Andrea Ferendeles
aferende@hotmail.com


# .NET Sql Authorization Manager

*(NetSqlAzMan)*

by Andrea Ferendeles

***.NET Sql Authorization Manager is an authorization manager for .NET Framework 2.0/3.5 developed applications (smart-client/web). The authorizations storage is MS Sql Server 2000/2005.***

***NetSqlAzMan has been developed with LINQ to SQL.***

Andrea Ferendeles
aferende@hotmail.com

## Summary

# .NET Sql Authorization Manager

..... 1

NetSqlAzMan is Open Source ..... 4

Requirements ..... 4

Security within the applications ..... 5

Ms Authorization Manager (AzMan) vs .NET Sql Authorization Manager (NetSqlAzMan) ..... 7

The NetSqlAzMan structure ..... 9

Store and Store Group ..... 10

Application and Application Group ..... 13

Item Definitions ..... 15

Biz Rules ..... 16

Item Authorizations ..... 17

Database Users Custom-Authentication ..... 20

NetSqlAzMan is Delegate-compliant ..... 21

NetSqlAzMan is Time-dependant ..... 22

Authorization Attribute ..... 22

Applicative Delegation ..... 27

Manipulating the NetSqlAzMan Storage by .NET code ..... 36

ENS (Event Notification System) ..... 39

Building Applications with NetSqlAzMan ..... 41

Tutorial 1: CheckAccess inside Windows/Web Applications ..... 41

Tutorial 2: CheckAccessHelper ..... 43

Tutorial 3: UserPermissionCache ..... 44

Tutorial 4: StorageCache ..... 45

Tutorial 5: WCF Cache Service ..... 46

Conclusions ..... 48

*Thanks to* ..... *49*

*References* ..... *49*

Andrea Ferendeles
aferende@hotmail.com

# What NetSqlAzMan is

*Its name is definitely colourful but it's neither a super-hero, not a recommended tooth-paste from the best Italian dentists ☺.*
NetSqlAzMan *is the* .NET Sql Authorization Manager *short form and is an applicative authorization manager, that is, given an application user, what this user is authorized to do within that application.*

NetSqlAzMan *is for all* Microsoft .NET 2.0/3.5 *developers that need to manage loosely-coupled applicative authorizations, that is, weakly coupled with source code, in a light and fast way having all these authorizations in a relational database such as* MS Sql Server (2000/2005/2008/MSDE/Express).

*For all of you who already knows* MS Authorization Manager (AzMan) *or* ADAM *(*Active Directory Application Mode*) then you are in the right place … but you have to expect a lot of innovations.*

## NetSqlAzMan is Open Source

NetSqlAzMan *is an Open Source project, accommodated by the CodePlex.com community.*
*From the following link it's possible to download both the source code (C#.net) and the setup package (.MSI) for the x32/x64 platform:* http://netsqlazman.codeplex.com

***Requirements***

- *Windows 2000 or later*
- *.NET Framework 3.5 (http://www.microsoft.com/downloads/details.aspx?familyid=AB99342F-5D1A-413D-8319-81DA479AB0D7&displaylang=en)*
- *MMC (Microsoft Management Console) 3.0 (http://support.microsoft.com/kb/907265/en-us)*
- *Sql Server 2000/2005/2008/MSDE/Express (http://www.microsoft.com/express/sql/download/)*

Andrea Ferendeles
aferende@hotmail.com

## Security within the applications

*When we talk about applicative security it's possibile to write symbolically the following expression:*

***{ Security } = { Authentication } + { Authorization }***

*where for Authentication we mean the credentials phase (who are you?) and for Authorization, what a user is granted to do and what he isn't (what can you do?). Generally, the authentication moment happens at first, and it's enough to assess once the user credentials to avoid repeating every time (at run-time) the same operation (think the "logon" moment on your pc).*

*The second phase supposes that the user credentials have already been verified and therefore the user identity is "assessed" and "well-known". At this point, the Authorization system is able to reply to a user request to execute a determined operation, for example, retrieving all the necessary informations from any data source (for example File System's ACL ). To do an example, you can think when you go to the airport to take a flight.*

*When you are at the Check-In, you are asked of an airplane ticket and an identity document so that you can have a security ticket back, the "boarding pass" (Authentication). At this point, and only after having got the boarding pass, you can go to the right exit of your flight. Once, inside the boarding zone, you aren't authorized to take any flight but exclusively the one indicated on the boarding; such rule is completed at the boarding moment (Authorization) and only with the right "ticket". I excuse myself if this example has turned out banal but it's very important to do clarity, in order to understand in which context* NetSqlAzMan *is placed.*

*Well,* NetSqlAzMan *is just what stores and preserves the authorizations, giving us an answer like "yes, autorized" or "no, not authorized" to anyone that makes a request (applications).*

NetSqlAzMan *is leaned instead on* MS Windows *operating system in order to carry out the authentication phase (Kerberos / NTLM).*

*We try to understand with a concrete example, the mechanism of the authorizations in an applicative context. We imagine a managerial application of any Company. Inside this Company we can identify some "Business Roles"such as: the administrator, the leaders, the secretaries, the employers etc. and some "Applicative Roles" that is groups of persons that can do the same operations in that Application. Suppose that the application ,you are writing, has functions about business accounting management, warehouse management and so on.*

*This application has to work with different user levels. As example, consider the "view – end-year budget" function, that lists in detail all the costs and revenues of an entire year of activity. It's reasonable to say that such operation must be granted only to the company administrator and maybe to some leaders but not to the employers. Moreover it's reasonable to think that the administrator or one of his leaders wants "to delegate" this operation to one of his secretaries, or grant (temporarly) the permission for such operations.*

*Now, how it's possibile to realize such an application, without "wire" the following instructions inside the source code:*

***If (user = "Jack" or user = "Michael") then***

***(authorized)***

***Else***

***(not authorized)***

*First of all, we should separate the "single customer" from the "group of customer" idea, that is, a set of users to which an operation will be granted. At design time, we'll decide which user will belong to which group:*

***If (user belongs to "Administrators" group) then***

***(authorized)***

***Else***

***(not authorized)***

*So we are introducing a too strong association between "groups" and what "they can do".*

*What will happen if tomorrow we decide to move a function from one group to another?!*

*Then we try to operate another process and say:*

***If (user has been authorized to do that X operation) then***

***(authorized)***

***Else***

***(not authorized)***

*You're doing now the easiest thing , that is, to ask yourself if that user can or cannot do such function, without taking in consideration his name or his group/role. The only link aspect betweeen the application and the authorizations repository is the process name. All these names together defines "a contract" and, this is binding; if you change a process name in the application or in the repository, it's necessary to change its name in the other part, too.*

*The unique* user *identification is provided by the Authentication system (Windows), while the Authorization system will answer "yes" or "no" to the* "he has been authorized to do that X operation" *question. It's therefore necessary to have a table with the "process name" and the relative "yes" or "no" for each user/group in any repository. The* NetSqlAzMan *repository is called* Storage *and is hosted by a* Sql Server *database (the setup package contains the Sql Script for the version 2000 or later).*

Andrea Ferendeles
aferende@hotmail.com

## Ms Authorization Manager (AzMan) vs .NET Sql Authorization Manager (NetSqlAzMan)

*As pointed out before, an analogous* Microsoft *product already exists and is called* Authorization Manager *(*AzMan*); AzMan is present, by default, in W*indows Server 2003 *and, through the* Admin Pack *setup*, *in* Windows XP.

