

Worlds Together: IMS and .NET

Evgeni Liakhovich, IMS Developer
evgueni@us.ibm.com



Customers Speak



“Our company's current IT management “wisdom” seems to believe that Microsoft is the answer (...) Because we don't have a .NET way to access IMS DB, most of our agencies seem to be doing IMS DB extracts and loading the data into Sql Server, Access, etc. This process gives management an additional excuse to move off the mainframe (...) There are solutions that would definitely solve the problem, unfortunately they are very expensive (...) So.. long story short, we need a .NET solution to IMS DB (both on/offline) yesterday”

- IMS Customer

Customer Requirements

- IMS DBMS customers want to be able to access IMS data directly from .NET applications using SQL
- Today, .NET-based IMS customers who need access to IMS data from .NET applications are forced to:
 - × Set up a replication environment where IMS data is copied to a .NET accessible database server
 - × Implement data proxy solutions that don't offer direct connectivity
 - × Increase code path by implementing bridge solutions
 - × Use expensive 3rd party products
- All of these alternate paths mean either higher development costs, higher management costs, and/or higher runtime costs



Introducing

IBM IMS Data Provider for Microsoft .NET

- **IBM IMS Data Provider for Microsoft .NET**
 - a component of IMS Enterprise Suite
- This product enables standard ADO.NET SQL access to IMS data from .NET applications in a simple, fast, well proven way
 - Develop and reuse .NET applications (written in any .NET language, e.g. C#, VB, VC++) to access IMS data
 - Perform CRUD operations via SQL directly against IMS data
 - No need for intermediate steps/tools (such as DB2 stored procedures, web services, or 3rd party products) to access IMS databases from .NET



Microsoft ADO.NET and Data Providers

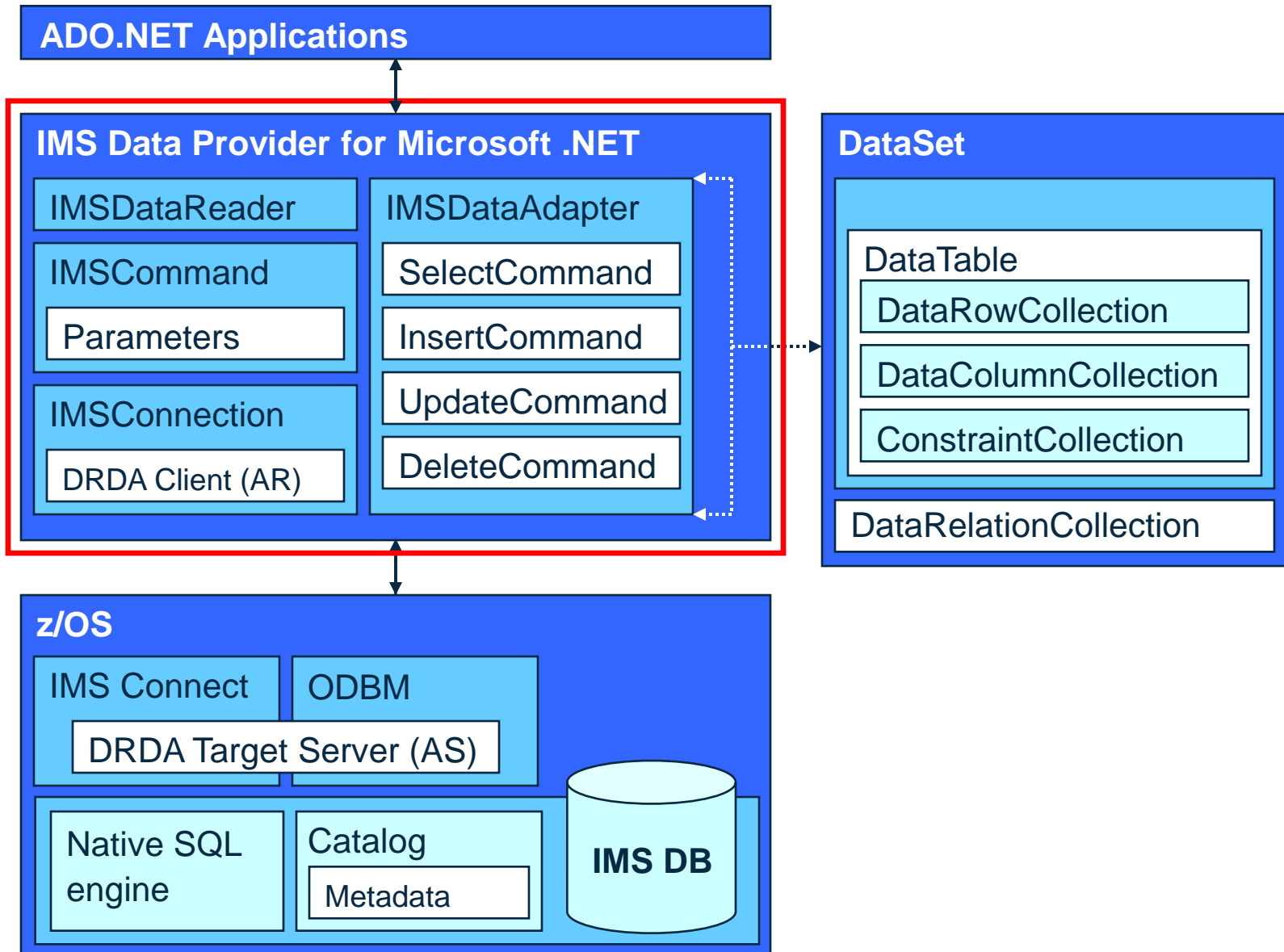
- Microsoft's development platform for Windows is known as the **.NET Framework**
 - Supports over **40** different programming languages. The most popular are **C#** and **Visual Basic**.
- The .NET Framework provides data access support through **ADO.NET**
- Your applications use databases through what's known as a **data provider**
 - Various database products include their own .NET data providers

