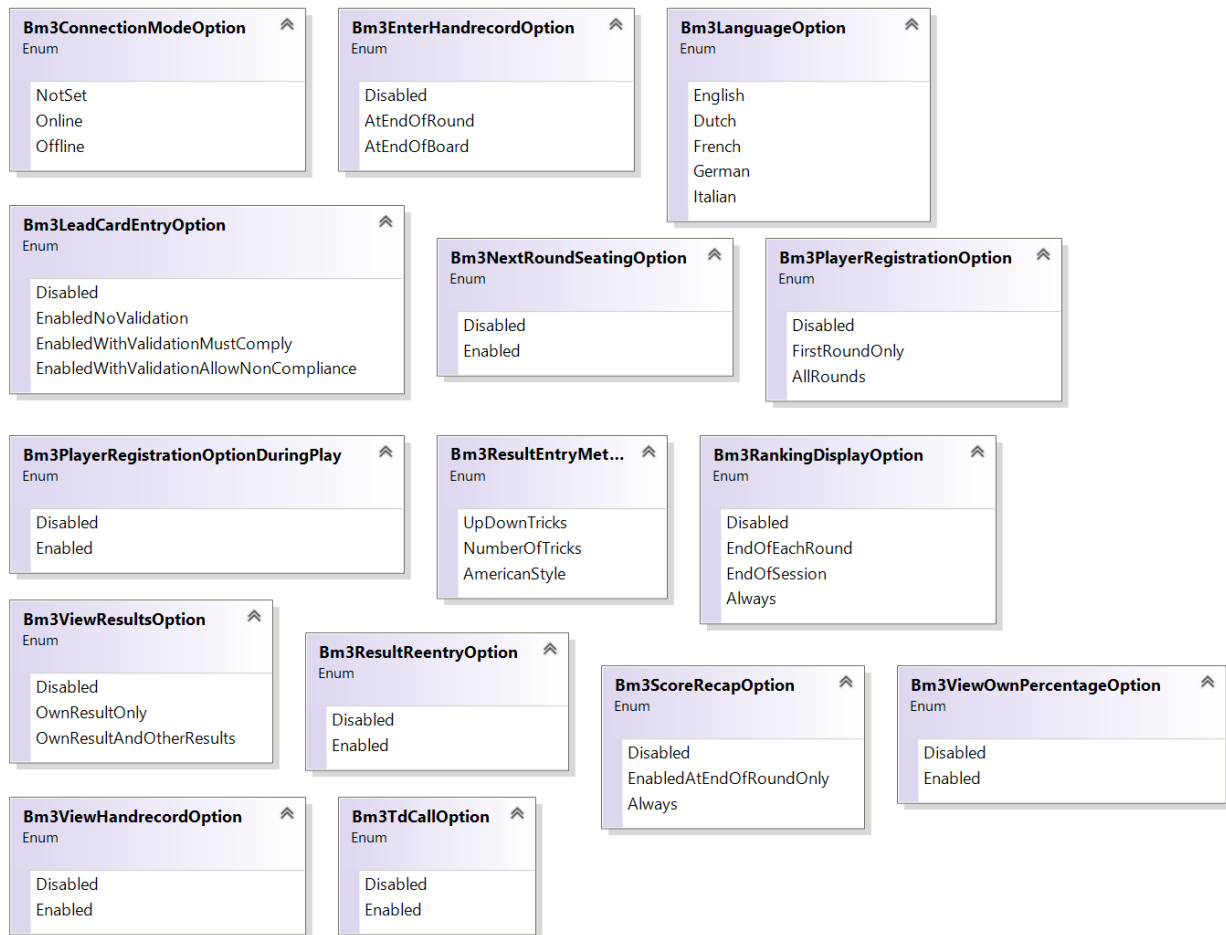
Boolean. If "True" show previous results on the board just played.

***VerificationTime property***

Integer (1-7) The time in seconds that the verification message is shown before the opponents get to confirm the result on the board just played.

# Bridgemate3SettingsDTO





The Bridgemate3SettingsDTO contains all settings for the Bridgemate 3. It can be used as part of the [InitDTO](#) for [the initialization of an event](#), or it can be used as data for the [PutBridgemate3Settings command](#) to update settings. When used all settings must be provided. For readability many of the settings are expressed as enum values. This is not necessary, the use of integer values is allowed as well. In the image above the enum values will always be consecutively numbered starting with zero. So Bm3ViewResultsOption.Disabled equals to zero and Bm3ViewResultsOption.OwnResultsAndOtherResults equals to two. The settings will be applied to all Bridgemate 3s for a section. The settings must be provided for each section, even if they are the same.

#### Note

In the diagram the Bridgemate3SettingsDTO is depicted as a child of the abstract BridgemateSettingsDTO. The latter's properties can be considered to be part of the Bridgemate3SettingsDTO.

### SessionGuid property

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### SectionLetters property

Required. Identifies the section that the settings pertain to.

### AudioVolume property

Integer (0-7). Currently not in use.

### ConnectionMode property

Integer/Enum (Bm3ConnectionModeOption). Specifies if the Bridgemates for this section will operate in offline mode. In offline mode the Bridgemate will not communicate with the Bridgemate 3 server during the session, but will try to send its results after the last board had been played.

***DimMode property***

Integer (0-15). Defines the number of seconds (times 5) before the screen of the Bridgemate will dim. So the default value of 2 amounts to 10 seconds.

***EnterHandrecord property***

Integer/Enum (BM3EnterHandrecordOption). Specifies if and when the players can enter the handrecord for a board if it is not present.

***Language property***

Integer/Enum (BM3LanguageOption). Specifies which language the Bridgemate will use. The default is zero for English.

***LeadCardEntry property***

Integer/Enum (BM3LeadCardEntryOption). Specifies if the leadcard must be entered together with the contract and if the leadcard should be validated against the handrecord.

***NextRoundSeating property***

Integer/Enum (BM3NextRoundSeatingOption). Specifies if the seatings for the next round must be displayed after the last result has been entered for the round.

***PlayerRegistration property***

Integer/Enum (BM3PlayerRegistrationOption). Specifies if and when the players can make themselves known by entering their playernumber and/or name on the Bridgemate.

***PlayerRegistrationDuringPlay property***

Integer/Enum (BM3PlayerRegistrationOptionDuringPlay). Specifies if and when the players can make themselves known by entering their playernumber and/or name on the Bridgemate while the round is in progress (i.e.: after a board has been entered).

***RankingDisplay property***

Integer/Enum (BM3RankingDisplayOption). Specifies if and when the Bridgemate will show the current ranking for the players on the table.

***ResultMethod property***

Integer/Enum (BM3ResultMethodOption). Specifies how players can enter the result for a board: Up/Down tricks, total tricks or American style.

***ResultReentry property***

Integer/Enum (BM3ResultReentryOption). Specifies if players may reenter the result of the board after the original entry has been confirmed by the opponents.

***ScoreRecap property***

Integer/Enum (BM3ScoreRecapOption). Specifies if and when the Bridgemate will show a recap of the scores obtained on the table in the current round.

***ScreenBrightness property***

Integer (1-7) Defines the screen brightness.

***ScreenOffMode property***

Integer (0-15). Defines the number of seconds (times 5) before the screen of the Bridgemate will turn off. A value of zero, the default, means "never turn off".

***SleepMode property***

Integer (0-120). Defines the number of seconds (times 5) before the Bridgemate will enter Sleepmode. Once the Bridgemate has entered sleepmode its powerbutton must be used to wake it up again.

***TdCall property***

Integer/Enum (BM3TdCallOption). Specifies if players can call for the Tournament Director using the Bridgemate.

***TdPinCode property***

A string of four digits. The code the TD must enter to get access to the TD menu. Defaults to "0000".

***ViewHandrecord property***

Integer/Enum (BM3ViewHandrecordOption). Specifies if players may view the handrecord after the result has been entered.

