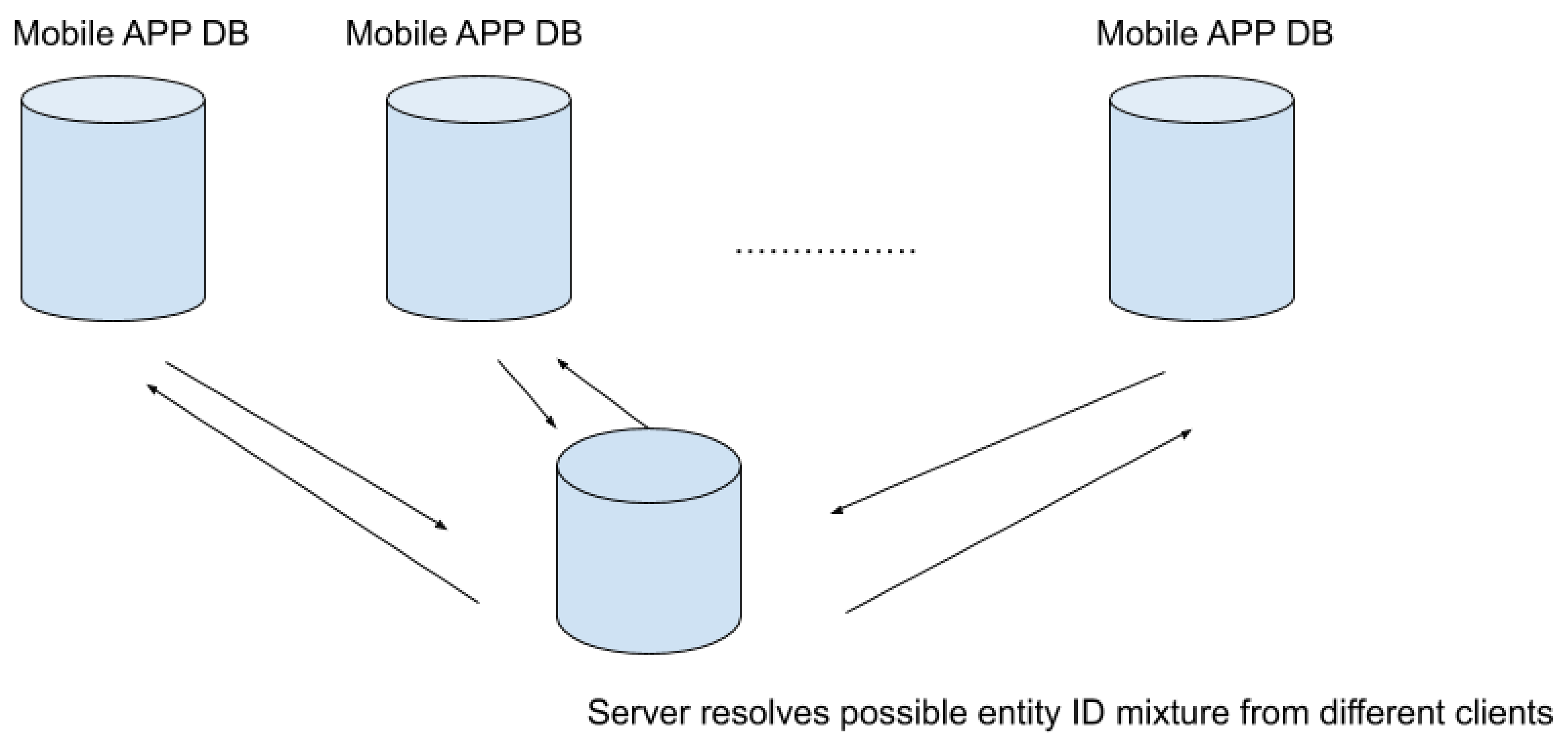
Introduction

EntitySyncing synchronizes entities of different types between the server and the clients, using DBreeze techniques (made for .NET C# Xamarin Core Standard).


Mobile APP DB
Mobile APP DB
Mobile APP DB
....
Server resolves possible entity ID mixture from different clients

Typical case is a mobile application (APP) that wants to have in a local database a list of TODO-tasks for the concrete user.
Local database gives an ability to read and create tasks being offline from the server.

Users can install such APP on several mobile devices.

EntitySyncingServer nuget package must be installed on the server, EntitySyncingClient must be installed on the client
and both must be configured.

The transfer data mechanizm is not implemented in this project (let's imagine that there is an open tcp/http channel between clients and the server), there are examples how to supply incoming data from the client to the server and which data to supply back.

General Provisions

Transferrable entities must be serializable and the serializer attached to DBreeze will be used to serialize and deserialize entities on both (client/server) sides.

Synchronization is based on the data that is stored in DBreeze tables. For each entity must be created one DBreeze table. Where first bytes 0xC8, 0xC9, 0xCA (200, 201, 202) will be used by sync mechanizm. Entity itself must be stored in the same (or in another table also possible) table as BLOB using InsertDataBlockWithFixedAddress.

The best new EntityID for the client (and server) is DateTime.UtcNow.Ticks, though the server can fix such IDs and even suggest a monotonically grown ID strategy.

Synchronization is always initiated from the client.

Synchronization lasts by chunks and repeats automatically, a maximal chunk that can be sent via wire is configured to 10K entities.

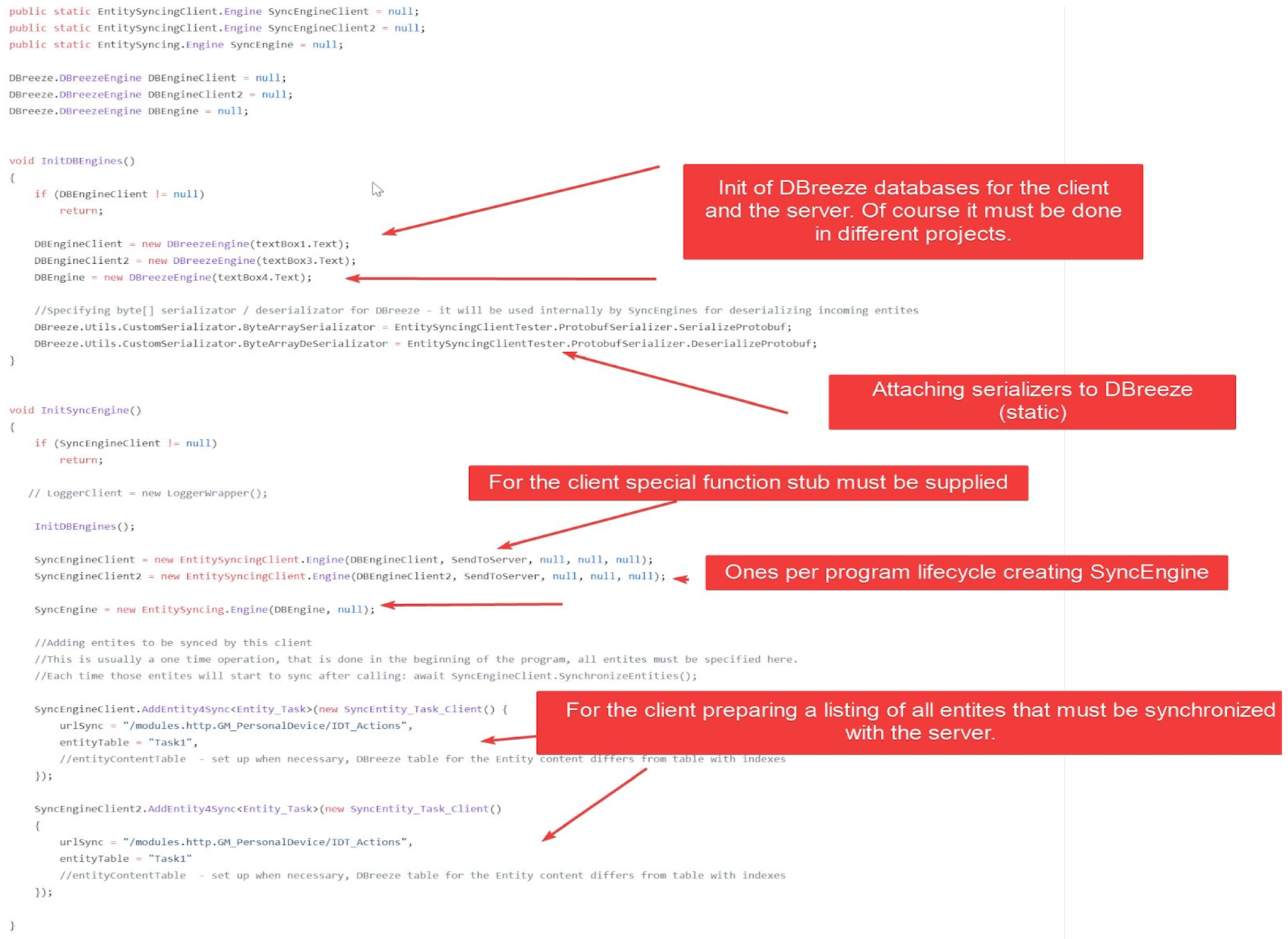
Entities can be synchronized in one of the following directions: both, from the client, from the server.

