

# NHibernate.Remote

## Reference Documentation

Version 0.2.x

<http://slagd.com>

## Table of Contents

1. About
  - 1.1. Technical Brief
    - 1.1.1. Data Transfers
    - 1.1.2. Sessions
    - 1.1.3. Saves
    - 1.1.4. Deletes
    - 1.1.5. Updates
  - 1.2. What works and what doesn't
    - 1.2.1. Working
    - 1.2.2. Not Working
    - 1.2.3. Not tested very well
    - 1.2.4. Methods working out of an ISession
2. Using NHibernate.Remote
  - 2.1. Server side
    - 2.1.1. References
    - 2.1.2. App.config
    - 2.1.3. Code
  - 2.2. Hosting in IIS
    - 2.2.1. Web.config
    - 2.2.2. The .svc file
  - 2.3. Client side
    - 2.3.1. References
    - 2.3.2. App.config
    - 2.3.3. Code
3. Technical Details



# 1. About

NHibernate.Remote aims to make the excellent [NHibernate](#) ORM mapper easier to use over the Windows Communication Foundation (WCF). There has always been trouble in getting data across app-domain boundaries and working with essentially disconnected data will always be more difficult than when you are directly tied to your database. While NHibernate.Remote will not erase those boundaries, its goal is to make it easier by allowing you to do most of the same operations remotely as you would if you were using an NHibernate session directly.

This documentation assumes that you are already familiar with the workings of NHibernate. If you are not, please do so at your earliest opportunity.

## 1.1 Technical Brief

### 1.1.1 Data Transfers

How do we transfer data in NHibernate.Remote? There are several main hurdles involved in getting NHibernate's functionality across the wire.

1. NHibernate makes good use of .net generics. If you have ever tried to use open generics over WCF or webservices you will find out that the serializer will throw a very nasty error.
2. WCF would "like" that the data sent over the wire is serialized in a very disconnected fashion. This means that the object that appears on the client end is very different than the one defined on the server. This is definitely the preferred method for interoperability between loosely coupled systems. However it does not make much sense (IMHO) when you know that all of your clients will be .net and you have good control over these clients.

So how are these challenges overcome in NHibernate.Remote?

1. A custom messaging library had to be developed to enable the exchange of open generics across WCF. A remote method is matched to a method on the server just the same as in regular WCF. However before it gets sent to the server it is broken down into its component types thus enabling the serializer to process it. On the server end the message is reconstituted and a message router uses reflection to call the appropriate method on the server. The response is then processed in the same way.
2. In order to make sure we have type continuity between the server and the client we have to force WCF to use the NetDataContractSerializer as described in Tim Scott's article [here](#). This is not the recommended route but we can take it because we control both the client and the server.

### 1.1.2 Sessions

Sessions are one of the more sticky issues involved in this technology. Right now I've chosen to implement this using WCF's reliable sessions. Basically this means that when you open a Remote Session on the client side an NHibernate session is opened on the server side and there is a one to one relationship between these sessions. When you dispose of a session with the client,

then that session is also disposed of on the server, the same goes with flushes and so on. These sessions should ultimately mirror each other.

Why do it this way? The other way would have been to go with a stateless server session which would obviously be better in terms of performance. The client would then submit requests only as needed. This is very tempting and could very well be looked at later on, however the loss of lazy loading and state tracking was not very appealing. For this first run I wanted to mimic a stateful ISession as closely as possible.

### **1.1.3 Saves**

Saving right now is an immediate call. When you save an item on the client it saves it on the server immediately. This is so that we can get a valid ID back to the client. This also most closely mimics NHibernate's functionality. Later on perhaps, we will make it so that saves that already have an ID or have a non DB generated ID do not have to be transmitted immediately but will rather get included in the flush data.

### **1.1.4 Deletes**

Deletes are also an immediate call. Deleting an item with the delete command will also remove that item from the persistence context so that further changes are not tracked. However if you delete by using a query some of those entities may still exist in the session cache causing unforeseen problems.

### **1.1.5 Updates**

Updates are a complicated thing. To help you better understand them, I will lay out the lifecycle of an entity that gets updated.