***ViewOwnPercentage property***

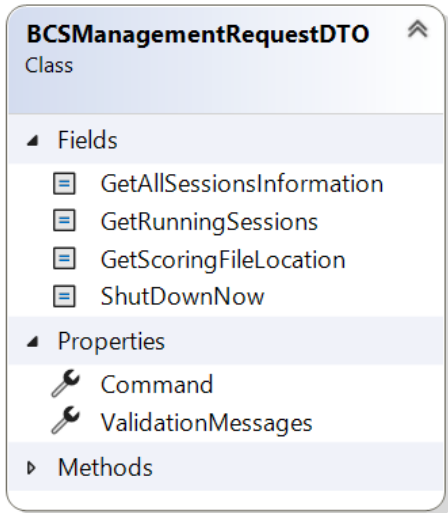
Integer/Enum (BM3ViewHandrecordOption). Specifies if the own percentage should be included in the results overview.

***ViewResults property***

Integer/Enum (BM3ViewResultsOption). Specifies if the Bridgemate will show the other results for the board that was just entered.



## BCSManagementRequestDTO

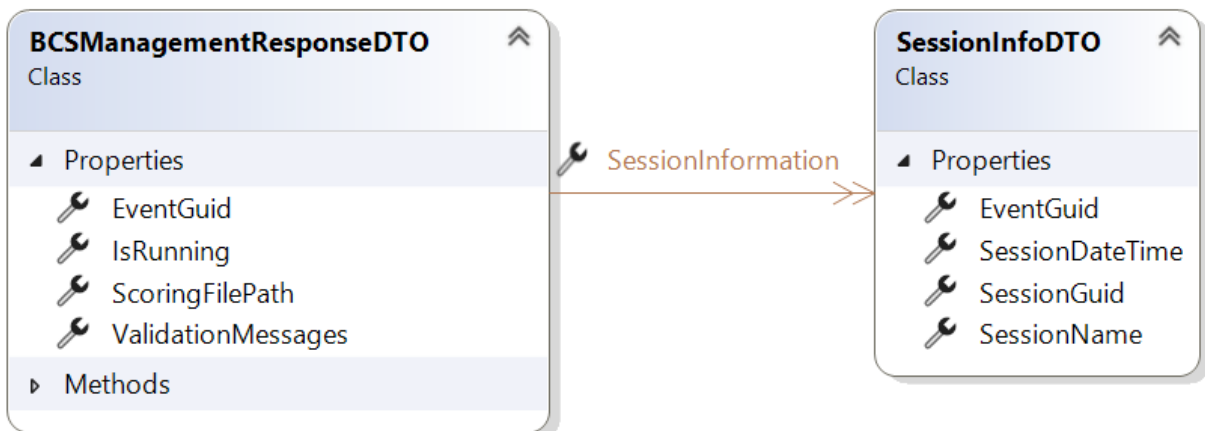


The BCSManagementDTO can be used to send a shut down request to Bridgemate Control Software. It can also be used to query some data about what events are known and which of them currently is being administered. Last it is possible to ask for the location of the scoring file in order to include this in back-ups. Depending on the given command the request will return different data. The BCSManagementRequestDTO is sent with a [ManageBCS command](#)

Below are the commands that can be given and their effects:

Command	Effect	Returns
1: ShutDownNow	Sends a request to BCS to shut down immediately, Mind that this can cause previous data that was sent not to be processed.	A ScoringProgramResponse with its DataType set to OK or an error code. No serialized data is returned.
2: GetRunningSessions	Queries which event is currently being administered.	A ScoringProgramResponse with as DataType EventInfo or an errorcode. The SerializedData property contains a <a href="#">BCSManagementResponseDTO</a> .
4: GetScoringFileLocation	Queries which scoring file is currently in use and where it is located.	A ScoringProgramResponse with as DataType ScoringFileLocation or an errorcode. The SerializedData property contains a <a href="#">BCSManagementResponseDTO</a> .
8: GetAllSessionsInformation	Queries which events are present in the current scoring file.	A ScoringProgramResponse with as DataType AllSessionsInfo or an errorcode. The SerializedData property contains a <a href="#">BCSManagementResponseDTO</a> .
6: 2+4	Combines the two commands	A ScoringProgramResponse with as DataType EventInfo or an errorcode. The SerializedData property contains a <a href="#">BCSManagementResponseDTO</a> .

## BCSManagementResponseDTO



### The *BCSManagementResponseDTO*

The *BCSManagementResponseDTO* is returned for a [BCSManagementCommand](#) that asks for session information and/or the scoringfile path.

#### *EventGuid property*

The guid of the current event being administered by Bridgemate Control Software. Will be empty if *IsRunning* is "False" or when the [BCSManagementRequestDTO.Command](#) property contains *GetAllSessionsInformation* (8).

#### *IsRunning property*

If "True" BCS is running.

#### *ScoringFilePath property*

The full path to BCS's scoring file. Can be used to make and restore back-ups.

### The *SessionInfoDTO*

The *SessionInfoDTO* contains either information on the sessions that BCS currently administers ([BCSManagementRequestDTO.Command](#) contains *GetRunningSessions* (2) or information on all known sessions in the scoring file ([BCSManagementRequestDTO.Command](#) contains *GetAllSessionsInformation* (8) . It will only contain data when *IsRunning* is "True".

#### *EventGuid property*

The guid of the event the session belongs to.

#### *SessionDateTime property*

The date and time when the session started,

#### *SessionGuid property*

The guid of the session.

#### *SessionName property*

The name of the session.

## HandrecordDTO

**HandrecordDTO**  
 Class

▶ Fields

◀ Properties

BoardNumber  
 DoubleDummyEastClubs  
 DoubleDummyEastDiamonds  
 DoubleDummyEastHearts  
 DoubleDummyEastNoTrump  
 DoubleDummyEastSpades  
 DoubleDummyNorthClubs  
 DoubleDummyNorthDiamonds  
 DoubleDummyNorthHearts  
 DoubleDummyNorthNoTrump  
 DoubleDummyNorthSpades  
 DoubleDummySouthClubs  
 DoubleDummySouthDiamonds  
 DoubleDummySouthHearts  
 DoubleDummySouthNoTrump  
 DoubleDummySouthSpades  
 DoubleDummyWestClubs  
 DoubleDummyWestDiamonds  
 DoubleDummyWestHearts  
 DoubleDummyWestNoTrump  
 DoubleDummyWestSpades  
 EastClubs

EastClubs  
 EastDiamonds  
 EastHearts  
 EastSpades  
 HasHandEvaluationData  
 NorthClubs  
 NorthDiamonds  
 NorthHearts  
 NorthSpades  
 ScoringGroupNumber  
 SectionLetters  
 SessionGuid  
 SouthClubs  
 SouthDiamonds  
 SouthHearts  
 SouthSpades  
 WestClubs  
 WestDiamonds  
 WestHearts  
 WestSpades

▶ Methods

The HandrecordDTO contains the cards of each hand for a [ScoringGroupDTO](#). Moreover it can contain double dummy data,

HandrecordDTOs can be sent as part of the [InitDTO](#) when [initializing a new event](#). Or they can be sent separately as data for the [PutHandrecordsCommand](#).

### Note

The handrecords are identified using the SessionGuid and Sectionletters properties, However, handrecords must be unique for each scoringgroup. When a scoringgroup has more than one section it suffices to send the handrecords for only one of these sections.

### SessionGuid property

Required. A guid uniquely defining the session. Must be exactly 32 character long, uppercase and cannot contain dashes or curly braces.

### SectionLetters property

Required. Together with the SessionGuid, the ScoringGroupNumber and BoardNumber it uniquely defines the handrecord.co

### ScoringGroupNumber property

Required. Together with the SessionGuid, SectionLetters and BoardNumber it uniquely defines the handrecord.

***BoardNumber property***

Required. Together with the SessionGuid, ScoringGroupNumber and SectionLetters it uniquely defines the hand record.

***NorthClubs...WestSpades properties***

Required. Strings that define the cards for a hand. Valid values are A K Q J T 9 8 7 6 5 4 3 2. Leave empty for a void. Note that the ten must be represented as 'T'.

***HasHandEvaluationData property***

Optional. Specifies if double dummy analysis data is included.