In case of an entity version conflict wins the newest entity.

There is a working example (just specify correct paths for DBreeze databases)

Initializations of the Client and the Server

EntitySyncing is a namespace for the server-side dll.
EntitySyncingClient is a namespace for the client-side dll.



```
public static EntitySyncingClient.Engine SyncEngineClient = null;
public static EntitySyncingClient.Engine SyncEngineClient2 = null;
public static EntitySyncing.Engine SyncEngine = null;

DBreeze.DBreezeEngine DBEngineClient = null;
DBreeze.DBreezeEngine DBEngineClient2 = null;
DBreeze.DBreezeEngine DBEngine = null;


void InitDBEngines()
{
    if (DBEngineClient != null)
        return;

    DBEngineClient = new DBreezeEngine(textBox1.Text);
    DBEngineClient2 = new DBreezeEngine(textBox3.Text);
    DBEngine = new DBreezeEngine(textBox4.Text);

    //Specifying byte[] serializator / deserializator for DBreeze - it will be used internally by SyncEngines for deserializing incoming entites
    DBreeze.Utils.CustomSerializator.ByteArraySerializator = EntitySyncingClientTester.ProtobufSerializer.SerializeProtobuf;
    DBreeze.Utils.CustomSerializator.ByteArrayDeSerializator = EntitySyncingClientTester.ProtobufSerializer.DeserializeProtobuf;
}
```

Init of DBreeze databases for the client and the server. Of course it must be done in different projects.

Attaching serializers to DBreeze (static)

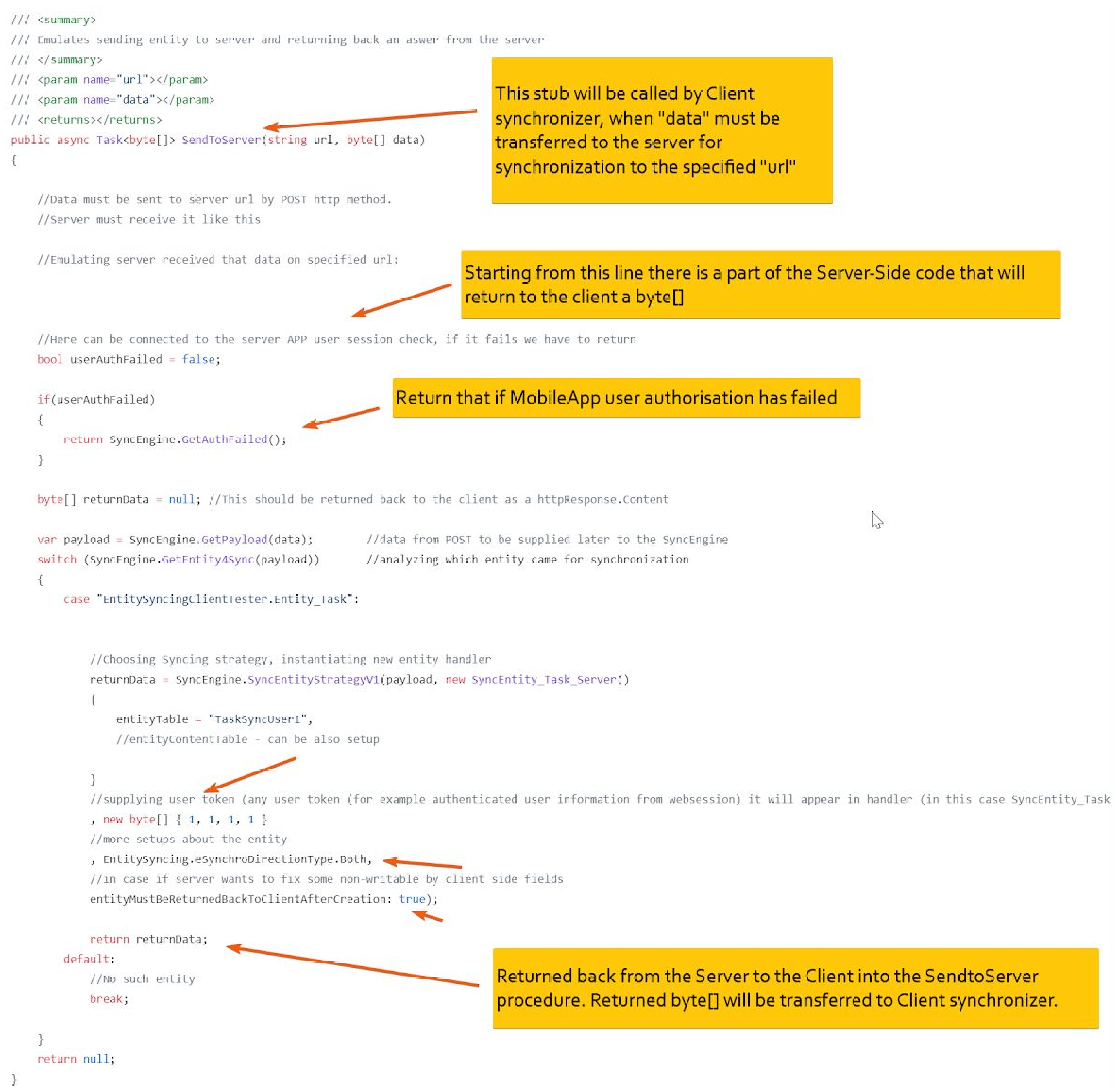
```
void InitSyncEngine()
{
    if (SyncEngineClient != null)
        return;

   // LoggerClient = new LoggerWrapper();

    InitDBEngines();

    SyncEngineClient = new EntitySyncingClient.Engine(DBEngineClient, SendToServer, null, null, null);
    SyncEngineClient2 = new EntitySyncingClient.Engine(DBEngineClient2, SendToServer, null, null, null);

    SyncEngine = new EntitySyncing.Engine(DBEngine, null);

    //Adding entites to be synced by this client
    //This is usually a one time operation, that is done in the beginning of the program, all entites must be specified here.
    //Each time those entites will start to sync after calling: await SyncEngineClient.SynchronizeEntities();

    SyncEngineClient.AddEntity4Sync<Entity_Task>(new SyncEntity_Task_Client() {
        urlSync = "/modules.http.GM_PersonalDevice/IDT_Actions",
        entityTable = "Task1",
        //entityContentTable  - set up when necessary, DBreeze table for the Entity content differs from table with indexes
    });

    SyncEngineClient2.AddEntity4Sync<Entity_Task>(new SyncEntity_Task_Client()
    {
        urlSync = "/modules.http.GM_PersonalDevice/IDT_Actions",
        entityTable = "Task1"
        //entityContentTable  - set up when necessary, DBreeze table for the Entity content differs from table with indexes
    });

}
```

For the client special function stub must be supplied

Ones per program lifecycle creating SyncEngine

For the client preparing a listing of all entites that must be synchronized with the server.