*The important difference between* AzMan *and* NetSqlAzMan *is that the first is Role-based, that is, based on the belonging - Role concept and the operations container in each role, while the second is Item-based (or if you prefer Operation-based), that is users or users group or group of groups that can or cannot belong to Roles or execute such Task and/or Operations (*Items*).*

*Here the most important features and differences between the two products:*

***Ms AzMan:***

- *It's COM.*
- *It's equipped by a MMC 2.0 (COM) console.*
- *Its storage can be an XML file or ADAM (Active Directory Application Mode – è un LDAP).*
- *It's role-based.*
- *It supports static/dynamic applicative groups, members/not-members.*
- *Structure based on Roles → Tasks → Operations. (Hierarchical Roles and Tasks , none Operations).*
- *Authorizations can be added only to Roles.*
- *It doesn't implement the "delegate" concept.*
- *It doesn't manage authorizations "in the time".*
- *It doesn't trigger events.*
- *The only type of authorization is "Allow".*
  *(to "deny" it needs to remove the user/group from his Role).*
- *It supports Scripting / Biz rules.*
- *It supports Active Directory users/groups and ADAM users.*

***NetSqlAzMan:***

- *It's .NET 3.5.*
- *It's equipped by a MMC 3.0 (.NET) console.*
- *Its storage is a* Sql Server *database(2000/MSDE/2005/Express).*
- *It's based on* LINQ to SQL *technology.*
- *It's Item-based.*
- *Structure based on Roles → Tasks → Operations. (all hierarchical ones).*
- *Authorizations can be added to Roles, Task and Operations.*

- *It supports static/dynamic applicative groups, members/not-members.*
- *LDAP query testing directly from console.*
- *It's time-dependant.*
- *It's delegate-compliant.*
- *It triggers events (ENS).*
- *It supports 4 authorization types:*
    - Allow with delegation *(authorized and authorized to delegate).*
    - Allow *(authorized).*
    - Deny *(not authorized).*
    - Neutral *(neutral permission, it depends on higher level Item permission).*
- *Hierarchical authorizations.*
- *It supports Scripting / Biz rules (compiled in .NET – C# - VB – and not interpreted)*
- *It supports Active Directory users/groups and custom users defined in SQL Server Database.*

Andrea Ferendeles
aferende@hotmail.com

## The NetSqlAzMan structure

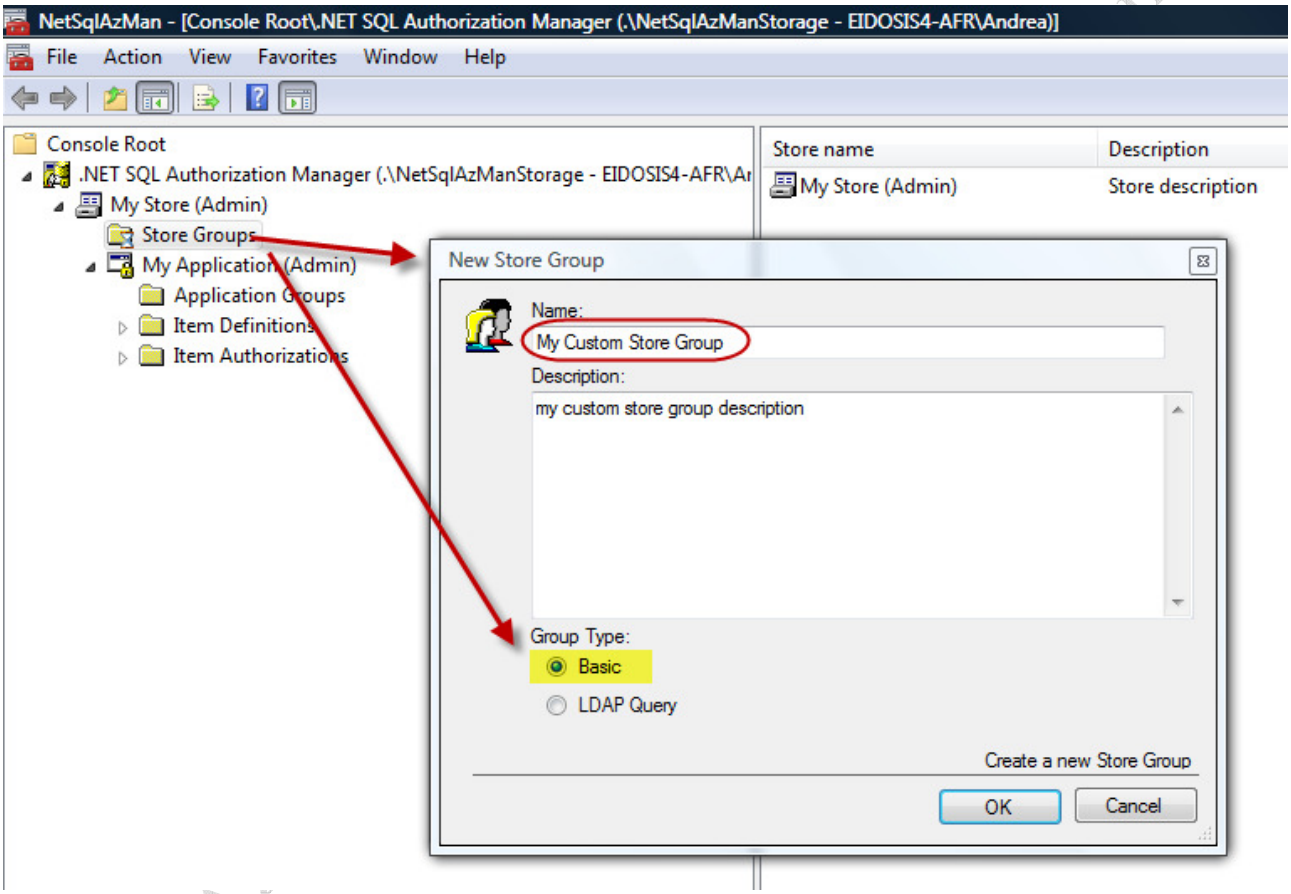
*You see the following structure both in the* Storage *and visually when you open the console (Start – run – "*NetSqlAzMan.msc*") :*

- ***Storage Sql Server (db)***
  - ***Store 1 (logical)***
    - ***Store Groups (Basic/LDAP, members/not-members)***
      - ***Store Group 1***
      - ***Store Group 2***
    - ***Application 1***
      - ***Application Groups (Bbasic/LDAP, members/not-members)***
        - ***Application Group 1***
        - ***Application Group 2***
      - ***Item Definitions***
        - ***Role Definitions***
          - ***Role 1***
        - ***Task Definitions***
          - ***Task 1***
        - ***Operation Definitions***
          - ***Operation 1***
      - ***Item Authorizations***
        - ***Role Authorizations***
          - ***Authorizations for Role 1***
        - ***Task Authorizations***
          - ***Authorizations for Task 1***
        - ***Operation Authorizations***
          - ***Authorizations for Operation 1***
    - ***Application 2***
      - ***…***
  - ***Store 2***
  - ***…***