Overview of IMS Data Provider

- The **IBM IMS Data Provider for Microsoft .NET** is a high performance, managed ADO.NET data provider created specifically for IMS database systems
- Designed for direct data manipulation and fast access to data
- Leverages IMS Catalog and IMS SQL engine
- Implemented according to ADO.NET data provider API specification
 - Enables standard, reusable code

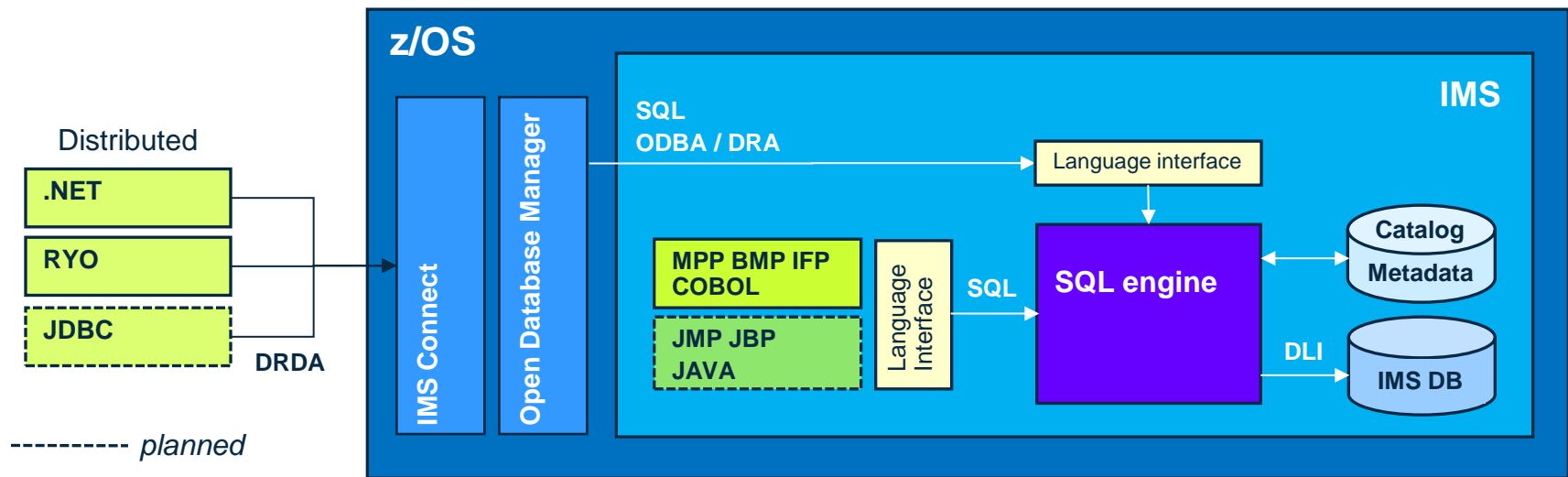
IBM IMS Data Provider for Microsoft .NET