Start of synchronization on the Client and the Server



```
/// <summary>
/// Emulates sending entity to server and returning back an aswer from the server
/// </summary>
/// <param name="url"></param>
/// <param name="data"></param>
/// <returns></returns>
public async Task<byte[]> SendToServer(string url, byte[] data)
{

    //Data must be sent to server url by POST http method.
    //Server must receive it like this

    //Emulating server received that data on specified url:



    //Here can be connected to the server APP user session check, if it fails we have to return
    bool userAuthFailed = false;

    if(userAuthFailed)
    {
        return SyncEngine.GetAuthFailed();
    }

    byte[] returnData = null; //This should be returned back to the client as a httpResponse.Content

    var payload = SyncEngine.GetPayload(data);        //data from POST to be supplied later to the SyncEngine
    switch (SyncEngine.GetEntity4Sync(payload))       //analyzing which entity came for synchronization
    {
        case "EntitySyncingClientTester.Entity_Task":


            //Choosing Syncing strategy, instantiating new entity handler
            returnData = SyncEngine.SyncEntityStrategyV1(payload, new SyncEntity_Task_Server()
            {
                entityTable = "TaskSyncUser1",
                //entityContentTable - can be also setup

            }
            //supplying user token (any user token (for example authenticated user information from websession) it will appear in handler (in this case SyncEntity_Task
            , new byte[] { 1, 1, 1, 1 }
            //more setups about the entity
            , EntitySyncing.eSynchroDirectionType.Both,
            //in case if server wants to fix some non-writable by client side fields
            entityMustBeReturnedBackToClientAfterCreation: true);

            return returnData;
        default:
            //No such entity
            break;

    }
    return null;
}
```

This stub will be called by Client synchronizer, when "data" must be transferred to the server for synchronization to the specified "url"

Starting from this line there is a part of the Server-Side code that will return to the client a byte[]

Return that if MobileApp user authorisation has failed

Returned back from the Server to the Client into the SendtoServer procedure. Returned byte[] will be transferred to Client synchronizer.

Entity definition

```
[ProtoBuf.ProtoContract]
public partial class Entity_Task
{

    [ProtoBuf.ProtoMember(1, IsRequired = true)]
    public long Id { get; set; }

    [ProtoBuf.ProtoMember(2, IsRequired = true)]
    public long SyncTimestamp { get; set; }

    [ProtoBuf.ProtoMember(3, IsRequired = true)]
    public bool Deleted { get; set; } = false;

    [ProtoBuf.ProtoMember(4, IsRequired = true)]
    public string Description { get; set; }
}
```

On shared entity definition example between client-side and server-side

Such partial stub of ISyncEntity must be added on the client-side

```
/// <summary>
/// Add on the client-side together with entity itself
/// </summary>
public partial class Entity_Task : EntitySyncingClient.ISyncEntity
{

}
```

Such partial stub of ISyncEntity must be added on the server-side (unremarked, of course)

```
///// <summary>
///// !!!!!!!!!!!! Add on the client-side together with entity itself
///// </summary>
//public partial class Entity_Task : Entity_Task, EntitySyncing.ISyncEntity
//{

//}
```

Removing entities

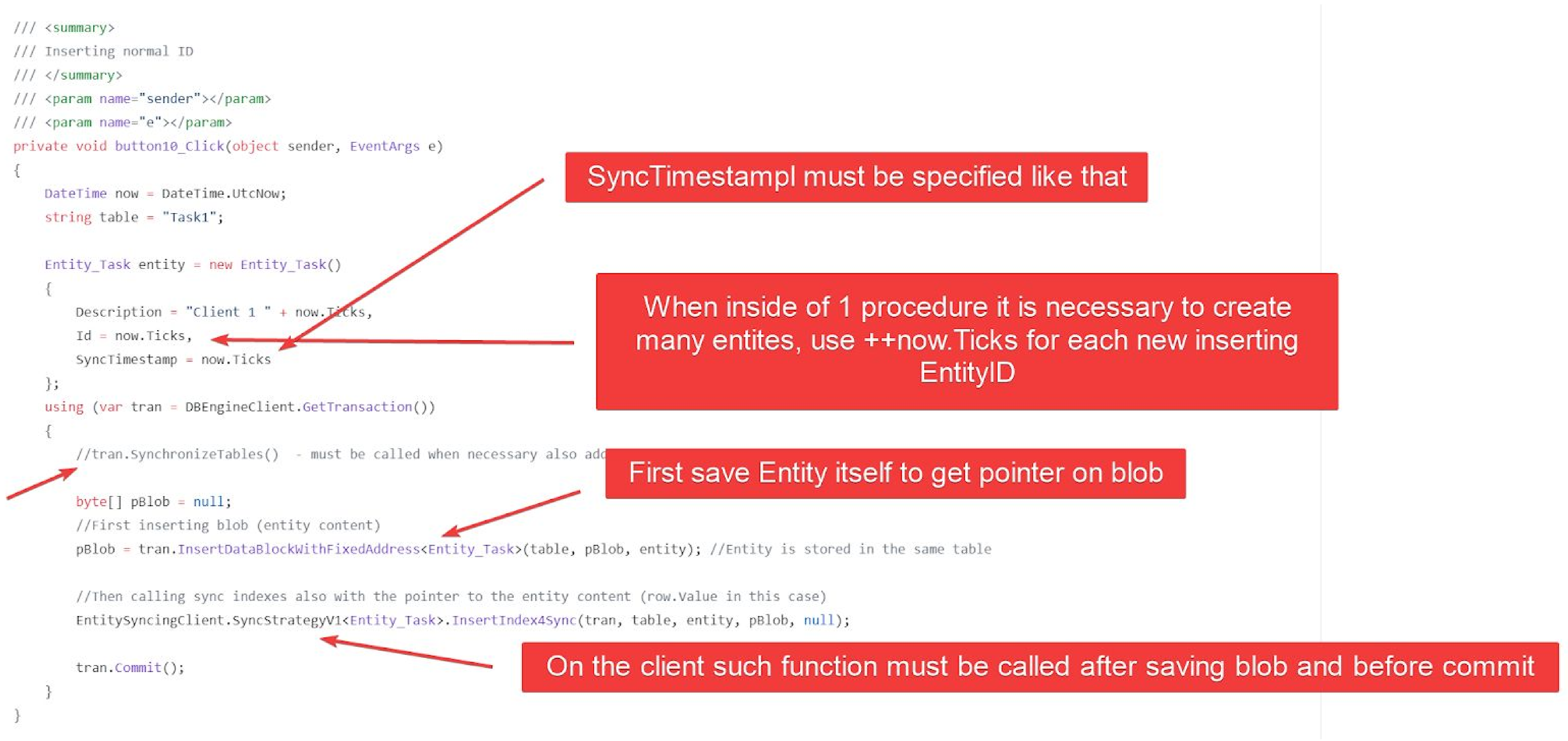
Entities must not be deleted, but marked as Deleted using ISyncEntity.Deleted = true, so on any side it will be possible to distinguish between non-deleted ones.

Inserting entities

Entity can be inserted from 2 different points:

Arbitrary / Random - practically from any part of the client software or via **SyncHandler**.