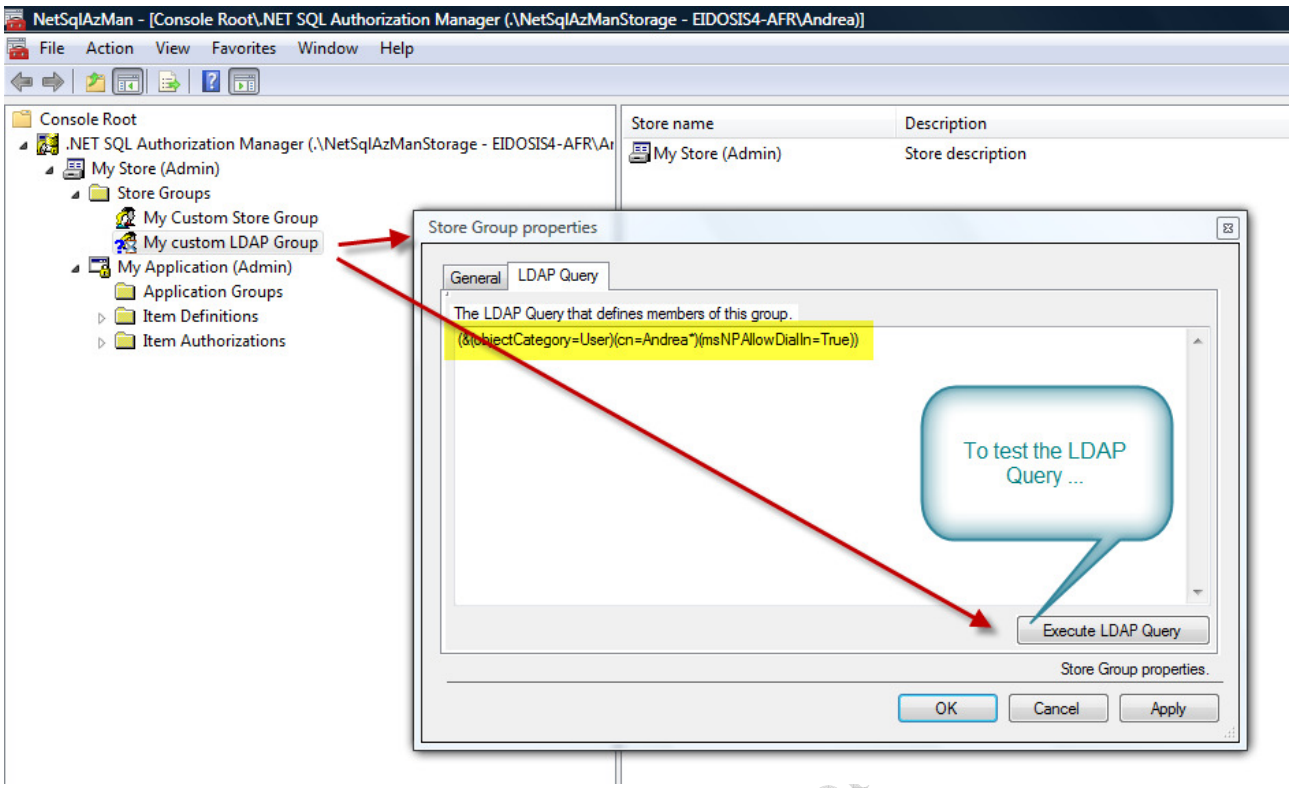
## Store and Store Group

*About* Store *we mean a logical grouping of applications. At this level it's possibile to define two object types:* Store Group *and* Application.

Store Groups *are applicative groups of:* Windows *Users/Groups, other* Store Groups *(such as COM+ roles or* Windows *security groups); each group can be* Basic *or* LDAP *type, that is,statically defined, by direct selection, or dynamically defined through a* LDAP *query, that* NetSqlAzMan *resolves at run-time. In this way it's possible, for example, to create an* Active Directory *users group with a name that begins with "Andrea" and Dial-In permissions. In* ***picture 1*** *we see how to create a* Store Group*; in* ***print 1*** *there is a* LDAP *query example and in* ***picture 2*** *how to execute that* LDAP *query directly from* NetSqlAzMan *console.*