1. An entity that is queried arrives at the client session.
2. If the entity does not already exist in the client side session
  - a. A snapshot of its state is taken
  - b. It is stored in the persistence context
3. The user makes some changes to the entity that they received
4. The user calls Flush()
5. All of the entities that are currently stored in the client persistence context are examined
6. The current state of each entity is compared against its snapshot
7. If there are any changes they are compiled into a change list
8. The change list is transmitted to the server
9. The changes to each entity is reflected to the entity on the server side
10. A pre flush snapshot is taken of each entity
11. The server side Flush() is called
12. Each flushed entity is examined to see if it has changed
13. Any changes are compiled into a change list
14. The change list is transmitted back to the client side
15. Any changes are reflected back to their client side objects

16. A snapshot is taken for each changed entity to reflect its updated state

As you can see, flushes are fairly intensive so it is always best to try to keep you sessions as small as possible. If you start to accumulate a lot of entities and collections you will find that flushes and queries will become slower as the session has to dig through a lot more items to find what it needs.

## **1.2 What Works and What Doesn't**

This technology is still quite immature and there are still quite a lot of things that are not working, however the list is getting shorter. There is also a growing list of tests that can be run against this library. Please note that NHibernate developers employ thousands of different types of entities that are queried in almost every way imaginable. There will be bugs in certain queries and operations that I have not been able to think about.

### **1.2.1 Working**

- HQL queries
- Criteria Queries
- Linq Queries
- Lazy Loaded entities and collections
- Updating entities
- Deleting entities
- Saving entities
- Flush change tracking
- Session concurrency (1 to 1 session mapping between client and server)
- Multiple simultaneous open sessions
- Loads(mostly)
- Session state information (mostly)
- Transfer over http and tcp (http not tested very well)

### **1.2.2 Not working**

- Transactions
- Filters
- Named queries
- Sql Queries
- Change tracking for collections
- A lot of single query methods

### **1.2.3 Not tested very well**

- Complex queries
- Updates and saves on complex nested object graphs

### **1.2.4 Methods working out of ISession**

Working	Works Somewhat	Does not work

`void Flush();`

Changes to entities are tracked, collections not implemented yet

`FlushMode FlushMode { get; set; }`

`CacheMode CacheMode { get;set;}`

`ISessionFactory SessionFactory { get; }`

`IDbConnection Connection { get; }`

`IDbConnection Disconnect();`

No actual DB connection available, thinking about doing a network db connection to bridge the gap.

`void Reconnect();`

Works on the connection level

`void Reconnect(IDbConnection connection);`

`IDbConnection Close();`

No actual DB connection available

`void CancelQuery();`

`bool IsOpen { get; }`

`bool IsConnected { get; }`

Works on the network connection only.

`bool IsDirty();`

`object GetIdentifier(object obj);`

`bool Contains(object obj);`

`void Evict(Object obj);`

`object Load(System.Type theType, object id, LockMode lockMode);`

`object Load(System.Type theType, object id);`

`T Load<T>(object id, LockMode lockMode);`

`T Load<T>(object id);`

`void Load(object obj, object id);`

void Replicate(object obj, ReplicationMode replicationMode);  
void Replicate(string entityName, object obj, ReplicationMode replicationMode);

No transient support, yet

object Save(object obj);  
void Save(object obj, object id);  
object Save(string entityName, object obj);

void SaveOrUpdate(object obj);  
void SaveOrUpdate(string entityName, object obj);

No transient support, yet

void Update(object obj);  
void Update(object obj, object id);  
void Update(string entityName, object obj);

No Transient Support yet

object Merge(object obj);  
object Merge(string entityName, object obj);

No Transient Support yet

void Persist(object obj);  
void Persist(string entityName, object obj);

No Transient Support yet

object SaveOrUpdateCopy(object obj);  
object SaveOrUpdateCopy(object obj, object id);

No Transient Support yet

void Delete(object obj);  
int Delete(string query);  
int Delete(string query, object value, IType type);  
int Delete(string query, object[] values, IType[] types);

IList Find(string query);  
IList Find(string query, object value, IType type);  
IList Find(string query, object[] values, IType[] types);

Obsolete methods,

IEnumerable Enumerable(string query);  
IEnumerable Enumerable(string query, object value, IType type);  
IEnumerable Enumerable(string query, object[] values, IType[] types);