Inserting Arbitrary entities on the client



```
/// <summary>
/// Inserting normal ID
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button10_Click(object sender, EventArgs e)
{
    DateTime now = DateTime.UtcNow;
    string table = "Task1";

    Entity_Task entity = new Entity_Task()
    {
        Description = "Client 1 " + now.Ticks,
        Id = now.Ticks,
        SyncTimestamp = now.Ticks
    };
    using (var tran = DBEngineClient.GetTransaction())
    {
        //tran.SynchronizeTables()  - must be called when necessary also add

        byte[] pBlob = null;
        //First inserting blob (entity content)
        pBlob = tran.InsertDataBlockWithFixedAddress<Entity_Task>(table, pBlob, entity); //Entity is stored in the same table

        //Then calling sync indexes also with the pointer to the entity content (row.Value in this case)
        EntitySyncingClient.SyncStrategyV1<Entity_Task>.InsertIndex4Sync(tran, table, entity, pBlob, null);

        tran.Commit();
    }
}
```

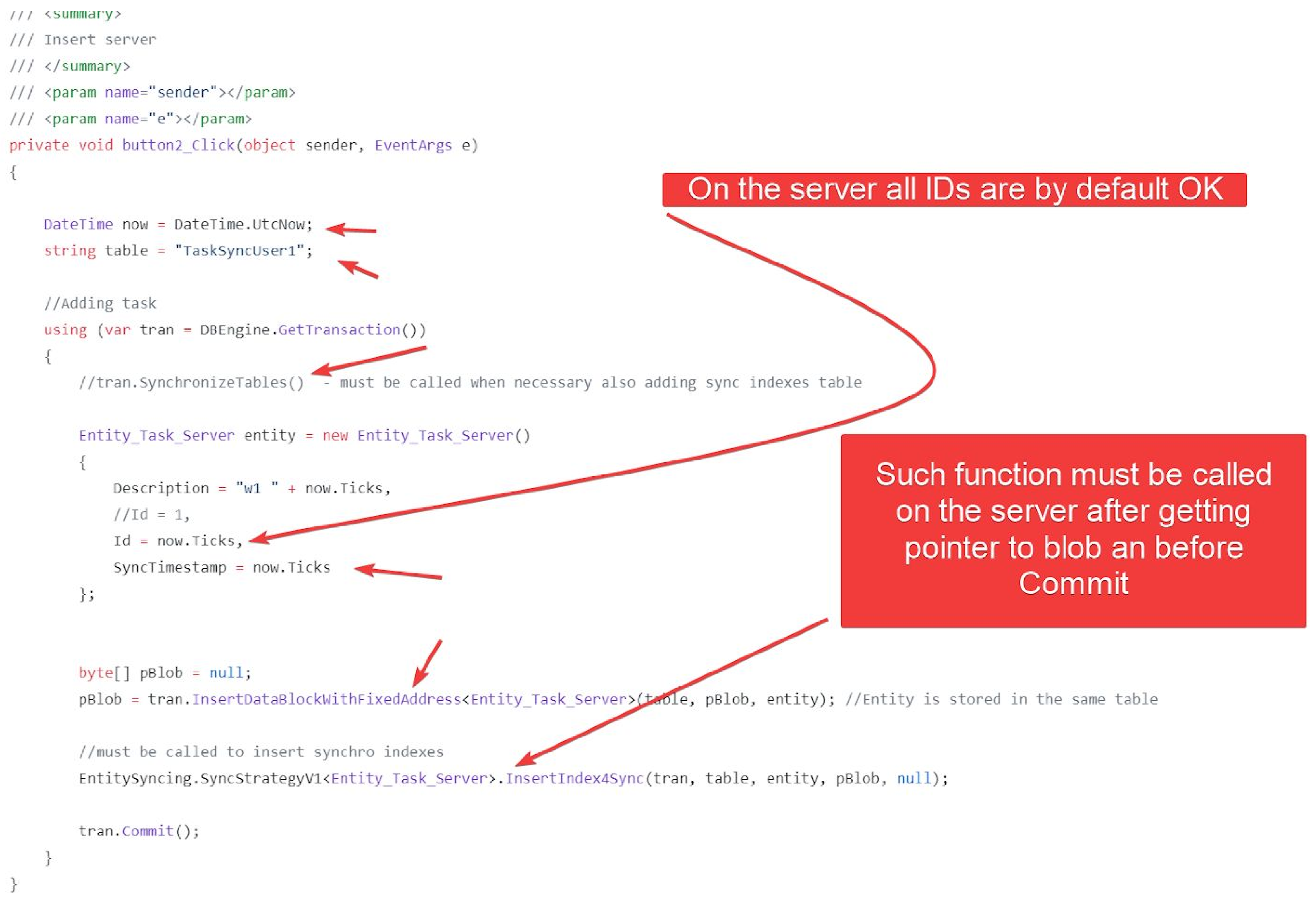
SyncTimestampl must be specified like that

When inside of 1 procedure it is necessary to create many entites, use ++now.Ticks for each new inserting EntityID

First save Entity itself to get pointer on blob

On the client such function must be called after saving blob and before commit

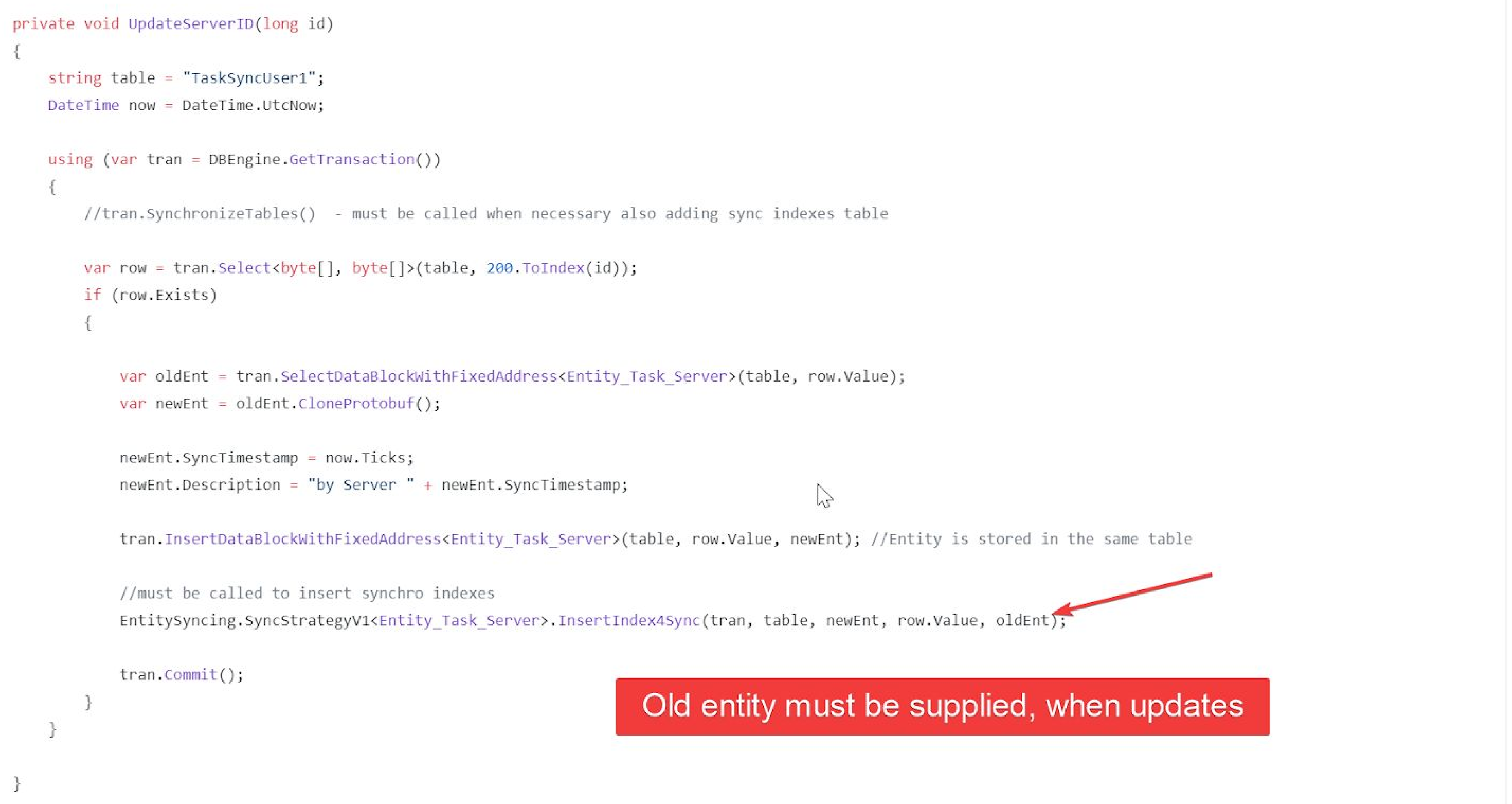
Inserting/Updating Arbitrary entities on the server



```
/// <summary>
/// Insert server
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button2_Click(object sender, EventArgs e)
{

    DateTime now = DateTime.UtcNow;
    string table = "TaskSyncUser1";

    //Adding task
    using (var tran = DBEngine.GetTransaction())
    {
        //tran.SynchronizeTables()  - must be called when necessary also adding sync indexes table

        Entity_Task_Server entity = new Entity_Task_Server()
        {
            Description = "w1 " + now.Ticks,
            //Id = 1,
            Id = now.Ticks,
            SyncTimestamp = now.Ticks
        };


        byte[] pBlob = null;
        pBlob = tran.InsertDataBlockWithFixedAddress<Entity_Task_Server>(table, pBlob, entity); //Entity is stored in the same table

        //must be called to insert synchro indexes
        EntitySyncing.SyncStrategyV1<Entity_Task_Server>.InsertIndex4Sync(tran, table, entity, pBlob, null);

        tran.Commit();
    }
}
```