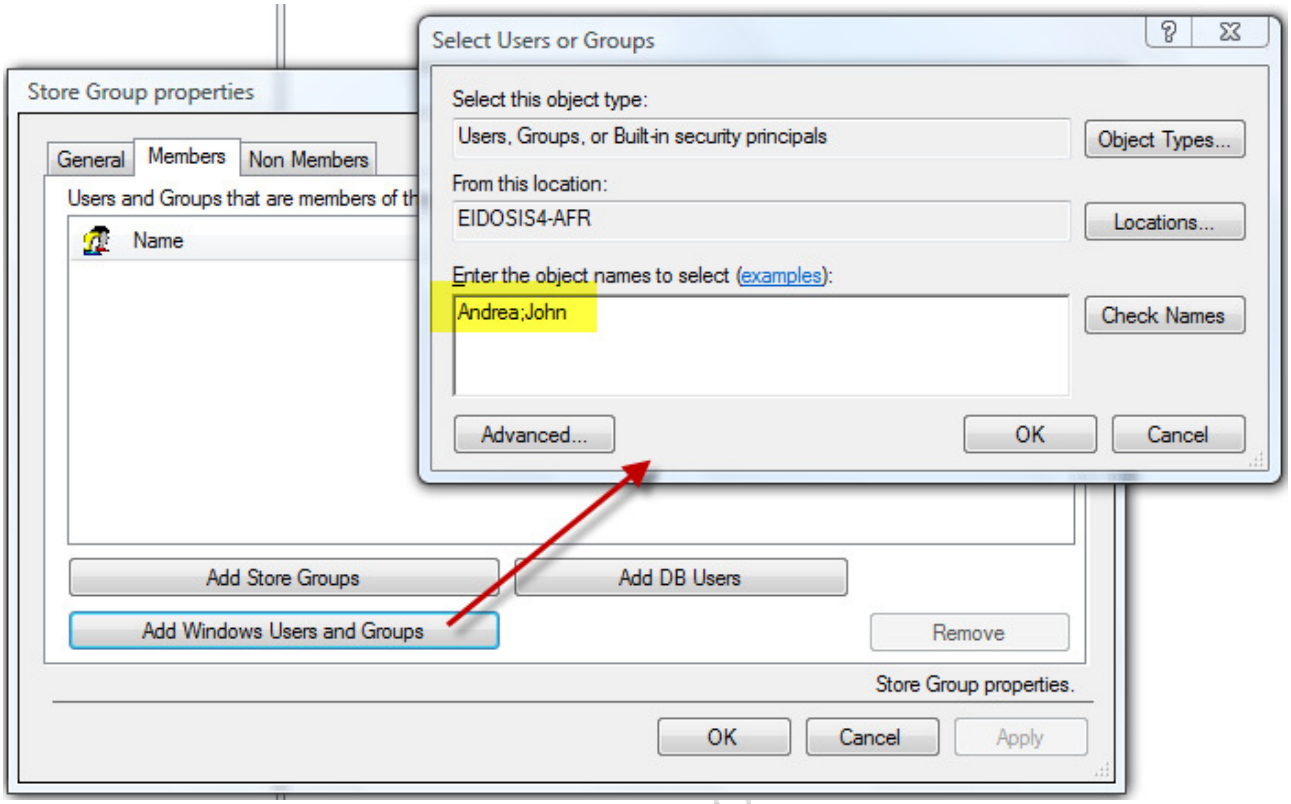



***Picture 1***



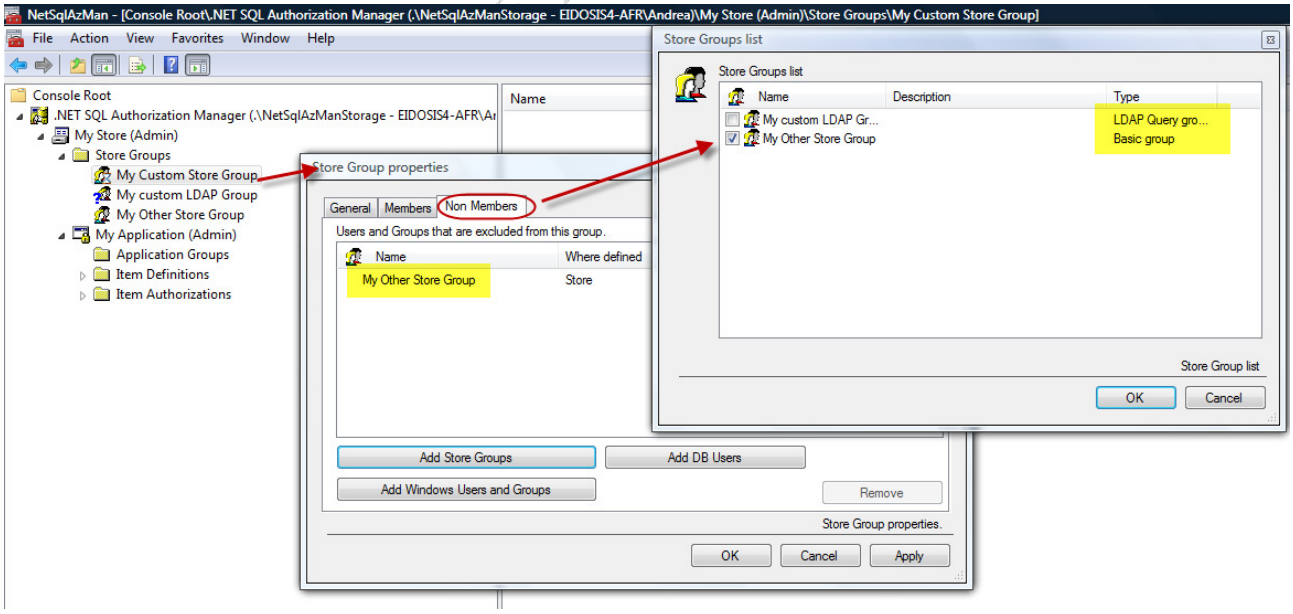

***Picture 2***

*Moreover, the "Basic" Groups are constituted by* Members *and* Not-Members *that is users/groups/*Store Groups *that must be "considered" when creating the* Store Group *while the* not-members *are users/groups/*Store Groups *that must* <u>not</u> *be considered. In other words, the members list that get part of a* Basic *group is given by { Members } – { Not-Members}. Since that members/not-members can be, at its turn, others* Store Group, *it's possible to realize powerful and complex recursive groups. In* ***pictures 3 and 4*** *we see the* Members *and* Non-Members *definition of one* Basic *group.*




***Picture 3***

*The* Store Groups *have a restricted visibility at* Store *level, so to be visibile from all the* Applications *contained in the Store.*




***Picture 4***

*The* Store Group *is the ideal in order to implement the concept of "Business Role" that is a grouping of persons in a Company who carry out one determined function with different rights from the rights that every role will have in every application; here some examples of business roles: "Managing", "Employees", "Responsible UO", etc….*

***Print 1**:*

***LDAP query to include all the Active Directory users whose name begins with Andrea and Dial-In permissions.***

*(&(objectCategory=user)(cn=Andrea*)(msNPAllowDialin=TRUE))*

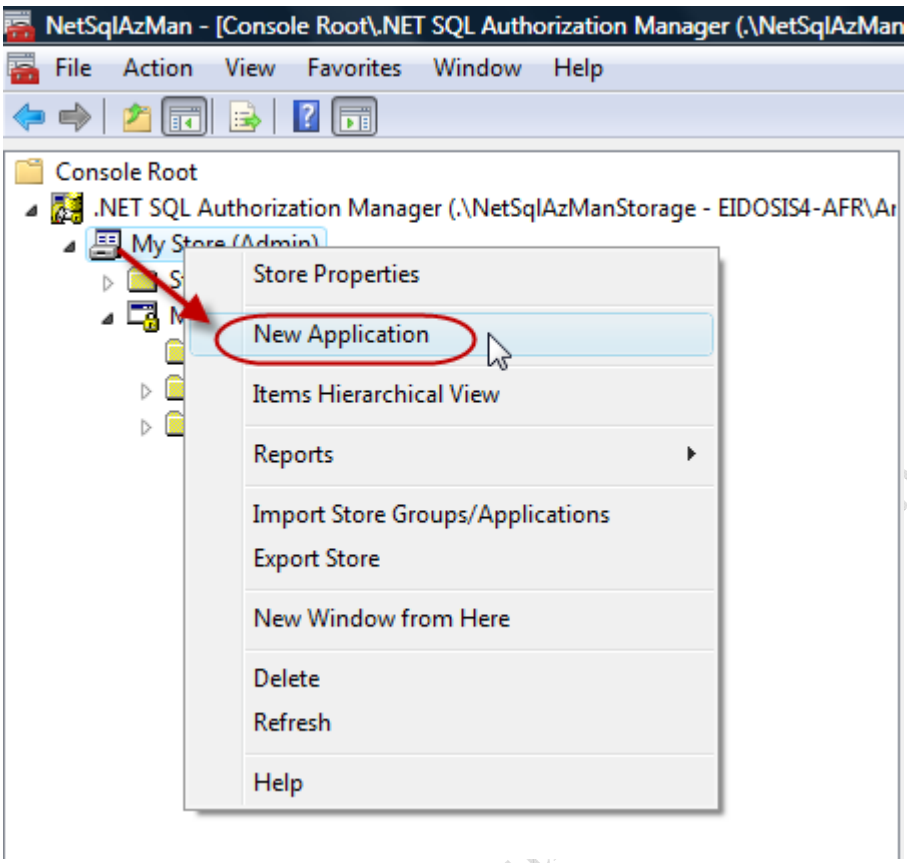
***Optionally you can include the RootDSE as query prefix and between [ ... ]:***

*[RootDSE:OU=My OU, DC=My Domain, DC=ext](&(objectCategory=user)(cn=Andrea*)(msNPAllowDialin=TRUE))*

*For all of you interested to know more about the syntax and potentiality of LDAP queries can read this interesting* Microsoft *article about the* LDAP *queries syntax:*
*http://www.microsoft.com/technet/prodtechnol/exchange/2003/insider/ldapquery.mspx*.