ICollection Filter(object collection, string filter);  
ICollection Filter(object collection, string filter, object value, IType type);  
ICollection Filter(object collection, string filter, object[] values, IType[] types);

```
void Lock(object obj, LockMode lockMode);  
void Lock(string entityName, object obj, LockMode lockMode);  
LockMode GetCurrentLockMode(object obj);
```

```
void Refresh(object obj);  
void Refresh(object obj, LockMode lockMode);
```

```
ITransaction BeginTransaction();  
ITransaction BeginTransaction(IsolationLevel isolationLevel);  
ITransaction Transaction { get; }
```

```
ICriteria CreateCriteria(System.Type persistentClass);  
ICriteria CreateCriteria(System.Type persistentClass, string alias);
```

```
IQuery CreateQuery(string queryString);  
IQuery CreateFilter(object collection, string queryString);
```

```
IQuery GetNamedQuery(string queryName);  
IQuery CreateSQLQuery(string sql, string returnAlias, System.Type returnClass);  
IQuery CreateSQLQuery(string sql, string[] returnAliases, System.Type[] returnClasses);  
ISQLQuery CreateSQLQuery(string queryString);
```

```
void Clear();
```

```
object Get(System.Type clazz, object id);  
object Get(System.Type clazz, object id, LockMode lockMode);  
object Get(string entityName, object id);  
T Get<T>(object id);  
T Get<T>(object id, LockMode lockMode);
```

```
string GetEntityName(object obj);
```

```
IFilter EnableFilter(string filterName);  
IFilter GetEnabledFilter(string filterName);  
void DisableFilter(string filterName);
```

```
IMultiQuery CreateMultiQuery();
```

```
IMultiCriteria CreateMultiCriteria();
```

```
ISession SetBatchSize(int batchSize);
```

```
ISessionImplementor GetSessionImplementation();
```

The underlying session implementor is there but far from complete

```
ISessionStatistics Statistics { get; }
```

```
ISession GetSession(EntityMode entityMode);
```

## 2. Using NHibernate.Remote

I'm not going to insult your intelligence by going into minute detail on how to create a new VS project, compile, copy dll's and other trivial details. This documentation assumes that you are familiar with the details of how to setup compile, and deploy .net assemblies and projects.

If you are looking for setup and usage examples, you can also check out the "Tests" project in the subversion source code in addition to the examples that I will give here.

This section follows along with the NHibernate.Remote.Sample project that is in Subversion.

### 2.1 Server Side

I'm using a console project to host the NHibernate.Remote but it could also be hosted with a windows service, or even in IIS.

#### 2.1.1 References

GenericMessaging  
NHibernate 2.0.1.4000 (2.0.1GA)  
NHibernate.Remote 0.2.0.0  
NHibernate.Linq (Optional)  
System.ServiceModel (Needed for anything WCF)

#### 2.1.2 App.config

```
<configSections>
  <section name="nhibernateRemoteConfig"
type="NHibernate.Remote.Cfg.ConfigurationHandler, NHibernate.Remote"/>
  <section name="hibernate-configuration"
type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate"/>
</configSections>

<nhibernateRemoteConfig>
  <genericMessagingConfig>
    <channel>
      <transport address="net.tcp://localhost:4533/RemoteSession"
type="tcp"/>
      <settings heartbeatInterval="60"/>
      <limits maxReceivedMessageSize="5242880"
maxStringContentLength="5242880" maxArrayLength="5242880" />
    </channel>
  </genericMessagingConfig>
</nhibernateRemoteConfig>

<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
  <session-factory>
    <property name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
```

```

        <property
name="connection.provider">NHibernate.Connection.DriverConnectionProvider</pr
operty>
        <property name="connection.connection_string">Server=localhost;Initial
Catalog=Northwind;Integrated Security=SSPI;</property>
        <mapping assembly="Northwind.Entities"/>
    </session-factory>

</hibernate-configuration>

```

The options here are very self explanatory. Basically the NHibernate configuration can change to match your needs; this just shows the configuration that works with my configuration.

You will note here that the database points to Northwind. I have included a dump of this database from SQL2005 as a backup file. You can import this into your db if you want to run my tests or the sample. Alternatively, I think that MS has this available as a download somewhere as well.

### 2.1.3 Code

```

static void Main(string[] args)
{
    //optionally initialize log4net

    //Build your NHibernate session factory however you choose
    ISessionFactory factory = new
Configuration().Configure().BuildSessionFactory();

    //Open a host using the factory and wait for requests
    ServiceHost host = new RemoteNHibernateServiceHost(factory);

    host.Open();

    Console.WriteLine("Host is now running, press any key to close.");
    Console.ReadKey();

    //close the host
    host.Close();
    factory.Dispose();
}

```

The code here is also self explanatory. Do whatever you need to build your NHibernate session factory. Open a new remote NHibernate host using that factory, and then wait for requests. On closing just dispose of things.

That was easy!

## 2.2 Hosting in IIS

Ok so many of you are wondering how to host your service in IIS. This is possible and I will show you how to do it under IIS6. I have heard that IIS7 is even easier for hosting these kinds of things but I don't have server 2008 yet so this will have to do.

The references are the same as for the console host.