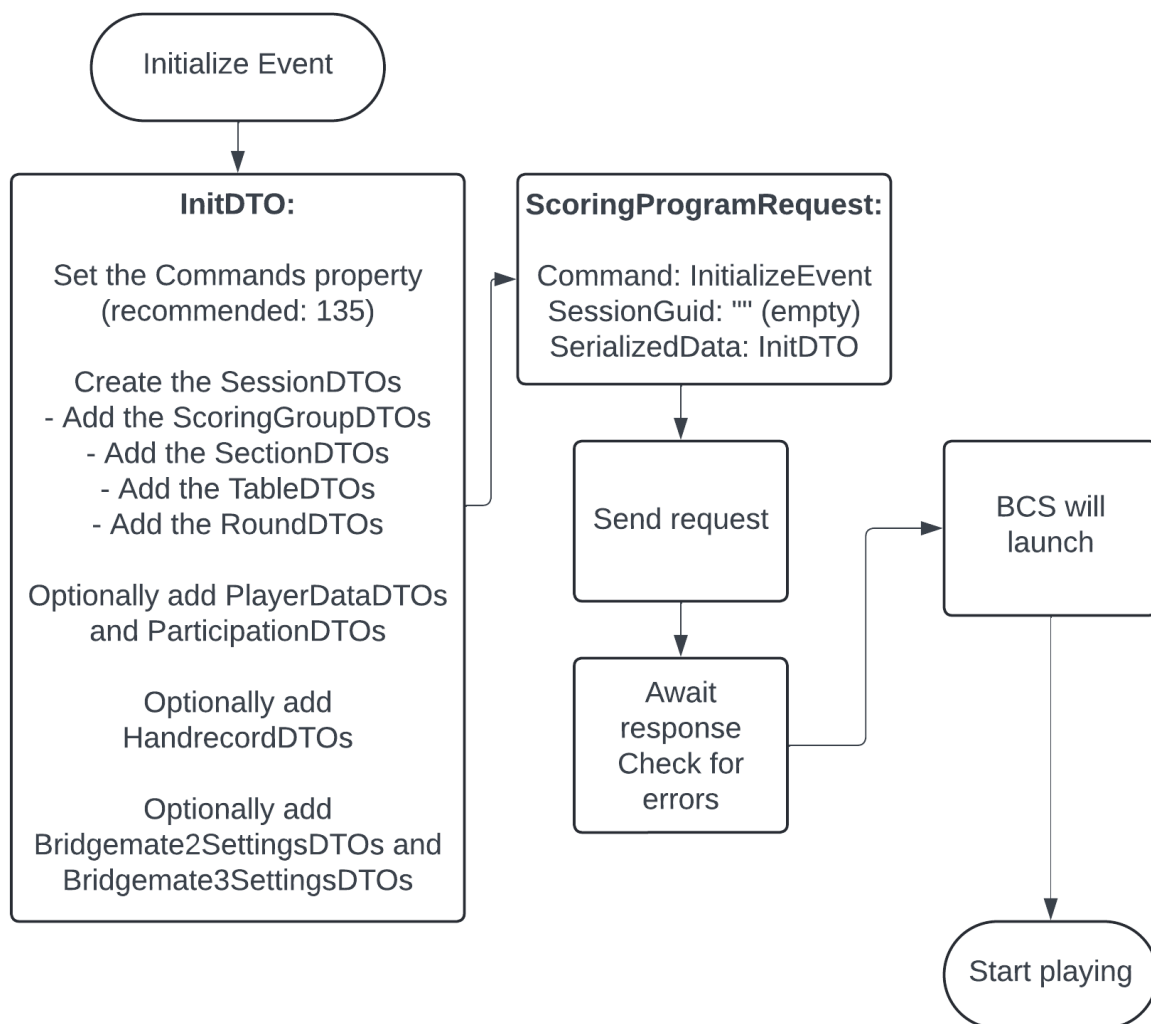
***DoubleDummyNorthClubs...DoubleDummyWestNoTrump properties***

Optional. Specifies the total number of tricks that can be made given the declarer and denomination.

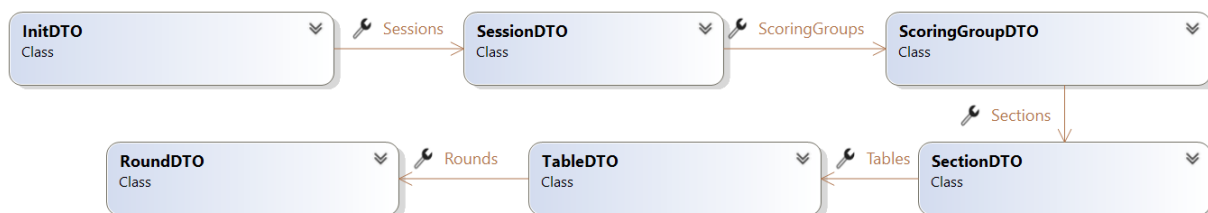
## Procedures

### Initialize an event

#### Procedure



#### Data structure



#### Description

An event can be initialized with one or more sessions. Initialization is done using the [InitializeEvent](#) command and an [InitDTO](#). The SessionGuid must not be set in the [ScoringProgramRequest](#). This DTO

must at least contain information on the scoring method, the scoring group for each section and the movement.

Player data, starting positions and handrecords are optional. Board results are no part of initialization data but can be sent later.

Furthermore, the InitDTO contains a [Commands](#) property that specifies which actions the Bridgemate Control Software must undertake when it has been launched.

## Example json code

The code below shows the minimum required json data to send to initialize a session with four pairs that all meet each other.

```
{
  "Command": 5,
  "SessionGuid": "",
  "SerializedData":
  {
    "Commands": 135,
    "EventGuid": "6D115AF1ABEB4462A299B1FE86274949",
    "AlternativeDataFolder": null,
    "Sessions": [
      {
        "ScoringGroups": [
          {
            "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
            "Sections": [
              {
                "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
                "ScoringGroupNumber": 1,
                "Letters": "A",
                "Name": null,
                "Winners": 1,
                "GameType": 10,
                "EWMoveBeforePlay": 0,
                "MissingPair": 0,
                "IsCombiSection": false,
                "NorthSouthPairSectionLetters": null,
                "EastWestPairSectionLetters": null,
                "IsDeleted": false,
                "Tables": [
                  {
                    "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
                    "SectionLetters": "A",
                    "TableNumber": 1,
                    "Rounds": [
                      {
                        "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
                        "SectionLetters": "A",
                        "TableNumber": 1,
                        "RoundNumber": 1,
                        "LowBoardNumber": 1,
                        "HighBoardNumber": 4,
                        "PairNS": 1,
                        "PairEW": 2,
                        "TeamNS": 0,
                        "TeamEW": 0,
                        "MatesTableSectionLetters": null,
                        "MatesTableTableNumber": 0,
                        "MatesTableRoundNumber": 0,
                        "Updated": false
                      },
                      {
                        "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
                        "SectionLetters": "A",
```

```

    "TableNumber": 1,
    "RoundNumber": 2,
    "LowBoardNumber": 5,
    "HighBoardNumber": 8,
    "PairNS": 1,
    "PairEW": 3,
    "TeamNS": 0,
    "TeamEW": 0,
    "MatesTableSectionLetters": null,
    "MatesTableTableNumber": 0,
    "MatesTableRoundNumber": 0,
    "Updated": false
  },
  {
    "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
    "SectionLetters": "A",
    "TableNumber": 1,
    "RoundNumber": 3,
    "LowBoardNumber": 9,
    "HighBoardNumber": 12,
    "PairNS": 1,
    "PairEW": 4,
    "TeamNS": 0,
    "TeamEW": 0,
    "MatesTableSectionLetters": null,
    "MatesTableTableNumber": 0,
    "MatesTableRoundNumber": 0,
    "Updated": false
  }
]
},
{
  "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
  "SectionLetters": "A",
  "TableNumber": 2,
  "Rounds": [
    {
      "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
      "SectionLetters": "A",
      "TableNumber": 2,
      "RoundNumber": 1,
      "LowBoardNumber": 1,
      "HighBoardNumber": 4,
      "PairNS": 3,
      "PairEW": 4,
      "TeamNS": 0,
      "TeamEW": 0,
      "MatesTableSectionLetters": null,
      "MatesTableTableNumber": 0,
      "MatesTableRoundNumber": 0,
      "Updated": false
    },
    {
      "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
      "SectionLetters": "A",
      "TableNumber": 2,
      "RoundNumber": 2,
      "LowBoardNumber": 5,
      "HighBoardNumber": 8,
      "PairNS": 4,
      "PairEW": 2,
      "TeamNS": 0,
    }
  ]
}