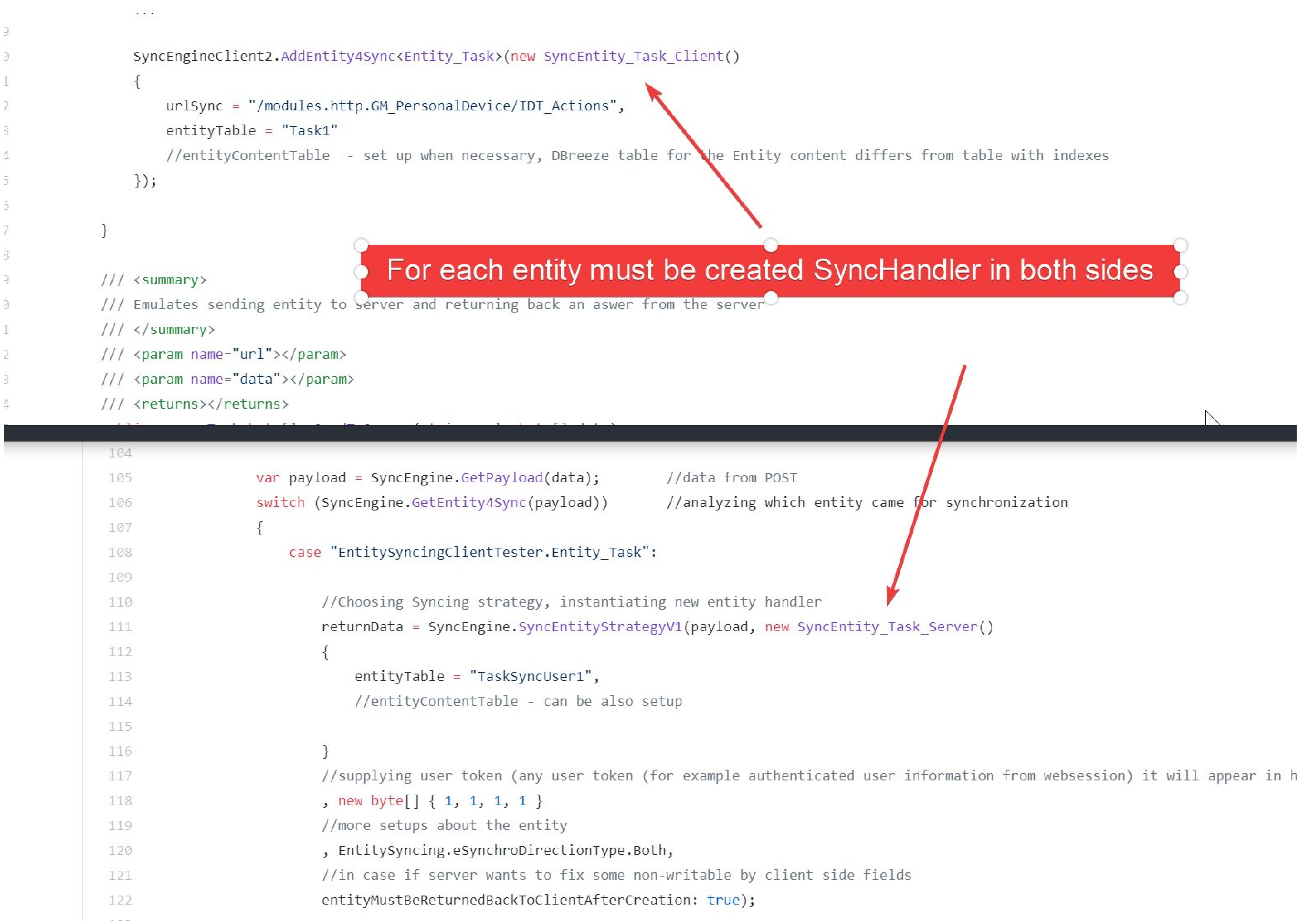
On the server all IDs are by default OK

Such function must be called on the server after getting pointer to blob an before Commit



```
private void UpdateServerID(long id)
{
    string table = "TaskSyncUser1";
    DateTime now = DateTime.UtcNow;

    using (var tran = DBEngine.GetTransaction())
    {
        //tran.SynchronizeTables()  - must be called when necessary also adding sync indexes table

        var row = tran.Select<byte[], byte[]>(table, 200.ToIndex(id));
        if (row.Exists)
        {

            var oldEnt = tran.SelectDataBlockWithFixedAddress<Entity_Task_Server>(table, row.Value);
            var newEnt = oldEnt.CloneProtobuf();

            newEnt.SyncTimestamp = now.Ticks;
            newEnt.Description = "by Server " + newEnt.SyncTimestamp;

            tran.InsertDataBlockWithFixedAddress<Entity_Task_Server>(table, row.Value, newEnt); //Entity is stored in the same table

            //must be called to insert synchro indexes
            EntitySyncing.SyncStrategyV1<Entity_Task_Server>.InsertIndex4Sync(tran, table, newEnt, row.Value, oldEnt);

            tran.Commit();
        }
    }

}
```