IMS Data Provider Architecture



IMS SQL Support

- SQL Engine for COBOL and distributed applications (.NET/JDBC)
- Provides standard SQL keywords to easily access IMS data
 - ✓ SELECT, INSERT, UPDATE, DELETE
 - ✓ Uses Dynamic SQL programming model
 - ✓ Converts SQL statements to DLI calls
 - ✓ Supports a subset of SQL keywords that are currently supported by IMS Universal JDBC driver
- Uses database metadata in IMS Catalog
 - ✓ No need to generate metadata for use in applications



Connecting to IMS

- A database connection is established through the `IMSConnection` class:

1. Create a string that stores the connection parameters. Typical format:

```
Server=<ip address/hostname>:<port number>;  
Database=<PSB name>;  
Datastore=<Datastore value>;  
User ID=<userID>;  
Password=<password>;  
Connect Timeout=<Timeout value>;  
Pooling=<true or false>;  
Encryption=<true or false>;
```

2. Pass the connection string to the `IMSConnection` constructor. Example:

```
String connectionString =  
    "Server=12.34.56.78:5555;Database=SAMPLE";  
IMSConnection conn = new IMSConnection(connectionString);
```

3. Use the `Open` method to connect to the database identified in the connection string:

```
conn.Open();
```

C# Application Example (SELECT)

Example.cs X

```
using IBM.Data.IMS;

static void IMSReader()
{
    // Use connection string to configure connection properties
    IMSConnection connection = new IMSConnection("Data source = MyIMS,5555;
        Database = Insurance");
    // Establish connection to IMS database
    connection.Open();

    // Specify SQL query in the IMSCommand object
    IMSCommand command = new IMSCommand("SELECT * FROM PCB01.CUSTOMERS",
        connection);

    // Execute query via the DataReader object
    IMSDataReader reader = command.ExecuteReader();

    // Iterate through results and output on the screen
    while (reader.Read())
        Console.WriteLine(reader.GetString(0));

    // Close the reader
    reader.Close();

    // Close the connection
    connection.Close();
}
```

C# Application Example (INSERT)

```
Example.cs X
using IBM.Data.IMS;

static void IMSWriter()
{
    // Use connection string to configure connection properties
    IMSConnection connection = new IMSConnection("Data source = MyIMS,5555;
        Database=Insurance");
    // Establish connection to IMS database
    connection.Open();

    // Specify SQL command in the IMSCommand object
    IMSCommand command = new IMSCommand("INSERT INTO PCB01.CUSTOMERS (NAME,
        POLICY) VALUES ('EVGENI', 1210050000)", connection);

    // Execute command, return number of rows affected
    int i = command.ExecuteNonQuery();

    // Close the connection
    connection.Close();
}
```

- INSERT, UPDATE and DELETE commands are used identically

Parameters

- Applications can reference IMS SQL data type values as SQL statement parameters
 - '?' is used as parameter marker
- The `IMSParameter` object is used to represent a parameter to be added to a `IMSCommand` object
 - When specifying the data type value for the parameter, the data type values available in the `IBM.Data.IMS.IMSType` namespace must be used