```

```

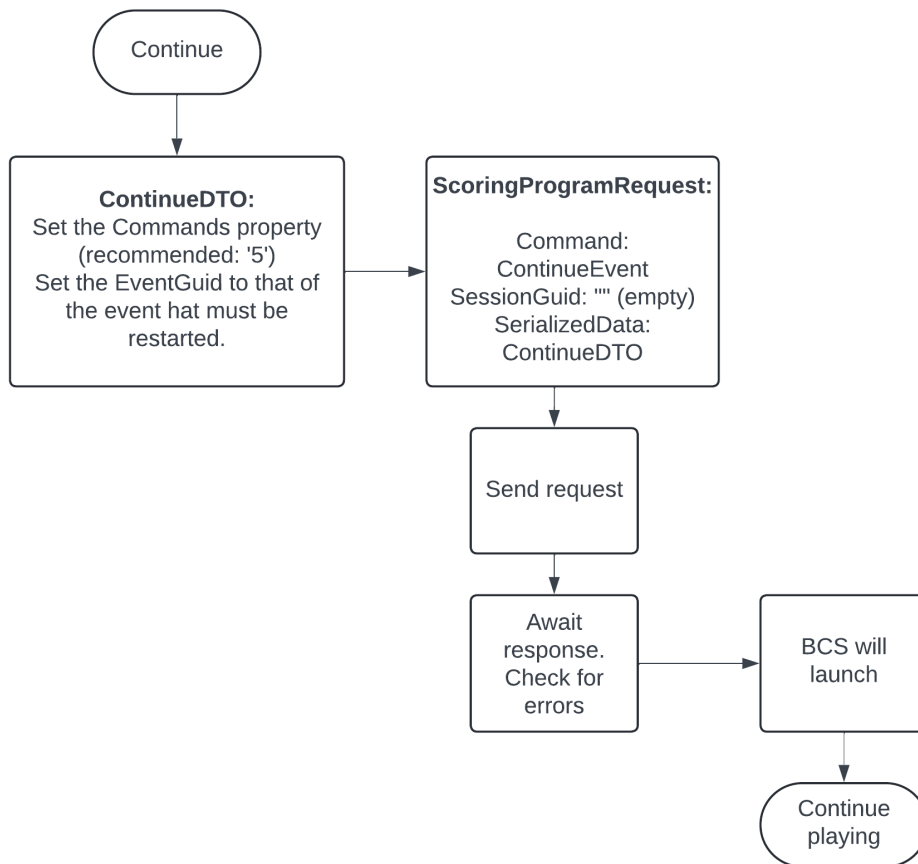
    "TeamEW": 0,
    "MatesTableSectionLetters": null,
    "MatesTableTableNumber": 0,
    "MatesTableRoundNumber": 0,
    "Updated": false
  },
  {
    "SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
    "SectionLetters": "A",
    "TableNumber": 2,
    "RoundNumber": 3,
    "LowBoardNumber": 9,
    "HighBoardNumber": 12,
    "PairNS": 2,
    "PairEW": 3,
    "TeamNS": 0,
    "TeamEW": 0,
    "MatesTableSectionLetters": null,
    "MatesTableTableNumber": 0,
    "MatesTableRoundNumber": 0,
    "Updated": false
  }
]
}
],
"ScoringGroupNumber": 1,
"ScoringMethod": 10,
"Name": null,
"IsDeleted": false
}
],
"EventGuid": "6D115AF1ABEB4462A299B1FE86274949",
"SessionGuid": "6D115AF1ABEB4462A299B1FE86274949",
"Name": "Minimal Session",
"Year": 2024,
"Month": 6,
"Day": 3,
"Hour": 0,
"Minute": 0,
"ShowInApp": true,
"EWReturnHome": false,
"PairsMoveAccrossField": false
}
],
"PlayerData": null,
"Participations": null,
"Handrecords": null,
"Bridgemate2Settings": null,
"Bridgemate3Settings": null
}
}

```

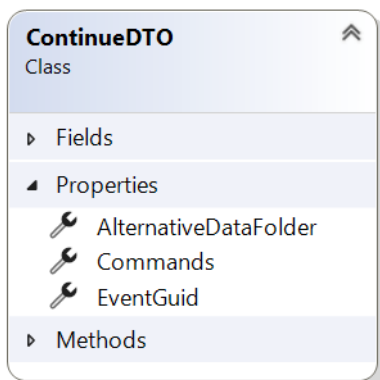


## Continue an event

### Procedure



### Data structure



### Description

To continue a previously initialized event with its sessions use the [ContinueEvent](#) command and a [ContinueDTO](#). The SessionGuid must not be set in the [ScoringProgramRequest](#). This DTO must contain a valid EventGuid.

Furthermore, the ContinueDTO contains a [Commands](#) property that specifies which actions the Bridgemate Control Software must undertake when it has been launched.

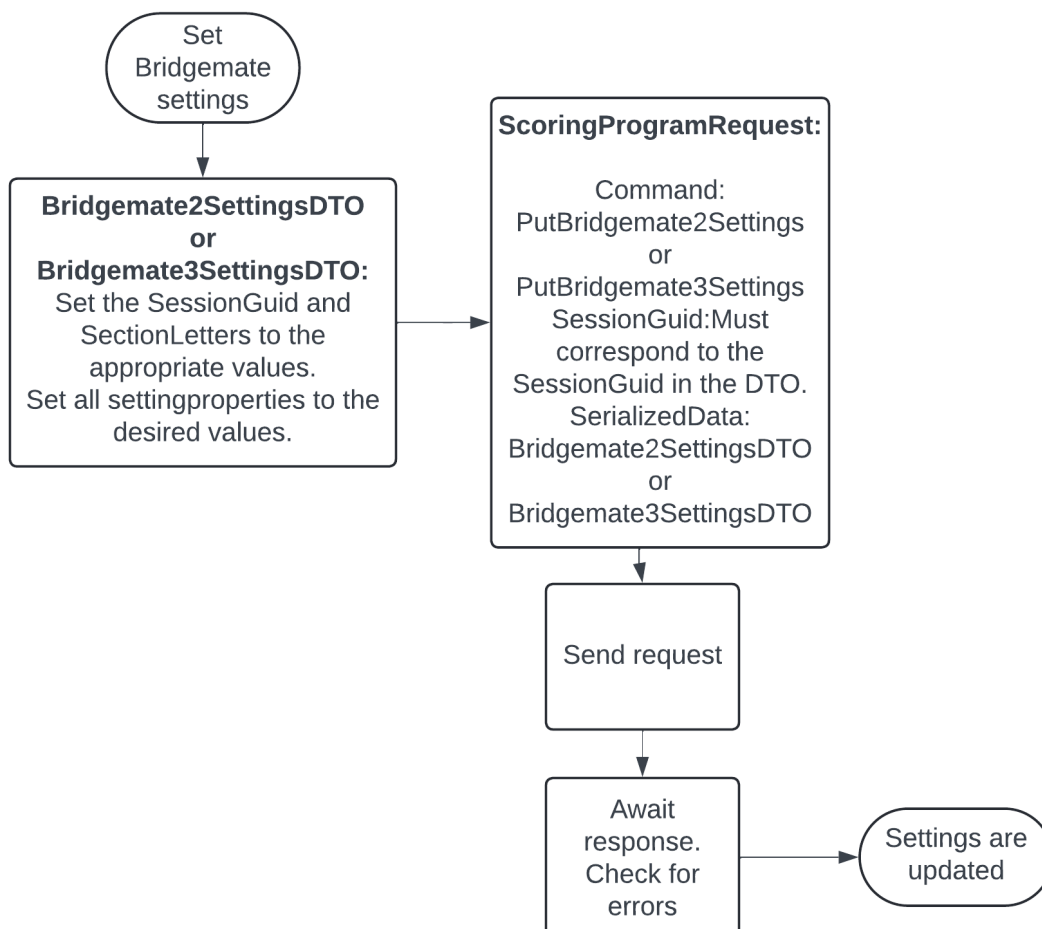
## Example json code

The code below shows json data that needs to be sent to continue a previously initialized event.