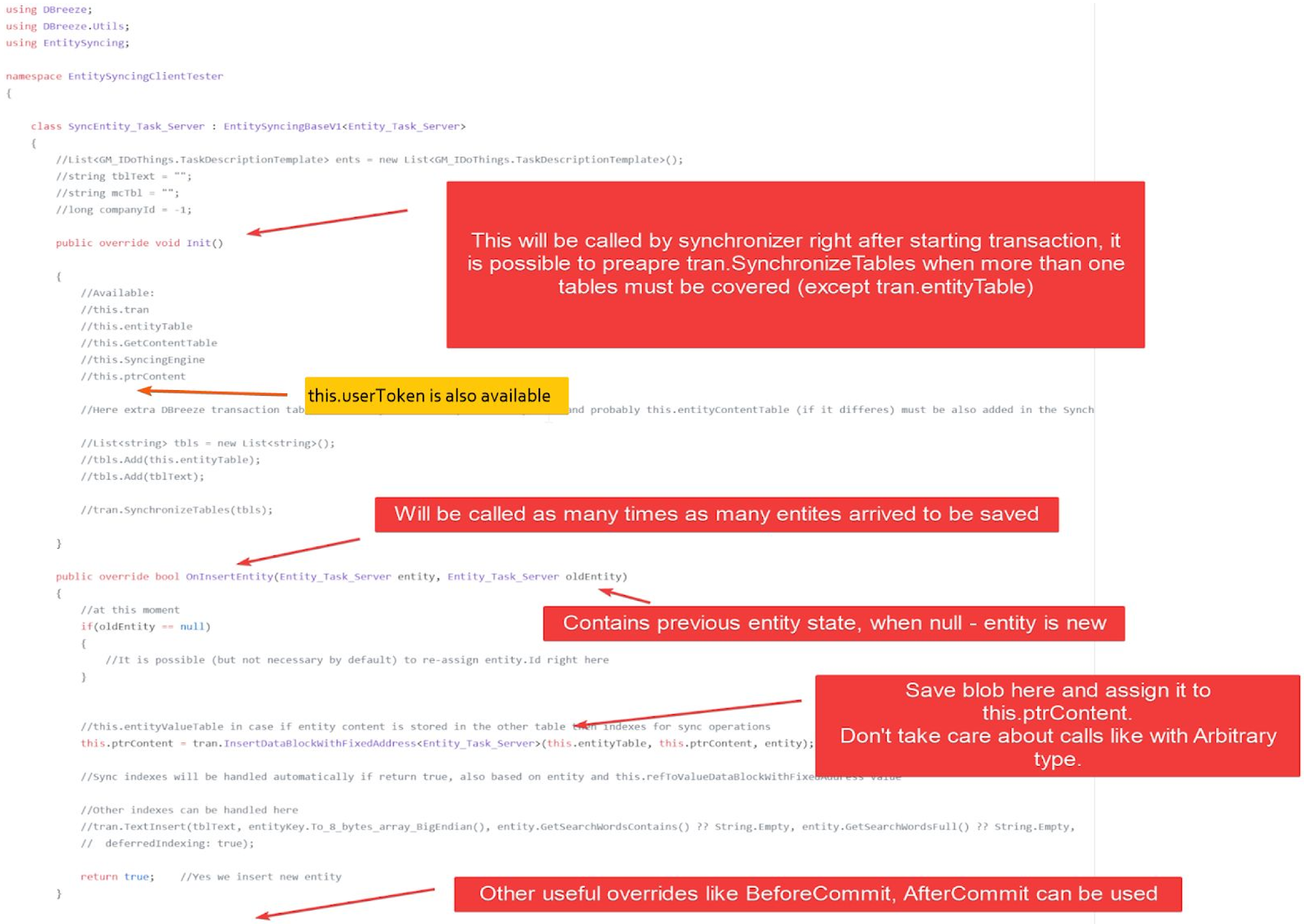
Inserting/Updating via SyncHandler



```
    SyncEngineClient2.AddEntity4Sync<Entity_Task>(new SyncEntity_Task_Client()
    {
        urlSync = "/modules.http.GM_PersonalDevice/IDT_Actions",
        entityTable = "Task1"
        //entityContentTable  - set up when necessary, DBreeze table for the Entity content differs from table with indexes
    });

}

/// <summary>
/// Emulates sending entity to server and returning back an aswer from the server
/// </summary>
/// <param name="url"></param>
/// <param name="data"></param>
/// <returns></returns>
```

For each entity must be created SyncHandler in both sides

```

            var payload = SyncEngine.GetPayload(data);        //data from POST
            switch (SyncEngine.GetEntity4Sync(payload))       //analyzing which entity came for synchronization
            {
                case "EntitySyncingClientTester.Entity_Task":

                    //Choosing Syncing strategy, instantiating new entity handler
                    returnData = SyncEngine.SyncEntityStrategyV1(payload, new SyncEntity_Task_Server()
                    {
                        entityTable = "TaskSyncUser1",
                        //entityContentTable - can be also setup

                    }
                    //supplying user token (any user token (for example authenticated user information from websession) it will appear in h
                    , new byte[] { 1, 1, 1, 1 }
                    //more setups about the entity
                    , EntitySyncing.eSynchroDirectionType.Both,
                    //in case if server wants to fix some non-writable by client side fields
                    entityMustBeReturnedBackToClientAfterCreation: true);
```



```
using DBreeze;
using DBreeze.Utils;
using EntitySyncing;

namespace EntitySyncingClientTester
{

    class SyncEntity_Task_Server : EntitySyncingBaseV1<Entity_Task_Server>
    {
        //List<GM_IDoThings.TaskDescriptionTemplate> ents = new List<GM_IDoThings.TaskDescriptionTemplate>();
        //string tblText = "";
        //string mcTbl = "";
        //long companyId = -1;

        public override void Init()

        {
            //Available:
            //this.tran
            //this.entityTable
            //this.GetContentTable
            //this.SyncingEngine
            //this.ptrContent

            //Here extra DBreeze transaction tab[illegible]and probably this.entityContentTable (if it differes) must be also added in the Synch

            //List<string> tbls = new List<string>();
            //tbls.Add(this.entityTable);
            //tbls.Add(tblText);

            //tran.SynchronizeTables(tbls);

        }

        public override bool OnInsertEntity(Entity_Task_Server entity, Entity_Task_Server oldEntity)
        {
            //at this moment
            if(oldEntity == null)
            {
                //It is possible (but not necessary by default) to re-assign entity.Id right here
            }


            //this.entityValueTable in case if entity content is stored in the other table then indexes for sync operations
            this.ptrContent = tran.InsertDataBlockWithFixedAddress<Entity_Task_Server>(this.entityTable, this.ptrContent, entity);

            //Sync indexes will be handled automatically if return true, also based on entity and this.refToValueDataBlockWithFixedAddress value

            //Other indexes can be handled here
            //tran.TextInsert(tblText, entityKey.To_8_bytes_array_BigEndian(), entity.GetSearchWordsContains() ?? String.Empty, entity.GetSearchWordsFull() ?? String.Empty,
            //  deferredIndexing: true);

            return true;    //Yes we insert new entity
        }
```

This will be called by synchronizer right after starting transaction, it is possible to preapre tran.SynchronizeTables when more than one tables must be covered (except tran.entityTable)

this.userToken is also available

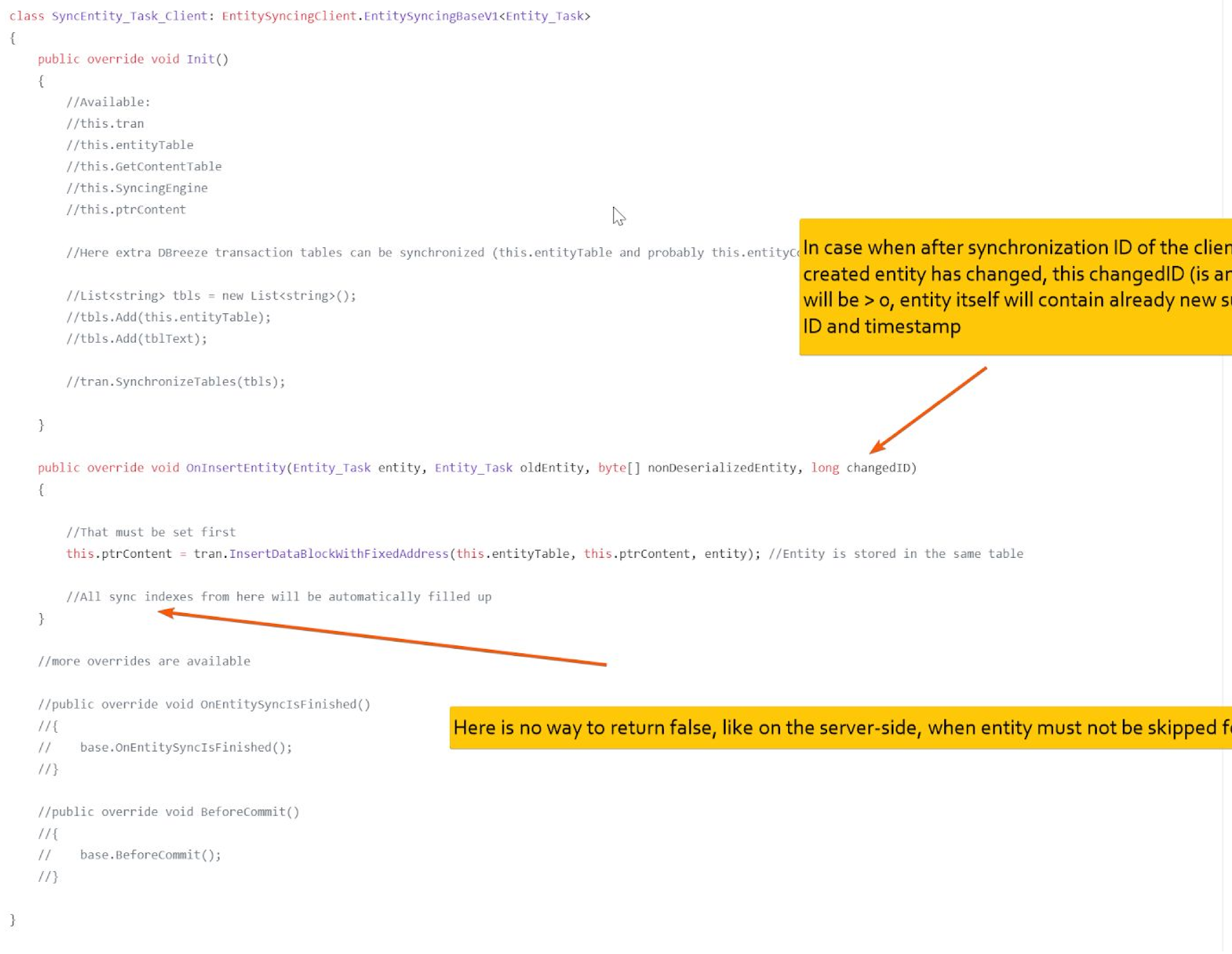
Will be called as many times as many entites arrived to be saved