### 2.2.1 Web.config

The only thing that changes in this file is the transport config.

```
<transport address="" type="http"/>
```

If you leave the address blank then the service will pull the first address from the list of addresses your server is hosted against. If you want to hardcode the value in the address field then that value will override the hosted web server address. You can only ever host a WCF service on one address and NHibernate.Remote is no exception. The type of the transport must be http, at least this is the way it works in IIS6.

The other thing to note about the config, is that when hosting in a console application you had the option to build your session factory and configure it however you wanted in code. Well in IIS right now the only option is to configure your factory in the web.config. For most of you this probably works just fine, however for the portion of the population that needs to instantiate their own session factory, there is a workaround. You will need to extend RemoteNHibernateServiceHostFactory and override the BuildSessionFactory method. You can then use your custom factory in place of the regular one.

### 2.2.2 The .svc file

```
<%@ ServiceHost Language="C#"
    Debug="true"
    Service="NHibernateRemote"
    Factory="NHibernate.Remote.RemoteNHibernateServiceHostFactory,
    NHibernate.Remote" %>
```

Not much to note here except that the Service parameter doesn't matter at all since it is ignored by the factory anyways. The factory simply points the NHibernate.Remote host factory or your own custom factory if you choose to extend it.

That's all there is to hosting in IIS.

## 2.3 Client Side

The client is nearly as easy as the server.

### 2.3.1 References

GenericMessaging  
NHibernate 2.0.1.4000 (2.0.1GA)  
NHibernate.Remote 0.2.0.0  
NHibernate.Linq (Optional)  
System.ServiceModel (Needed for anything WCF)  
PostSharp.Core  
PostSharp.Laos

The only catch here is with the Post Sharp libraries. You will need to include them in your project, as well as making sure that they get copied locally. You can change this by going to properties for each one of these dlls in visual studio. Alternately you can install Post Sharp on your machine from the Post Sharp website.

### 2.3.2 App.Config

```
<configSections>
  <section name="nhibernateRemoteConfig"
type="NHibernate.Remote.Cfg.ConfigurationHandler, NHibernate.Remote"/>
  <section name="hibernate-configuration"
type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate"/>
</configSections>

<nhibernateRemoteConfig>
  <genericMessagingConfig>
    <channel>
      <transport address="net.tcp://localhost:4533/RemoteSession"
type="tcp"/>
      <settings heartbeatInterval="60"/>
      <limits maxReceivedMessageSize="5242880"
maxStringContentLength="5242880" maxArrayLength="5242880" />
    </channel>
  </genericMessagingConfig>
</nhibernateRemoteConfig>

<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
  <session-factory>
    <property name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
    <mapping assembly="Northwind.Entities"/>
  </session-factory>

</hibernate-configuration>
```

The config section is nearly identical to the server side. However there are some minor differences. You will notice that there is still a regular NHibernate configuration section, why? This is because NHibernate.Remote still needs a session factory on the client side to be able to manipulate entities. This session factory does not have a connection provider or a connection string making it essentially toothless.

Of course you need to make sure that the transport type and address are the same as the server side or there will be a connection failure.

### 2.3.4 Code

```
static void Main(string[] args)
{
    //setup an NHibernate session factory without a connection string
    ISessionFactoryImplementor nhFactory = new
Configuration().Configure().BuildSessionFactory() as
ISessionFactoryImplementor;

    //use the NHibernate factory implementation to create a remote session
factory
    IRemoteSessionFactory factory =
RemoteNHibernateConfig.CreateFromConfig().BuildSessionFactory(nhFactory);

    //Use the factory to open a session and do some work
    using (ISession session = factory.OpenSession())
    {
        IList<Customer> customers = session.CreateQuery("from
Customer").List<Customer>();

        Console.WriteLine(String.Format("Got {0} customers from server.",
customers.Count));
    }

    //Close
    Console.WriteLine("Press any key to close.");
    Console.ReadKey();

    factory.Dispose();

    nhFactory.Dispose();
}
```

Again, this is pretty simple stuff. Get an NHibernate session factory but with a slight twist. NHibernate.Remote needs the implementor in order to build a remote session factory so you will have to cast your session factory to that. Use your session factory implementor to create a remote session factory. Don't try to open a session with the NHibernate session factory or 1) You will get a local connection if you supplied a connection string or 2) it will fail because it has no connection. Instead use the IRemoteSessionFactory to open a session.

You can then use the session just like you would a regular NHibernate session (Well, actually there are quite a few things that don't work yet).

When you are finished with the session, close it and dispose of it. You should also do the same with the remote factory and your NHibernate factory.

Simple stuff.

## 3. Technical details

Sorry this part is not finished yet, I'm still working on the documentation.

Please feel free to comment or email me, but please bear in mind that this is very definitely a work in progress.

Daniel