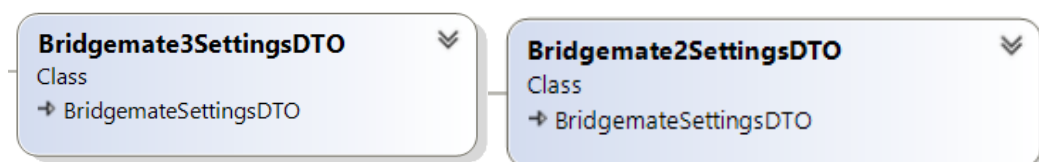
```
{
  "Command":30,
  "SessionGuid":"",
  "SerializedData":
  {
    "EventGuid": "6D115AF1ABEB4462A299B1FE86274949",
    "Commands": 5,
    "AlternativeDataFolder": null
  }
}
```

## Update Bridgmate settings

### Procedure



### Data structure



### Description

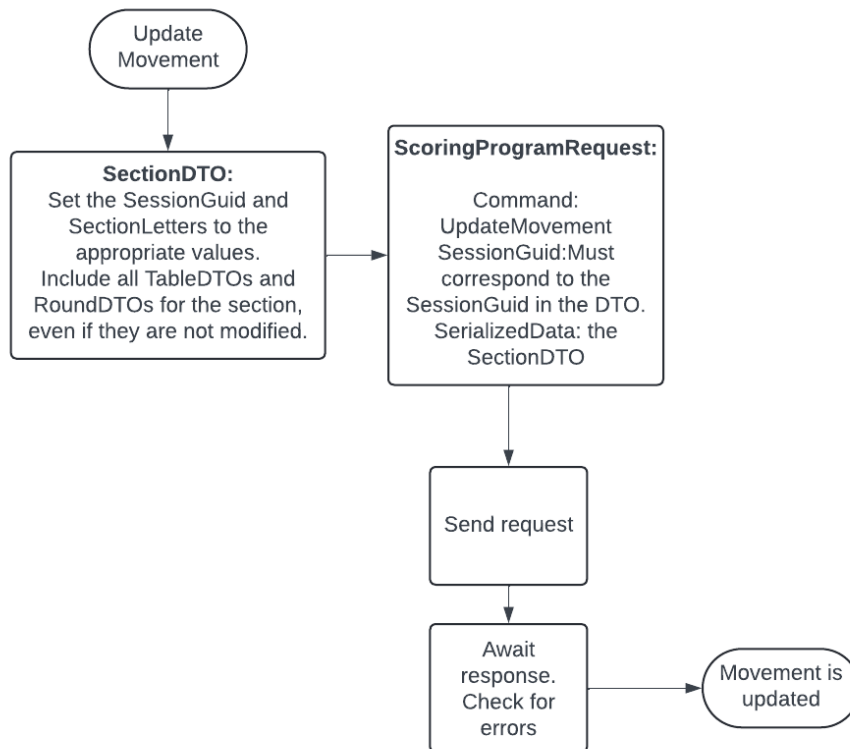
Bridgmate settings can be included into the [InitDTO](#) when [initializing the event](#), or can be updated later using the [PutBridgmate2Settings command](#) or the [PutBridgmate3Settings command](#). These commands use the [Bridgmate2SettingsDTO](#) and [Bridgmate3SettingsDTO](#) respectively. The settings must be updated for all sections in the event, even if they are the same.

### Example json code

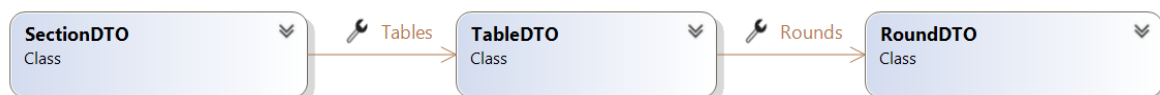
The code below shows json data that needs to be sent to update the Bridgmate 3 settings for a section.

Updating the Bridgемate 2 settings will have a similar structure, but with different setting properties.

## Update the movement for a section



## Data structure



## Description

### Update the movement for a section

When after the play of the session has started the need arises to adjust the movements, the movement for a section can be updated using a [SectionDTO](#) and the [UpdateMovementCommand](#). Reasons for such an update could be a pair arriving late or playing a Swiss pairs event, The update is communicated to the Bridgemate Data Connector by sending a SectionDTO that contains the new movement data for the section. On reception of this data BCS will figure out how to update the section. Movements are expressed by adding the [tables](#) and their [rounds](#) to the section.

The ScoringProgramResponse, if successful, contains a SectionDTO in its SerializedData property that specifies which rounds have been updated: Each round below the first specified round on a table is unchanged, each round above the last specified round will be deleted.

### Adding a section

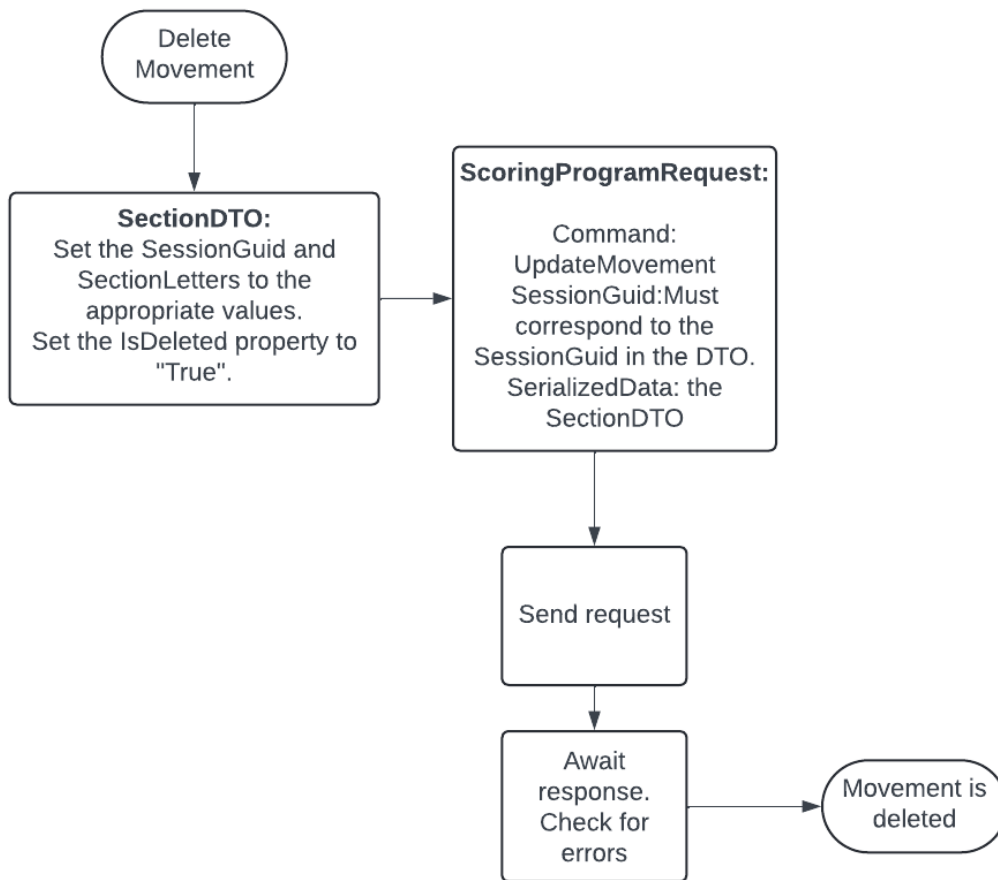
Adding a section works the same way. The Bridgemate Data Connector will detect that the section did not exist before and will add it. [But be sure to update the scoring groups first if the new section has its own scoring group.](#)

## Note

- Do not forget to send PlayerDataDTOs and ParticipationDTOs for new players that have been added to the event.
- Boardresults will be deleted from the first round where the round data for a table has changed. The

ScoringProgramResponse contains the rounds (as a SectionDTO) for which this is the case. If applicable send the results for these rounds again (but as a rule it should not apply).