## *Application and Application Group*

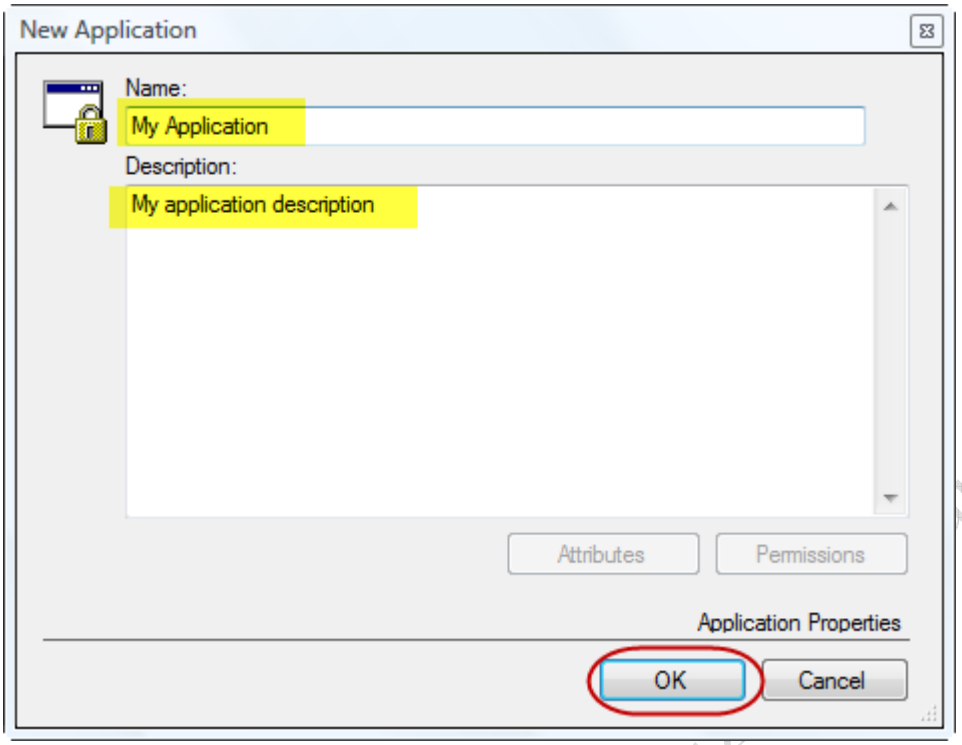
*For* Application *we mean that one that will request at run-time the* NetSqlAzMan *storage; here the definition is purely logical and so it doesn't require any association with the type or name of the real application. To avoid any confusion it's better to assign the same name and maybe writing a little description about what the application does in the appropriate gap as* ***pictures 5 and 6*** *show.*




***Picture 5***

*An* Application Group *is identical to a Store Group except that its scoping (visibility) is restricted only for the* Application *where is defined; an* Application Group *may consider a* Store Group *among its members/not-members but the vice versa cannot never happen.*

*The* Application Group *is the ideal to represent "Applicative Roles".*




***Picture 6***

## *Item Definitions*

*Every* Application *contains the definition of* Item*. Item contains the types* Role*,* Task *and* Operation

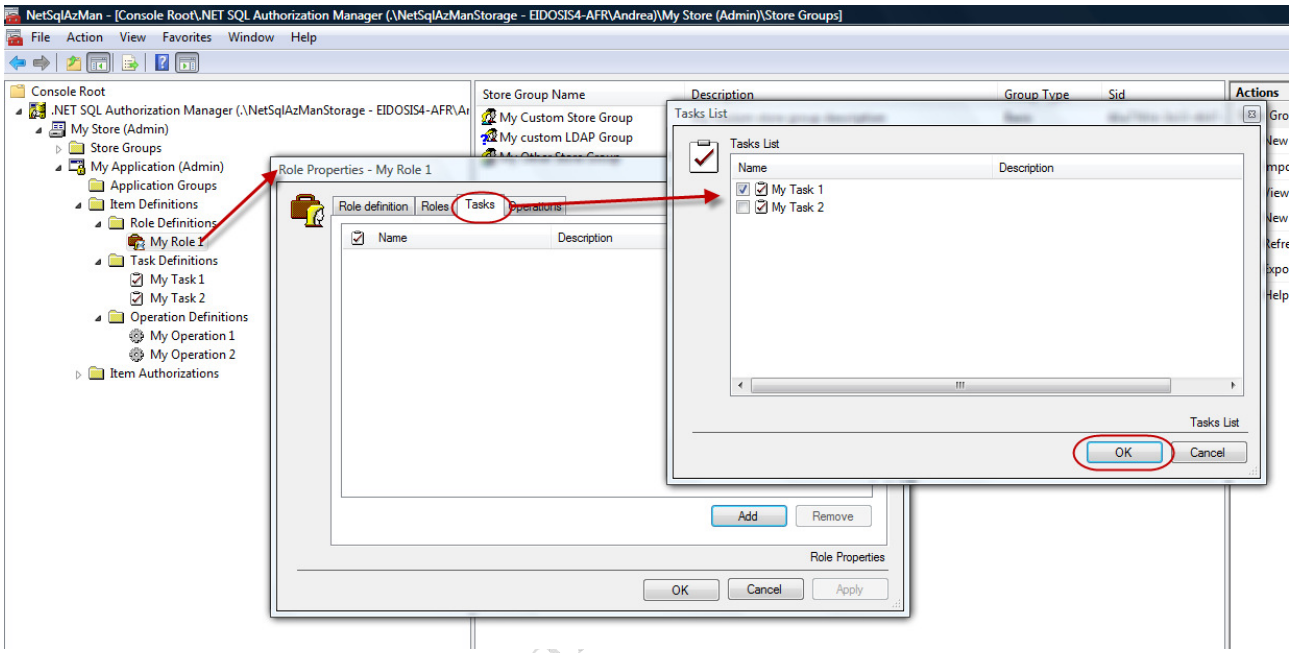
*Logical meanings of items type are defined as follow:*

- Role*: is a group of "users" that can do the same operations;*
  - *A* Role *can have members type such as* Role, Task, Operation*.*
  - *Samples of* Role *are: "Administrator", "General Director", "Product Manager".*
- Task*: is a logical macro-functionality of the Application*
  - *A* Task *can have members type such as* Task, Operation*.*
  - *Samples of* Task *are: "Insert", "Update", "View Report".*
- Operation*: is a micro- functionality of the Application*
  - *An* Operation *can have only* Operation *members type.*
  - *Samples of* Operation *are: "Add new user", "Update user", "View Report 1".*

*It usually happens that doing a certain operation, it automatically implies doing other ones of a lower level, or this idea can be used to assign authorizations in a hierarchical way.*

*If we consider, just as an example, "Delete" and "Insert" operations and assume that a users group is authorized to "delete", another one can "insert" and other groups are authorized to both "delete" and "insert".*

*In this case is necessary to create a "Modify" Task and two "Insert" and "Delete" Operations therefore change the* Task *properties(from the console) and say that it's constituted by two* Operations *(members* Item*). This fact will be important when you'll assign authorizations to these* Items*, because the* Task *rights (container* Item*) wil be inherited from* Operation *(*Item *members) but the vice versa will not happen (towards bottom); the one that will be authorized to "modify" will be implicitly authorized to "insert" and "delete" too. In* ***picture 7*** *there's a sample about this hierarchy.*




***Picture 7***

*When we'll finish creating all the* Operation *and, aggregate in logical* Task *of the Application, we can finally say what each applicative Role can do. We create as many* Role *as they are indicated in the application analysis and we'll specify for each Role which* Tasks *are part of it.*