Example.cs

```
String CommandText = "INSERT INTO PCB01.CUSTOMERS (NAME, POLICY) VALUES (?, ?)";
IMSCommand command = new IMSCommand(CommandText, conn);

// Create and prepare an SQL statement.
IMSParameter nameParam = new IMSParameter("NAME", IMSType.VARCHAR, 100);
IMSParameter policyParam = new IMSParameter("POLICY", IMSType.BIGINT);
nameParam.Value = "Evgeni Liakhovich";
policyParam.Value = 1210050000;
command.Parameters.Add(nameParam);
command.Parameters.Add(policyParam);

command.ExecuteNonQuery();
```

Transactions

- IMS Data provider supports **Local Transactions**
 - **IMSTransaction** object is responsible for rolling back and committing database transactions

```
Example.cs ×
IMSqlCommand command = connection.CreateCommand();
IMSTransaction transaction;

// Start a local transaction.
transaction = connection.BeginTransaction("SampleTransaction");
command.Transaction = transaction;

try
{
    command.CommandText = "INSERT INTO REGION (ID, NAME) VALUES (100, 'Portland')";
    command.ExecuteNonQuery();

    command.CommandText = "INSERT INTO REGION (ID, NAME) VALUES (200, 'Vegas')";
    command.ExecuteNonQuery();

    transaction.Commit(); // Attempt to commit the transaction.
}
catch (Exception ex)
{
    transaction.Rollback(); // Attempt to roll back the transaction.
}
```

Generic Coding

- .NET Framework outlines the "**generic coding**", or "**factory-based**" interface that is planned to be supported by IMS Data Provider
 - Facilitates generic ADO.NET application development, constant programming interface across different databases
 - When using this technique, proprietary class names, such as `IMSConnection`, are replaced with common names, such as `DbConnection`

```
Example.cs X
using IBM.Data.IMS;

static void GenericReader()
{
    DbProviderFactory factory = DbProviderFactories.GetFactory("IBM.Data.IMS");
    DbConnection connection = factory.CreateConnection();
    DbConnectionStringBuilder sb = factory.CreateConnectionStringBuilder();

    connection.ConnectionString = sb.ConnectionString;
    connection.Open();

    DbCommand command = new DbCommand("SELECT * FROM PCB.CUSTOMERS", connection);
    DbDataReader reader = command.ExecuteReader();
}
```

Connection Pooling

- When a connection is first opened against an IMS database, a connection pool is created. As connections are closed, they enter the pool, ready to be reused.
- The IMS Data Provider enables connection pooling by default
 - Note: You can turn connection pooling off using the **Pooling=false** connection string keyword/value pair
- You can control the behavior of the connection pool by setting connection string keywords for the following:
 - The minimum and maximum pool size (**MinPoolSize** and **MaxPoolSize**)
 - The length of time a connection can be idle before it is returned to the pool (**ConnectionLifetime**)

Data Types

- Supported IMS data types and their .NET equivalents

IMS Type	.NET Type	Method
TINYINT	Byte	GetByte
SMALLINT / INTEGER / BIGINT	Int16 / Int32 / Int64	GetInt16/GetInt32/GetInt64
DOUBLE / FLOAT	Double / Single	GetDouble / GetFloat
BIT	Boolean	GetBoolean
CHAR / VARCHAR	String	GetString
PACKEDDECIMAL / ZONEDDECIMAL	Decimal	GetDecimal
DATE / TIME / TIMESTAMP	DateTime	GetDateTime
BINARY	Byte[]	GetBytes

- No truncation or data loss occurs when using listed methods for corresponding data types. Other methods can often be used, however, data integrity cannot be ensured.

