# Why Datasnap 2010 is a toy library (http://www.sandon.it/node/57)

**Posted on:** Tue, 06/15/2010 - 21:01 **By:** ldsandon

To this StackOverflow question (http://stackoverflow.com/questions/3033180/delphi-datasap-in-http-proxy-server/3033394#3033394), I simply answered to forget D2010 Datasnap because IMHO it is only half-backed and lacks several needed features and is not flexible. Two people commented my answer was not helpful, and wrote Datasnap is absolutely flexible (later one comment was withdrawn). I am going to explain in details why the "new" Datasnap is not a usable solution for remoting in Delphi, and while the "old", DCOM based one is under several aspects far more powerful although DCOM is a complex beast to use, and Delphi doesn't exploit its power fully.

Datasnap 2010 offers two transport protocols, pure TCP and HTTP. They would be perfectly adequate for most scenarios (although it lacks an asynchronous transport, like Microsoft Messege Queues that in some scenarios would be really useful). But soon we find a big issue. The HTTP implementation does not support setting a proxy. Most business environments use a non trasparent proxy to control HTTP traffic. If the Datasnap server is inside the network, or the proxy is transparent, it will work. But if you have to go through a proxy you're out of luck. How could Embarcadero *forget* proxies do exist? Moreover the transpoort is handled by a TidHTTP component, which supports connection through a proxy. Were the programmers too busy or too lazy to overlook a basic feature? I do not know, but the result is Datasnap can't work through a proxy.

The transmitted data encoding is made using JSON. JSON was designed for JavaScript and the web. It is lighter then SOAP, but it is still a text encoding of data, JavaScript is not good at processing binary data. If you look at it just from the standard web application perspective, it is a good protocol. But Datasnap should have not been designed just with the standard web application in mind, an application that exchange a limited amount of data, mostly textual, on slower networks following user commands. A general remoting solution should be able to handle much heavier loads, applications working on gigabit (or faster) networks exchanging lots of data, maybe binary ones. Why spend time to encode and decode them to/from text streams? JSON is interoperable, true, but when the communication is between Delphi application, a binary transfer protocol may be far more efficient. For example WCF offers what it calls "compiled XML" to achieve better transfer capabilities.

And there is more. A good remoting solution should offer four basic capababilities. *Authentication*, *authorization*, *integrity* and *encryption*. Authentication ensure that *both* parties involved in the communication are who they say they are. *Authorization* ensure  that once authenticated, only permitted operation can be performed. *Integrity* ensures received data are the same data sent and were not tampered with even if they are transmitted in plain text, and *encryption* of course protects sensitive data.

Datasnap 2010 offers almost nothing of those basic capabilities. It supports HTTPS and server authentication, but it does not support *client authentication* but simple a user/password one. The TCP transport does not offer almost anything, only the simple user/password authentication provided. Usually strong authentication implies a third trusted party (a certificate authority, a domain controller,

a Kerberos KDC) to validate the two endpoints, but there's almost nothing in Datasnap 2010. As said, nothing beyond authenticating the server via HTTPS. There are some events fired when the connection is established, but they are too simple to allow for a complete handshacking that may require more than one roundtrip to establish authentication. And TLS/SSL is a general security protocol, it doesn't work together HTTP only. It can be implemented over plain TCP as well.
DCOM Datasnap uses DCOM facilities, and thereby uses Windows security (Kerberos, if you're in a domain) - it can authenticate not only the connection, but each call if needed.

Once an application connects to the server, it may not be able to call every server available function. A good remoting framework should be able to map an authenticated user to a set of allowed operations. That requires that each call carries an authentication token or the like, to allow the receving party to match it against an access list. Again, Datasnap 2010 has not such a feature, and adding it using filters may be a real challenge, because of the way filters are implemented and the layer they work at. A modern remoting framework should offer some way to create and store ACLs (LDAP, Active Directory, a local database, whatever), and match them against the calling user. An event fired before each function call with an authentication token where the call could be aborted would help... Again DCOM Datasnap has such a feature (because DCOM has), although it can be difficult to use beyond Delphi basic support.

Speaking about filters, we came to the last features: encryption, integrity and message authentication. Embarcadero people will tell you you can use filters to implement encryption and message integrity and authentication. That is simply not true. Sure, you can apply an encryption/decryption algorith to your data stream, but the issue are the encryption keys. Datasnap lacks a way to exchange (and change) session keys securely (again, it can require more than one roundtrip to establish, check for example the classic Diffie-Hellman algorithm)  and plugging such a feature in yourself is not easy at all, because Datasnap architecture was not designed to allow it.
If your application uses static keys never changed, embedded in the application code or stored on disk, especially on the client side, no matter how you try to protect them, you have a security hole. An HMAC algorithm, a good way to ensure message integrity and authentication, relies on a secret (and really "secret") key being exchanged too. Thereby the filter mechanism may be good for compression, but not to add proper and strong security to exchanged data. Again DCOM Datasnap offers this kind of functionality.

It's somewhat appaling that in 2010 Embarcadero totally overlooked the need to protect data properly while on the wire. I wonder what kind of application they had in mind when they designed Datasnap 2010. Data streams move more and more sensitive data, and their protection is no longer optional, and can't be ensured using a DIY approach. In security, DIY is usually a recipe for disaster. There are not many ways to implement security properly, and a little deviation from the proper road can easily expose sensitive data.

These major flaws makes me believe the actual implementation is just a toy one, and can't be used in any real application beyond simply basic needs. There are other issues, for example regarding asynchronous communication, scalability, service discovery, fault tolerance, clustering, and so on, but they could be advanced topics and I'll write about them another time.

I believe the basic flaw of this Datasnap is its underlying design. It designed over dbExpress, and it is implemented in a way that resembles too much calling database stored procedures (look how parameters are passed and remote functions invoked). They overlooked that many of the features above in a database are handled by the database itself (authentication, authorization) or the client library and its communication protocol (integrity, encryption). If you remove the database from the equation, you should add all those capabilities yourself. But Embarcadero pretends there's no need. But that's just a toy, then.

- Log in (http://www.sandon.it/user/login?destination=/comment/reply/node/57/comment_node_blog%23comment-form) to post comments