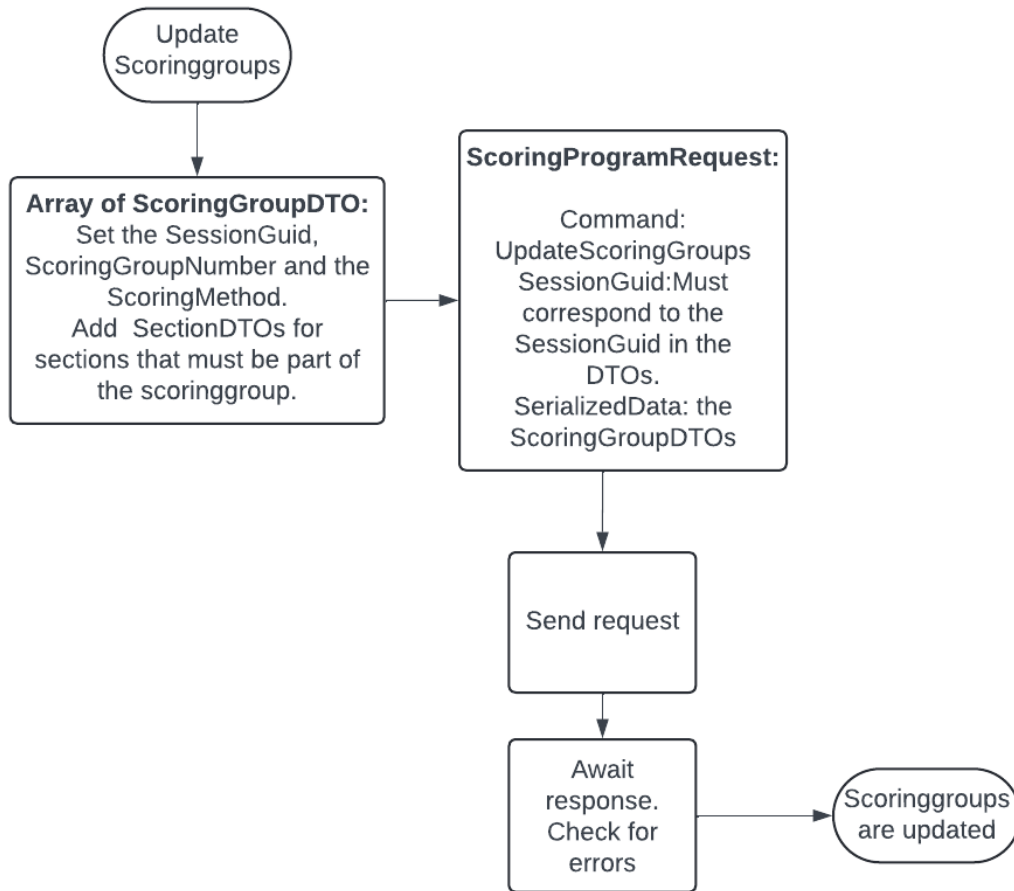
## Delete a section



### Description

To delete a section Send a [SectionDTO](#) with its IsDeleted property set to "True" using the [UpdateMovement command](#).

## Update scoringgroups



## Data structure

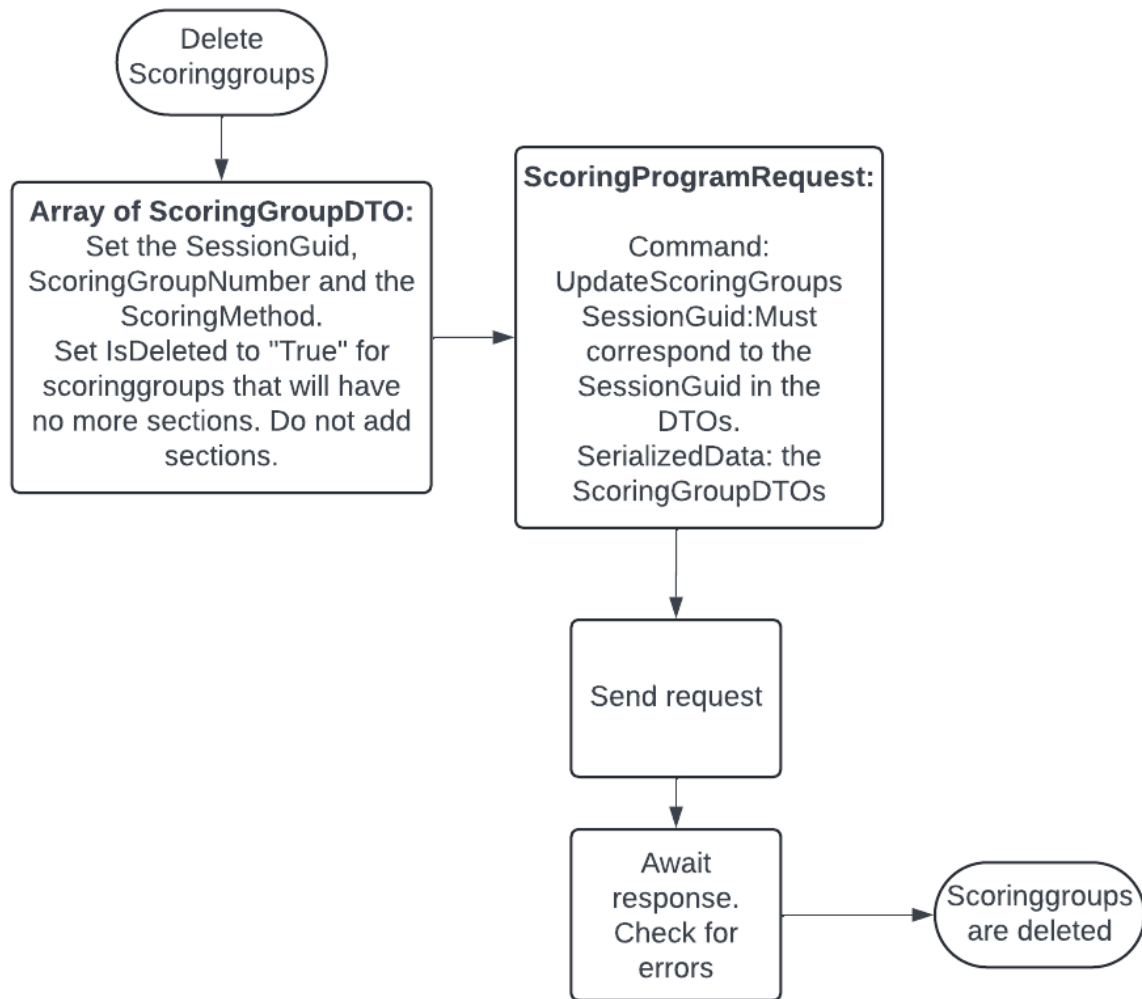


## Description

The configuration of the scoringgroups can be upated using a [ScoringGroupDTO](#) and attaching the [SectionDTOs](#) to it that should become part of it. Send the ScoringGroupDTOs using the [UpdateScoringGroups](#) command. Mind that each section must have a scoringgroup, so take care that no section is without one. Existing scoringgroups will be updated, the rest will be added.