Reading Results

- Reading the result sets is done through a `IMSDataReader` object. `Read()` method is used to advance to the next row in the result set.
- The methods `GetString()`, `GetInt32()`, `GetDecimal()`, etc. are used to extract data from the individual columns
- `Close()` method is used to close the `IMSDataReader` object, which should always be done when reading the output is finished

Example.cs X

```
int row = 1;
while (reader.Read())
{
    //Print out Column Names if we are on the first row
    if (row == 1)
        for (int i = 0; i < reader.FieldCount; i++)
            Console.Write(reader.GetName(i) + "\t");
    Console.WriteLine();
    //Print out Column values converted to String
    for (int i = 0; i < reader.FieldCount; i++)
        Console.Write(reader.GetString(i) + "\t");
    row++;
}
```

Connected vs Disconnected Modes

- IMS Data Provider will support both **Connected** and **Disconnected** data access modes
 - The **connected** mode uses `DataReader` class and allows direct data manipulation and fast, forward-only, read-only access to data
 - IMS Data Provider also serves as a bridge between a data source and an ADO.NET `DataSet` interface
 - The `DataSet` object is central to supporting **disconnected**, data scenarios with ADO.NET
 - The `DataSet` is a memory-resident representation of data that provides a consistent relational programming model regardless of the data source
 - The **disconnected** architecture allows fetching data from IMS into a `DataSet`, manipulating data without holding database locks, and committing all changes at once to IMS when all work is finished

Choosing between DataReader and DataSet

- Consider the type of functionality that your application requires. Use a **DataSet** to do the following:
 - Cache data locally in your application so that you can **manipulate** it. If you only need to read the results of a query, the **DataReader** is the better choice.
 - Perform extensive processing on data without requiring an open connection to IMS, which frees the connection to be used by other clients.
 - Disconnected mode uses **optimistic concurrency**, good for environments with **low data contention**
- If you do not require the functionality provided by the **DataSet**, you can improve the performance of your application by using the **DataReader** to return your data in a forward-only, read-only manner.

Processing Metadata

- **Database Metadata** (or **Schema**) will be used to describe design and specification of the data structures in the database
- IMS Data Provider will offer two ways to read and process IMS metadata:
 - `GetSchemaTable()` method of the `IMSDataReader` class – result set level metadata
 - `GetSchema()` method of the `IMSConnection` class – collections of metadata about the entire database

Protecting Your Data

- Encrypt your data (AT-TLS standard will be supported):
 - Configure AT-TLS on the host side
 - Generate server certificate
 - Install server certificate on the client machine
 - Connection String: `Encrypt = true`
- Encrypt Configuration Files
 - Store and encrypt connection strings in configuration files, instead of embedding them in your application's code
- RACF Authentication
 - Connection string: `User ID=<userID>;
Password=<password>;`

Performance Tips

- Select only what you need
 - Applies to rows as well as columns
 - `IMSDataReader.Read` assumes all columns will be Get'd
- Connections – open late, close early
 - Maximize the use of connection pooling
- Other objects – close / dispose, set reference to null
 - `IMSDataReader.Close`, `IMSCommand.Dispose`
 - Allows the server to free locks and clean up cursors earlier
- Set `IMSCommand.FetchSize` to optimize network performance
 - If the end user consumes a lot of data, you will want your application to minimize the number of round-trip data fetches
- Qualify IMS segment names with PCB names
 - For example `PCB01.HOSPITAL`

GUI / Web Development

- Visual Studio is a powerful environment for developing GUI and web applications
 - Interactive applications that work with IMS data are easy to develop

My ASP.NET APPLICATION

Home About

WELCOME TO IMS AND ASP.NET!

	WARDNAME	WARDNO	HOSPNAME	HOSPCODE
Edit	GENERAL	0001	ALEXANDRIA	R1210010000A
Edit	SPECIAL	0002	ALEXANDRIA	R1210010000A
Edit	INTERNAL	0003	ALEXANDRIA	R1210010000A
Edit	SURGICAL	0004	ALEXANDRIA	R1210010000A
Edit	COSMETIC	0005	ALEXANDRIA	R1210010000A
Edit	INTENSIVE	0007	ALEXANDRIA	R1210010000A
Edit	GENERAL MED	0001	SANTA TERESA	R1210020000A
Edit	DERMATOLOGY	0002	SANTA TERESA	R1210020000A
Edit	PEDIATRICS	0003	SANTA TERESA	R1210020000A
Edit	ORTHOPEDICS	0004	SANTA TERESA	R1210020000A

1 2

Query: SELECT HOSPNAME,HOSPCODE,HOSPLL FROM PCB01.HO!