## *Biz Rules*

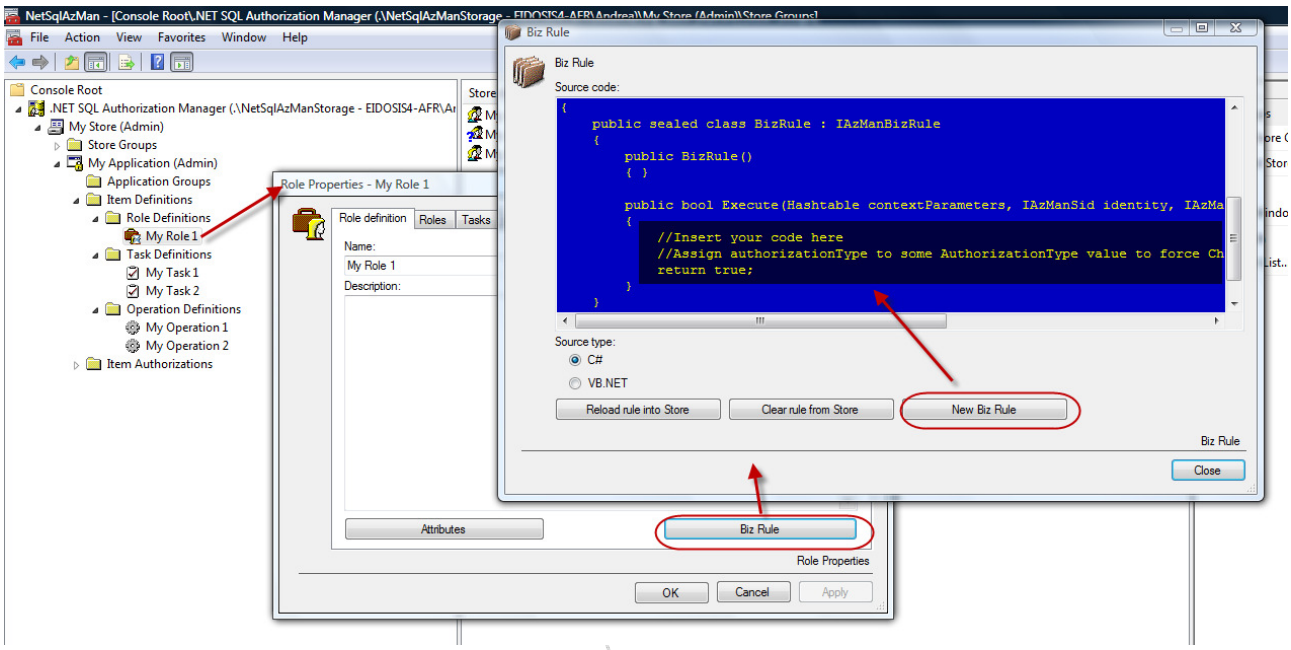
*For each Item is possibile to define a Business Rule (Biz Rule, that is logical rule). A Biz Rule has to determine if the authorization, eventually assigned for a user that is doing access check (CheckAccess) has to be considered or not. In othe words, this is a way to determine the belongings to an Item or not at Run-Time.*

*To define a Biz Rule is sufficient to create an Item, therefore confirm the new added from the Snap-In, access the Item properties and press the "Biz Rule" botton.*

*At this point we can write the .NET source code in C#/VB directly in the rule definitions window, or you can (advised) choose a different language and click on "New Biz Rule", then copy the rule "template" and edit it in a more comfortable environment such as VS.NET 2005.*

*When we finish writing our source code (and without syntactic errors) you paste the Biz Rule code and press the "Reload Biz Rule" botton. In that moment NetSqlAzMan compiles the rule and generates a .NET assembly in full rule and copies its binary image inside the "BizRules" sql table.*

*At Run-Time … the compiled assembly wil be read again from the db (and put it in cache for all the session time) and then executed. If you observe the Execute method firm, its return type is simply a boolean one that, when true makes CheckAccess method to continue otherwise it's interrupted (inside the Item hierarchy). The true difference between MS AzMan and NetSqlAzMan, about the Biz Rule, is the possibility to write Biz Rule in .NET code instead using Scripting (jscript/vbscript) and consequently a great increase of performance at run-time.*




## Item Authorizations

*Once we've defined all the* Item *(*Role, Task, Operation*) and created a right hierarchy among them, we should be worried about "to fill up"these containers with real users/groups and therefore "who … can do … what", indicating for everyone their permissions.*

*"Who" can be one of the following objects:*

- *A* Windows *user*
- *A* Windows *group*
- *A* Store Group
- *An* Application Group

*"Can do" can be one of the following authorizations:*

- Allow with delegation
- Allow

- Deny
- Neutral

*"What" will be one of the* Item *definined before:*

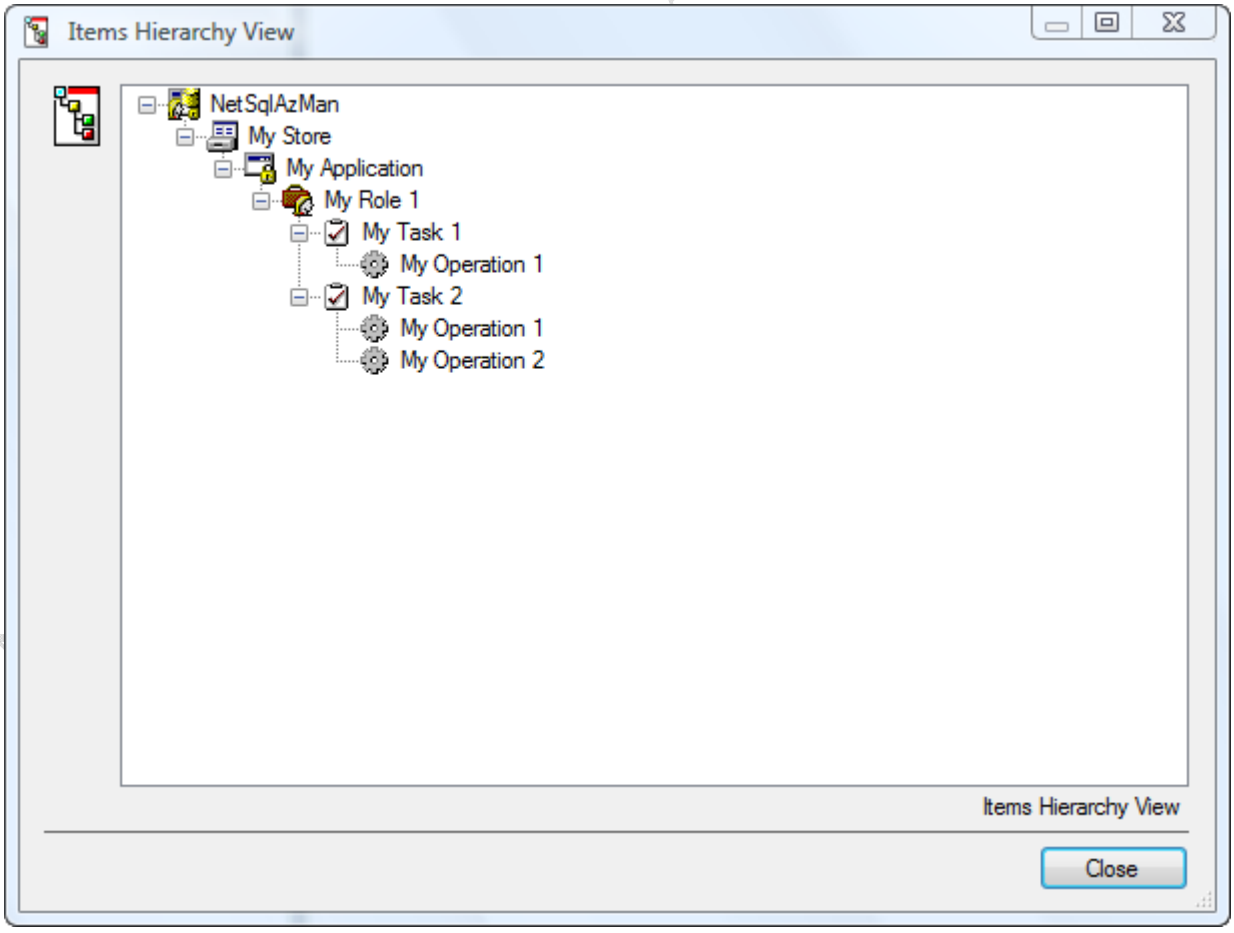
- Role
- Task
- Operation

*If the* Item *hierarchy has been correctly built , usually we sould be able to assign permissions only to Roles. When an application will ask* NetSqlAzMan *… "can I do an X Operation ?", its run-time will verify whether that operation belongs to a* Task *, that in turn belongs to a* Role *of which the application user takes part.*