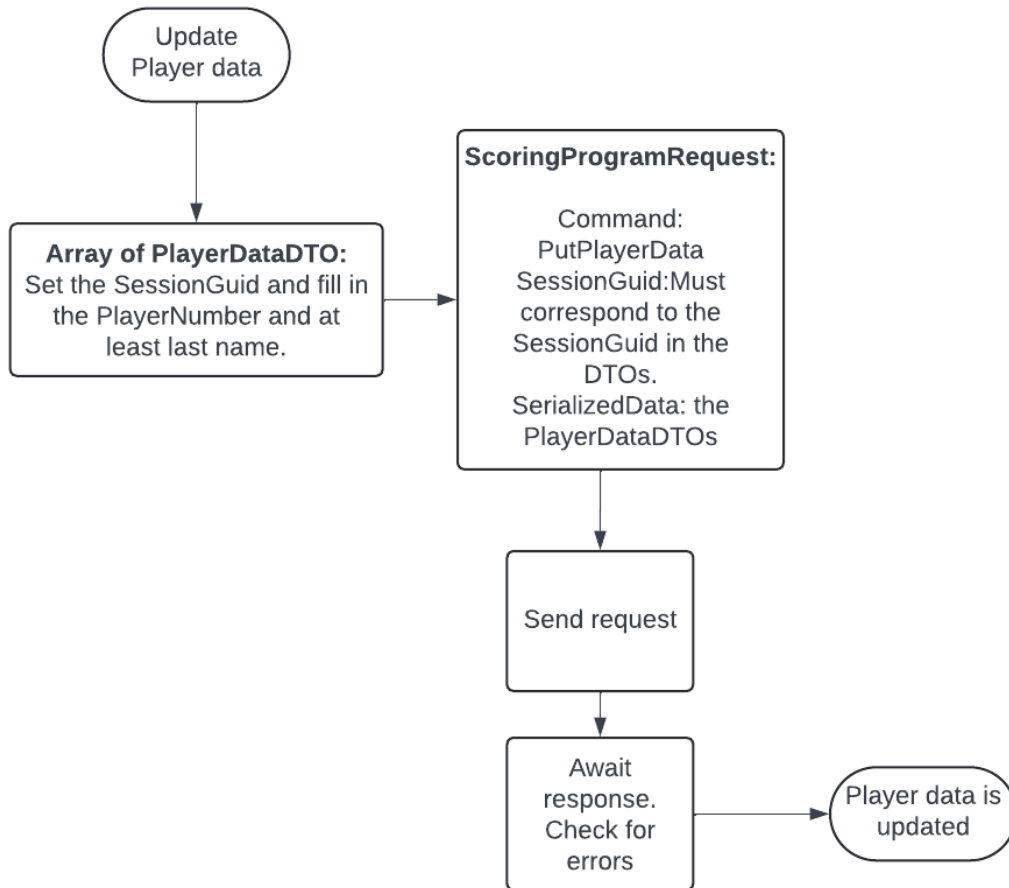
## Delete scoringgroups



### Description

To delete scoringgroups send [ScoringGroupDTOs](#) with their IsDeleted property set to "True" using a UpdateScoringGroupsCommand. Do not add any sections to the scoringgroups. A scoringgroup cannot be deleted while it has sections attached to it. [Update the scoringgroups first to assign the sections of the to be deleted scoringgroup an existing or new scoringgroup.](#)

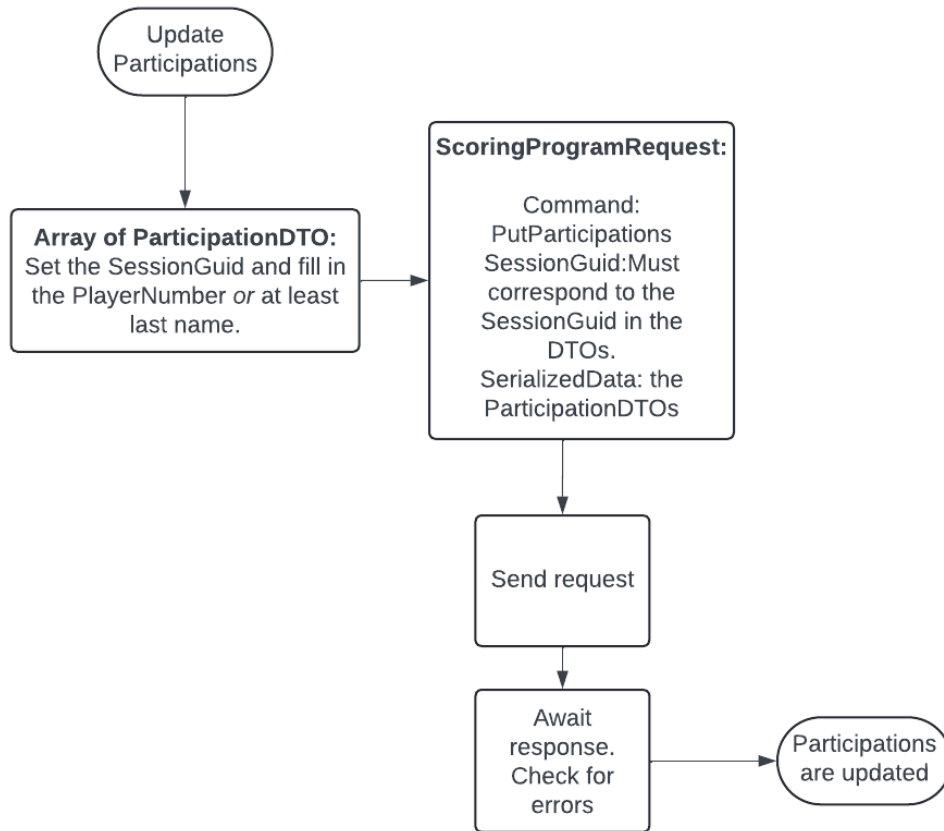
## Update player data



### Description

Player data can be updated with an array of [PlayerDataDTOs](#) using the [PutPlayerData command](#). Existing player data will be updated, enabling changing first name, last name and country code, the rest will be added. Each ParticipationDTO that is sent to the Bridgemate Data Connector with its SessionGuid and PlayerNumber properties set must have a corresponding PlayerData that was sent before. Player data preferably is sent with the [InitDTO](#) while [initializing the event](#); this is more performant. Use the updates for movement changes after the event has started.

## Update participations



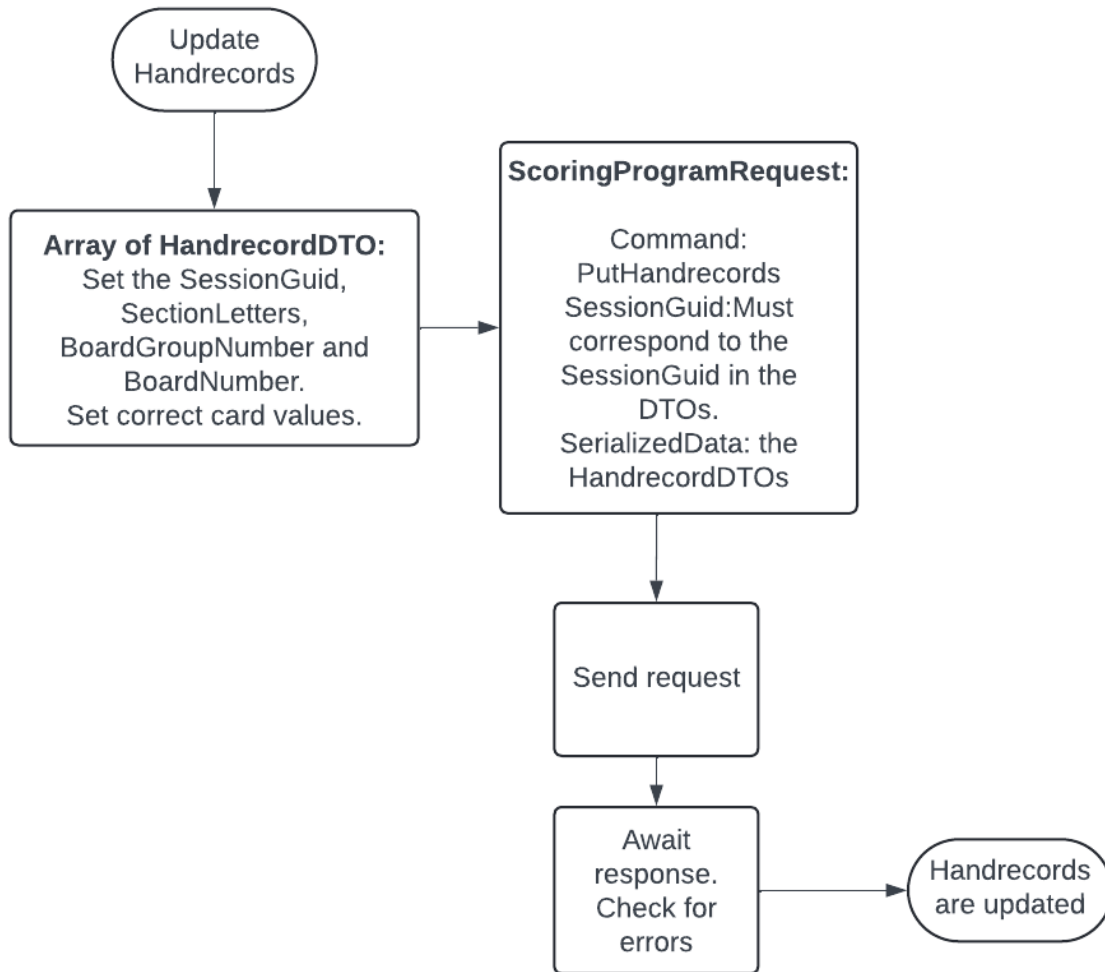
### Description

Participations can be updated with an array of [ParticipationDTOs](#) using the [PutParticipations command](#). Existing participations will be updated, enabling changing the player in a pair, the rest will be added. Each ParticipationDTO that is sent to the Bridgemate Data Connector with its SessionGuid and PlayerNumber properties set must have a corresponding PlayerData that was sent before. Participations preferably are sent with the [InitDTO](#) while [initializing the event](#): this is more performant. Use the updates for movement changes after the event has started.