Name: OrderId

	HOSPNAME	HOSPCODE	HOSPLL
▶	ALEXANDRIA	R1210010000A	900
	SANTA TERESA	R1210020000A	900
	SANTA CLARA	R1210030000A	900
	NEW ENGLAND	R1210040000A	900
	SAINT VINCENT	R1210060000A	900
*			

Buttons: Fill, Insert, Update, Delete

IBM logo

Available now

ibm.com/ims → Downloads tab

IBM IMS

- Overview
- Products
- What's new?
- IMS User Groups
- Downloads**
- Resources

Downloads

IMS Enterprise Suite

Integration solutions and tooling that support open integration technologies, enable new application development, and extend access to IMS transactions and data. The IMS Enterprise Suite download includes the following components that run on Windows or Linux:

- **IMS Enterprise Suite Data Provider for Microsoft .NET**
- IMS Enterprise Suite SOAP Gateway
- IMS Enterprise Suite Connect APIs for Java and C
- IMS Enterprise Suite Explorer for Development
- IMS Enterprise Suite Java Message Service API

[Download now](#)

IMS TM Resource Adapter


Enables you to more easily create Java applications that either access IMS transactions or process callout requests from IMS applications over the Internet. The Message Format Service support for Service Oriented Architecture (MFS SOA) run time support is also embedded in the IMS TM Resource Adapter.

[Download now](#)

Contact IBM

Considering a purchase?

- [Email IBM](#)
- [Request a quote](#)
- [Or call us at: 1-877-426-3774](#)
Priority code: Portfolio

 **The Role of IMS in Today's Enterprise**
This new analyst report on IMS, by Colin White from BI Research, looks at the role of IMS in today's highly complex IT environment and examines how organizations can integrate IMS into an open and flexible enterprise IT infrastructure that can evolve as technology changes.

[Get the report \(448KB\)](#)

Related links

- [Success stories](#)
- [Events](#)
- [Training and certification](#)
- [Services](#)
- [Support](#)

Join the IMS conversation

IMS discussions, idea sharing, direct interactions with experts, and more

IMS Data Provider for Microsoft .NET installation

- Go to the **IMS Enterprise Suite download site** and log in.
- Select IMS Enterprise Suite Version 3.1 and click Continue.
- Select IMS Data Provider for Microsoft .NET and click Continue
 - IMS Data Provider for Microsoft .NET download page
- Select the IMS Data Provider for Microsoft .NET repository file
- Click **Download now** to download the selected files.
- Store the compressed repository file in a accessible location
- You must have IBM Installation Manager Version 1.5.3 or later installed

Getting Started

- Documentation:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims.net31.doc/net_intro.htm

- “Verifying installation” page is a good place to start
- Look for `getting_started.txt` and a `sample project` in the installation directory after installing the .NET Data Provider
- Video tutorials and demos on **YouTube**:
 - http://bit.ly/IMS_YouTube

System Requirements

- Software requirements
 - IMS DB v13, APARs PM96324 and PI05437
 - IMS Connect, ODBM
 - Catalog
 - .NET Framework 4.0
 - Windows XP, Windows 7
- Hardware requirements
 - For IMS DB - same as IMS v13
 - For .NET Data Provider and Visual Studio
 - Computer that has a 1.6GHz or faster processor
 - 1 GB (32 Bit) or 2 GB (64 Bit) RAM (Add 512 MB if running in a virtual machine)
 - 3GB of available hard disk space
- Tooling
 - Microsoft Visual Studio

Thank you!

Your feedback is important to us!