***Note 1:***

***If in the* NetSqlAzMan *console you select one or more* Application*, or at least the entire* Store*, it's possible to view the* Item *hierarchy by clicking the right botton mouse and choosing the "Items Hierarchical View" menu (picture 11).***



*We now try to give a meaning to these four available authorizations, just remember that in NetSqlAzMan knows the idea of "delegate", that is a user that delegates another one to do an operation originally allowed only for the first one.*

*In a particular way that user that has got the "*Allow with delegation*" authorization can surely do that operation and moreover he can delegate one or more users to do the same operation originally granted to him. This delegate mechanism , for security reasons , cannot be extended beyond the first level. That means that the delegated user cannot delegate another user to do the same operation (*Allow with delegation *doesn't propagate).*

*The* Allow *right grants the authorizations to do that determined* Item *without delegate right.*

*The* Deny *right denies,on the contrary, every authorization while a* Neutral *type authorization is ,indeed, neutral, that is neither "yes" or "not", but it's the higher level* Item *to decide about.*

*The* Neutral *authorizations exist with their administrative scope and allow to maintain the "who"-"what" association in the store without saying if "what" can or cannot be done . A typical use of a* Neutral *authorization and for that* Item *which permissions continually change, and therefore it would be tedious, repetitively, to add and delete authorization subjects from the* Storage*; it's better to leave it there and modify only the permission when necessary.*

*One last consideration about the* Deny *authorization type: if for an* Item*, a user is authorized to a* Deny *permission, this user cannot surely do such operation and not even one of the eventually subordinate operations (sons* Item*) even if one of these is authorized to* Allow *o* Allow with delegation *permisssion; as it happens for* File System *and* Sql Server *authorizations , the more restrictive permission is successful. Viceversa, if for an* Item *a user is authorized to both* Allow *and* Allow with delegation *permissions*, *the less restrictive permission is successful that is* Allow with delegation*.*

*Attention to this affirmation, that is true only for authorizations allowed on the same* Item *and doesn't for hierarchical* Item*;* Allow with delegation *doesn't propagate on sons* Item*.*

*In* ***picture 8*** *the console view where you can set the authorizations.*

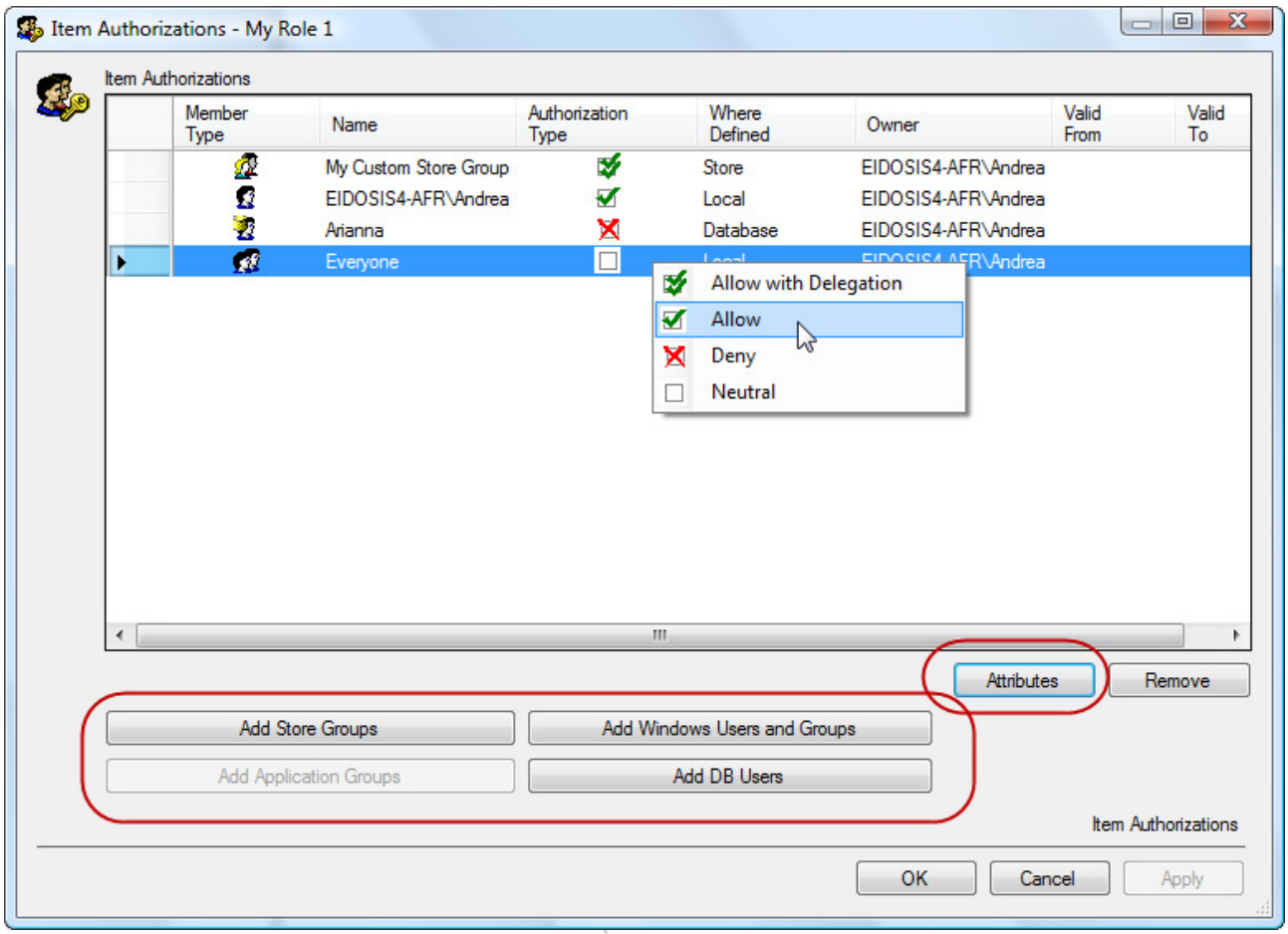



***Figura 8***

## *Database Users Custom-Authentication*

*From the* NetSqlAzMan *ver. 1.3.0.0 is possible to execute the check-access not only through Windows authentication but even through custom authentication for defined users in a MS Sql Server database.*