Contains previous entity state, when null - entity is new

Save blob here and assign it to this.ptrContent.
Don't take care about calls like with Arbitrary type.

Other useful overrides like BeforeCommit, AfterCommit can be used

Client SyncHandler differs a bit from server-side SyncHandler type, but not much.



```
class SyncEntity_Task_Client: EntitySyncingClient.EntitySyncingBaseV1<Entity_Task>
{
    public override void Init()
    {
        //Available:
        //this.tran
        //this.entityTable
        //this.GetContentTable
        //this.SyncingEngine
        //this.ptrContent

        //Here extra DBreeze transaction tables can be synchronized (this.entityTable and probably this.entityC

        //List<string> tbls = new List<string>();
        //tbls.Add(this.entityTable);
        //tbls.Add(tblText);

        //tran.SynchronizeTables(tbls);

    }

    public override void OnInsertEntity(Entity_Task entity, Entity_Task oldEntity, byte[] nonDeserializedEntity, long changedID)
    {

        //That must be set first
        this.ptrContent = tran.InsertDataBlockWithFixedAddress(this.entityTable, this.ptrContent, entity); //Entity is stored in the same table

        //All sync indexes from here will be automatically filled up
    }

    //more overrides are available

    //public override void OnEntitySyncIsFinished()
    //{
    //    base.OnEntitySyncIsFinished();
    //}

    //public override void BeforeCommit()
    //{
    //    base.BeforeCommit();
    //}

}
```

In case when after synchronization ID of the client-side created entity has changed, this changedID (is an old id) will be > 0, entity itself will contain already new suggested ID and timestamp

Here is no way to return false, like on the server-side, when entity must not be skipped for storing.

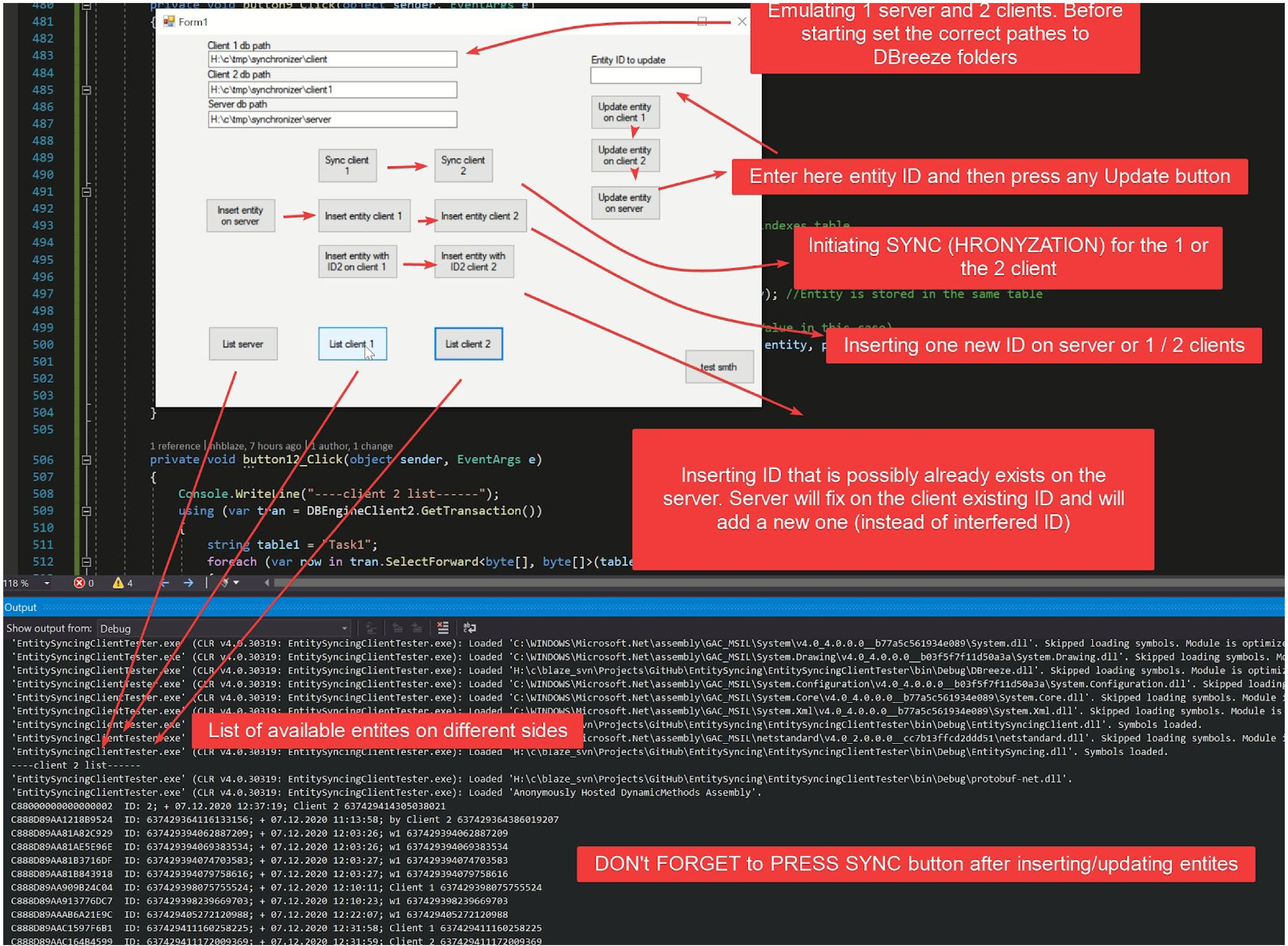
Inside of OnInsertEntity server can try to fix the newly arrived EntityID or just let it work automatically.

Client side dependent entities

It can happen that a new entity on the client side receives a new ID (updates its ID). That’s why client-side inside of SyncHandler.OnInsertEntity must try to solve possible dependent entities problems.
There are many ways to do that, e.g. holding dependent objects references inside the updated entity.
Client-side OnInsertEntity inside the EntitySyncHandler returns changedID.

Different scenarios can be discussed later to fill up this chapter.