#### Note

Currently only participations for round zero or one are accepted. BCS will calculate the participations for the subsequent rounds and add or update them.

## Update handrecords



### Data structure

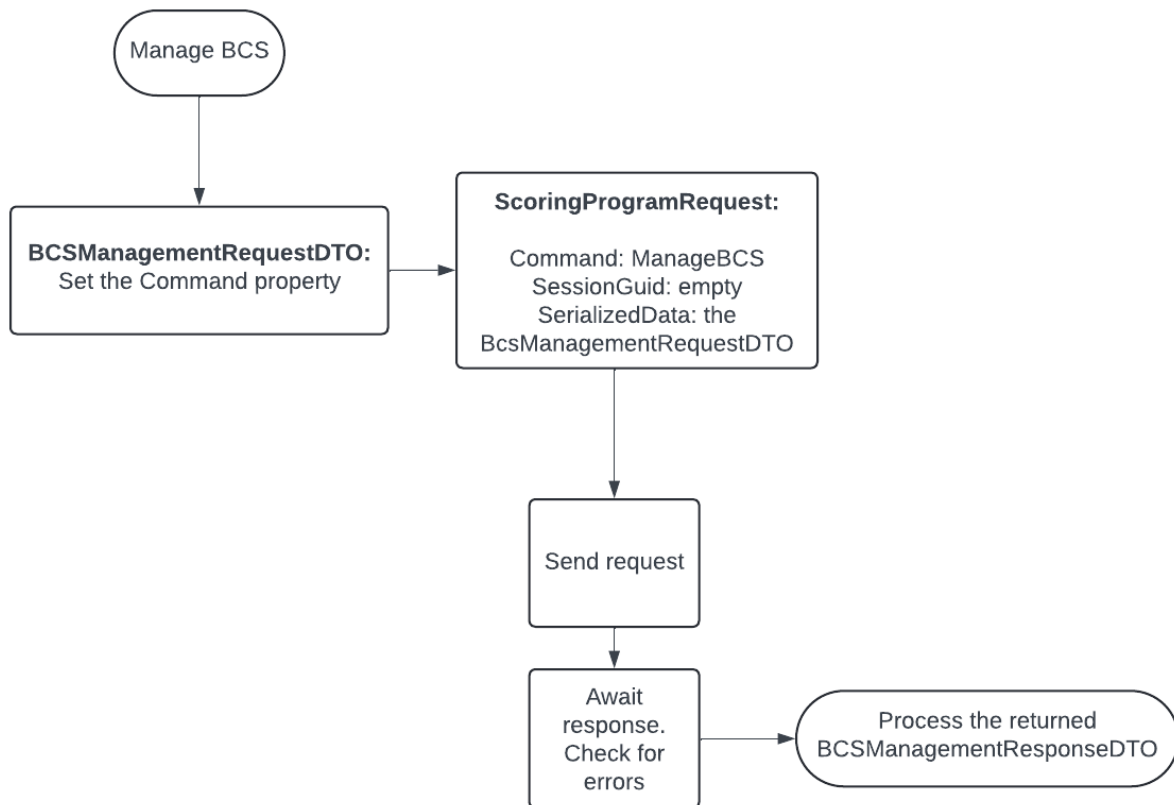
**HandrecordDTO** ⌵

Class

### Description

Handrecords can be updated with an array of [HandrecordDTOs](#) using the [PutHandrecords command](#). Existing handrecords will be updated, the rest will be added. Handrecords preferably are sent with the [InitDTO](#) while [initializing the event](#): this is more performant. Use the updates for movement changes or corrections after the event has started.

## Manage BCS



## Data structure



## Description

Using the [BCSManagementRequestDTO](#) and the [ManageBCS command](#) it is possible to query information about BCS' data and to issue a "Shut down now" command. In the last case the ScoringProgramResponse will not contain serialized data. In the other cases the response will carry a serialized [BCSManagementResponseDTO](#).

## Example json code

An example request asking for the scoring file location and the currently administered sessions:

```
{
  "SessionGuid":null,
  "Command":33,
  "SerializedData":
  "{
    \"Command\":6
  }"
```

SerializedData property of the scoring program response:

```
{
  "EventGuid": "9B720E0FEBF741E9B3D4D9C4C71CD732",
```

```

"SessionInformation": [
  {
    "EventGuid": "9B720E0FEBF741E9B3D4D9C4C71CD732",
    "SessionGuid": "9B720E0FEBF741E9B3D4D9C4C71CD732",
    "SessionName": "Friday evening bridge",
    "SessionDateTime": "2022-04-05T00:00:00"
  }
],
"IsRunning": true,
"ScoringFilePath":
"C:\\Users\\aners\\AppData\\Local\\BCS.Net\\ScoringFiles\\BridgeSystems.Bridgemate.BCSDData.db",
}

```

An example request asking for the scoring file location and all sessions known to BCS:

```

{
  "SessionGuid": null,
  "Command": 33,
  "SerializedData":
    "{
      \"Command\": 8
    }"
}

```

SerializedData property of the scoring program response:

```

{
  "EventGuid": "",
  "SessionInformation": [
    {
      "EventGuid": "7D20A8B6F2E7431F9D65B3FDDD874040",
      "SessionGuid": "7D20A8B6F2E7431F9D65B3FDDD874040",
      "SessionName": "Monday bridge",
      "SessionDateTime": "2024-06-12T00:00:00"
    },
    {
      "EventGuid": "AC46A9EFF3894F2D9A5434FCC76149F7",
      "SessionGuid": "AC46A9EFF3894F2D9A5434FCC76149F7",
      "SessionName": "Friday bridge",
      "SessionDateTime": "2024-06-11T00:00:00"
    },
    {
      "EventGuid": "ECE27FE8E19D4C72A57C759D460F8EB8",
      "SessionGuid": "ECE27FE8E19D4C72A57C759D460F8EB8",
      "SessionName": "Thursday bridge",
      "SessionDateTime": "2024-06-10T00:00:00"
    },
    {
      "EventGuid": "673B2AB4BA9F45BDB758026011F34B3E",
      "SessionGuid": "673B2AB4BA9F45BDB758026011F34B3E",
      "SessionName": "Qualification tournament",
      "SessionDateTime": "2022-05-24T00:00:00"
    }
  ],
  "IsRunning": true,
  "ScoringFilePath": "",
  "ValidationMessages": null
}

```

## Explanation of used terms

---

- **BCS or Bridgemate Control Software:** Windows software program to communicate with the Bridgemate base station, which communicates with the wireless Bridgemate terminals. Movement data is uploaded by this program to the base station, and results are retrieved from it the other way around. Without further specification the term "BCS" refers to the newer, .Net based, version as the original, classic, version does not support use of the Bridgemate Data Connector.
- **The (external) scoring program:** Windows software program that is used for scoring bridge sessions, calculating results, printing rankings etc and which uses Bridgemates for input of results.
- **The Data Connector:** The Bridgemate Data Connector process: a messaging system to send requests and commands from the external scoring program to BCS and to receive data from BCS for the external scoring program.
- **DTO: Data Transfer Object.** A class or structure holding data for BCS, to be sent to it using the Data Connector. A DTO must be serialized as json in a request to the Data Connector and must be deserialized from json when returned from the Data Connector
- **The communication channel:** the method used to exchange data with the Bridgemate Data Connector. The data will always be serialized as JSON. The transfer method currently is by using NamedPipes only.
- **Event:** one or more sessions that are grouped together in the Bridgemate servers.
- **Session:** a separately scored group of sections.
- **Sit-out:** a round on a table where a pair has no opponent.
- **Empty table:** a round on a table without pairs.
- **Participation:** the position of a player on a table in a given round, identified by his playernumber.