Example application



Emulating 1 server and 2 clients. Before starting set the correct pathes to DBreeze folders

Enter here entity ID and then press any Update button

Initiating SYNC (HRONYZATION) for the 1 or the 2 client

Inserting one new ID on server or 1 / 2 clients

Inserting ID that is possibly already exists on the server. Server will fix on the client existing ID and will add a new one (instead of interfered ID)

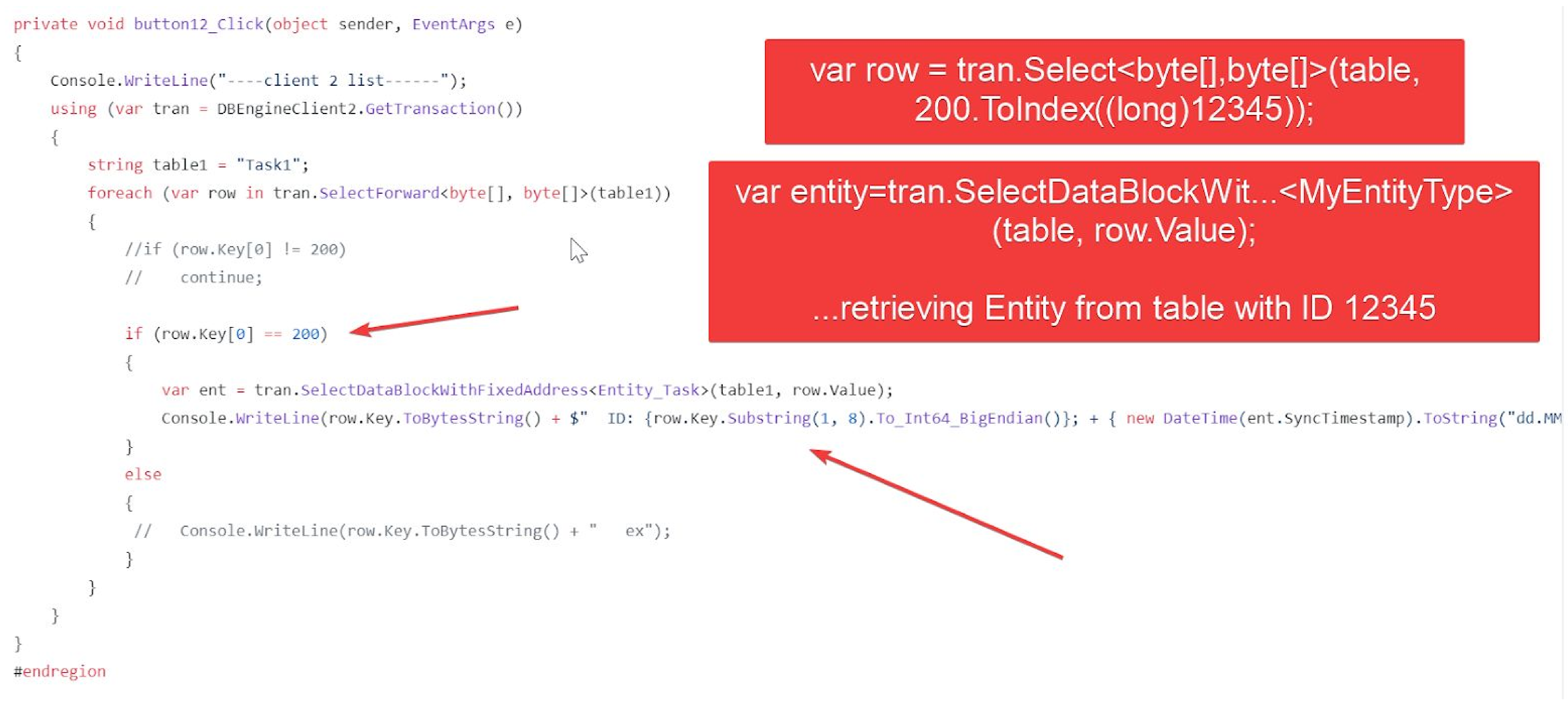
List of available entites on different sides

DON't FORGET to PRESS SYNC button after inserting/updating entites

```
private void button9_Click(object sender, EventArgs e)
{
                                            ndexes table
                                            ); //Entity is stored in the same table
                                            alue in this case)
                                            entity, p
}

1 reference | hhblaze, 7 hours ago | 1 author, 1 change
private void button12_Click(object sender, EventArgs e)
{
    Console.WriteLine("----client 2 list------");
    using (var tran = DBEngineClient2.GetTransaction())
    {
        string table1 = "Task1";
        foreach (var row in tran.SelectForward<byte[], byte[]>(table
```

```
Output
Show output from: Debug
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System\v4.0_4.0.0.0__b77a5c561934e089\System.dll'. Skipped loading symbols. Module is optimize
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Drawing\v4.0_4.0.0.0__b03f5f7f11d50a3a\System.Drawing.dll'. Skipped loading symbols. Mo
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'H:\c\blaze_svn\Projects\GitHub\EntitySyncing\EntitySyncingClientTester\bin\Debug\DBreeze.dll'. Skipped loading symbols. Module is optimiz
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Configuration\v4.0_4.0.0.0__b03f5f7f11d50a3a\System.Configuration.dll'. Skipped loading
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Core\v4.0_4.0.0.0__b77a5c561934e089\System.Core.dll'. Skipped loading symbols. Module i
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Xml\v4.0_4.0.0.0__b77a5c561934e089\System.Xml.dll'. Skipped loading symbols. Module is
'EntitySyncingClientTester.exe'                                         vn\Projects\GitHub\EntitySyncing\EntitySyncingClientTester\bin\Debug\EntitySyncingClient.dll'. Symbols loaded.
'EntitySyncingClientTester.exe'                                         icrosoft.Net\assembly\GAC_MSIL\netstandard\v4.0_2.0.0.0__cc7b13ffcd2ddd51\netstandard.dll'. Skipped loading symbols. Module i
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'H:\c\blaze_svn\Projects\GitHub\EntitySyncing\EntitySyncingClientTester\bin\Debug\EntitySyncing.dll'. Symbols loaded.
----client 2 list------
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'H:\c\blaze_svn\Projects\GitHub\EntitySyncing\EntitySyncingClientTester\bin\Debug\protobuf-net.dll'.
'EntitySyncingClientTester.exe' (CLR v4.0.30319: EntitySyncingClientTester.exe): Loaded 'Anonymously Hosted DynamicMethods Assembly'.
C880000000000002  ID: 2; + 07.12.2020 12:37:19; Client 2 637429414305038021
C888D89AA1218B9524  ID: 637429364116133156; + 07.12.2020 11:13:58; by Client 2 637429364386019207
C888D89AA81A82C929  ID: 637429394062887209; + 07.12.2020 12:03:26; w1 637429394062887209
C888D89AA81AE5E96E  ID: 637429394069383534; + 07.12.2020 12:03:26; w1 637429394069383534
C888D89AA81B3716DF  ID: 637429394074703583; + 07.12.2020 12:03:27; w1 637429394074703583
C888D89AA81B843918  ID: 637429394079758616; + 07.12.2020 12:03:27; w1 637429394079758616
C888D89AA909B24C04  ID: 637429398075755524; + 07.12.2020 12:10:11; Client 1 637429398075755524
C888D89AA913776DC7  ID: 637429398239669703; + 07.12.2020 12:10:23; w1 637429398239669703
C888D89AAAB6A21E9C  ID: 637429405272120988; + 07.12.2020 12:22:07; w1 637429405272120988
C888D89AAC1597F6B1  ID: 637429411160258225; + 07.12.2020 12:31:58; Client 1 637429411160258225
C888D89AAC164B4599  ID: 637429411172009369; + 07.12.2020 12:31:59; Client 2 637429411172009369
```

In List server/client functions it is possible to see how to retrieve entity by ID:



```
private void button12_Click(object sender, EventArgs e)
{
    Console.WriteLine("----client 2 list------");
    using (var tran = DBEngineClient2.GetTransaction())
    {
        string table1 = "Task1";
        foreach (var row in tran.SelectForward<byte[], byte[]>(table1))
        {
            //if (row.Key[0] != 200)
            //    continue;

            if (row.Key[0] == 200)
            {
                var ent = tran.SelectDataBlockWithFixedAddress<Entity_Task>(table1, row.Value);
                Console.WriteLine(row.Key.ToBytesString() + $"  ID: {row.Key.Substring(1, 8).To_Int64_BigEndian()}; + { new DateTime(ent.SyncTimestamp).ToString("dd.MM
            }
            else
            {
             //   Console.WriteLine(row.Key.ToBytesString() + "   ex");
            }
        }
    }
}
#endregion
```

var row = tran.Select<byte[],byte[]>(table,
200.ToIndex((long)12345));

var entity=tran.SelectDataBlockWit...<MyEntityType>
(table, row.Value);

...retrieving Entity from table with ID 12345