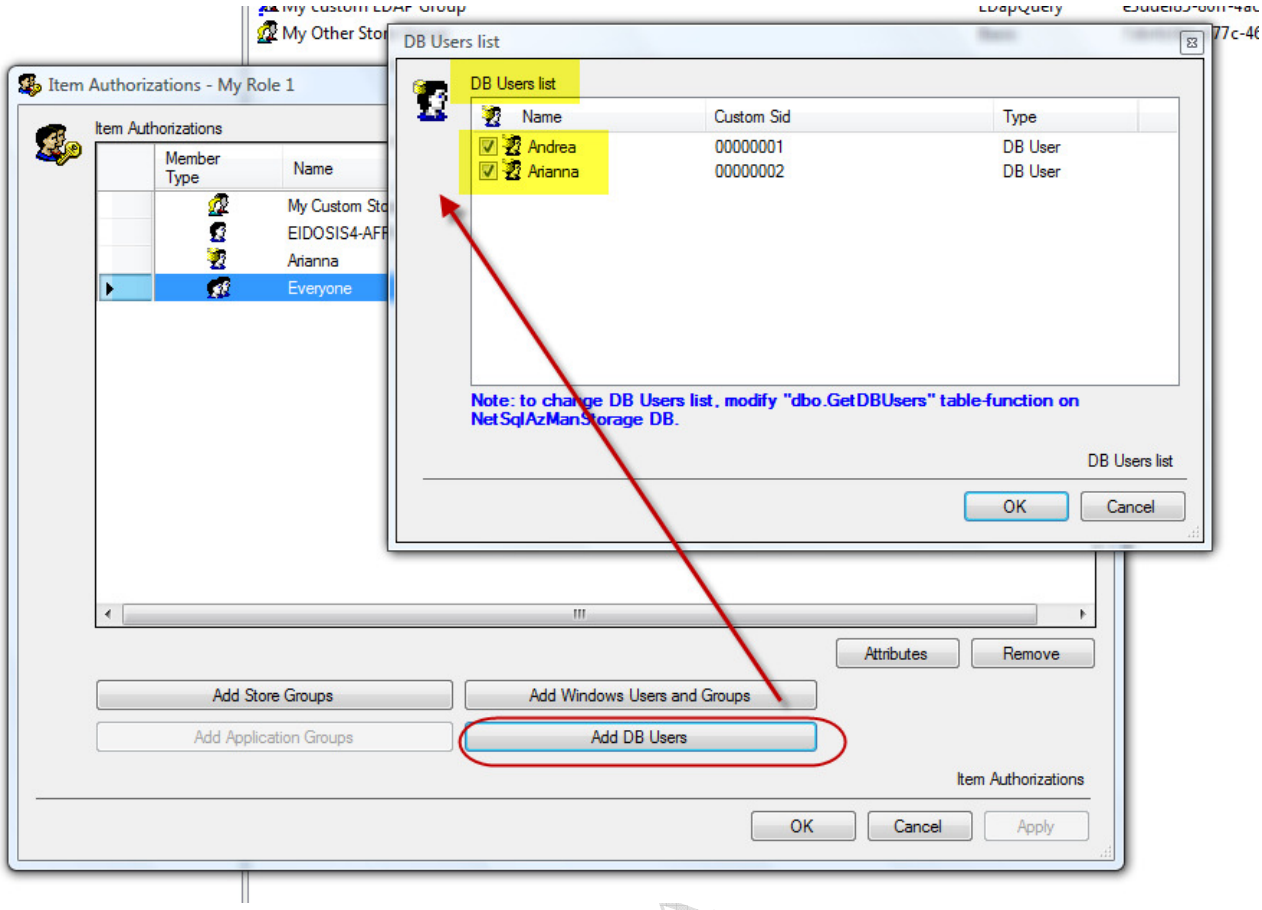
*This new functionality exists because of the* **dbo.GetDBUsers** *table-function, defined in the NetSqlAzMan storage Database storage. This table-function has to return a list of users defined through a value pair:* CustomSid, Username (varbinary(85), varchar(255)*).*

*It's very important to modify, before starting to use this functionality, the* GetDBUsers *table-function to correctly the list of users from any users table (even from a different Database), previously defined.*

*At run-time we find 2 methods to retrieve such list (or just a specific user) through the* **GetDbUsers/GetDBUser** *functions, defined in* IAzManStorage, IAzManStore *e* IAzManApplication *object type.*

*Finally , for the check-access you should use a specific overload of the CheckAccess function that accepts in input an* **IAzManDBUser** *object type, returned by the over cited methods.*

*In the following picture we see a screenshot that shows an added DB User to which assign the skilled authorizations.*




*Finally a* **DB User**, *can be added to a* Store Group, *to an* Application Group *and directly as an authorization member in the* Item Authorization.

## *NetSqlAzMan is Delegate-compliant*

*When an authorization is granted by the* NetSqlAzMan *Administrator, through the administrative console, we are talking properly about "Authorization". The same mechanism can be used during run-time by special users that allow other users to do a determined operation in their place. In this case we are talking about "Delegate".*

*In order to delegate, the delegant user must have the* Allow with delegation *permission directly on the* Item *and must belong to* Sql Server: NetSqlAzMan_Users *role. In the Sql* Storage *are present in fact 3 (three) different* Database Roles:

- NetSqlAzMan_Administrators *(full control)*
- NetSqlAzMan_Users *(only reading and delegate permission on* Item *with* Allow with delegation *permission)*
- NetSqlAzMan_Readers *(only reading)*

*Further on we'll see how to use the* Item.CreateDelegateAuthorization *method of the* NetSqlAzMan API.

## NetSqlAzMan is Time-dependant

*Each authorization can "be lavished" for an indetermined or determined time. The two* Valid From *e* Valid To *fields in the authorizations window, indicate respectively the starting and expiring date of the same authorization. Arranging this characteristic with the possibility to add more than an authorization for subject and for different temporal intervals, in addition the authorizations chain and the permission type (allow or deny) is possible to set authorizations that change during passing time. You can say, for example, that a "u1" user can execute the "x"* Item *only from the 1st-january-2006 to 30 rd-june-2006 and then from the1st-january-2007 to 30rd-june-2007. In these two time gaps you can exclude specific periods, adding other permissions for smaller periods and even of* Deny *type.*

*The same matter is for delegations, because they're authorizations in all respects (but set by different* Owner *than Administrator).*

## Authorization Attribute

*The last ring of the chain of this structure is represented by the* Authorization Attribute *that is all the attributes of an authorization . Physically an attribute is represented by a simple brace "key-value" , both of string type. For each* Authorization *is possibile to define infinite attributes with the only rule that the* Attribute *name (the* key*) must be unique for that authorization. Vice versa we can define another attribute with the same* key *for another authorization.*

*The* Authorization Attribute *scope is that one to concur with the worse level added than the custom information for every single authorization, increasing the granularity permissions level.*

*A typical use of attributes is that one for a user profile data. We suppose that a "u1" user has the "Project manager" role for an IT Company about a certain "p1" project; we imagine moreover that in the same Company it's been developed an application that allows all the project-managers to monitor the progress state about their own projects. Suppose that the application provides also a "SAL Check" (working progress state) operation. What does it happen if the "u1" user wants to delegate a "u2" user, a trusty person of his development team, to check ,in his stead, the "p1" project state? A delegation from "u1" to "u2" means that "u2 is authorized to do the SAL Check operation at "u1" place but he wouldn't know the "…for the only p1 project" information.*

*The "project-p1" (key-value) attribute added to the delegate authorization from "u1" to "u2" solves such problem; but the fact that NetSqlAzMan, in front of such a "u2" request "can I execute the SAL Check item ?" will reply* Allow*; then the application has to read the eventual* Authorization Attribute *and sort the projects list to use for the operation. Other attribute examples are given by : "Authorization Date", "membership UO ", "Area", "Department", etc….*

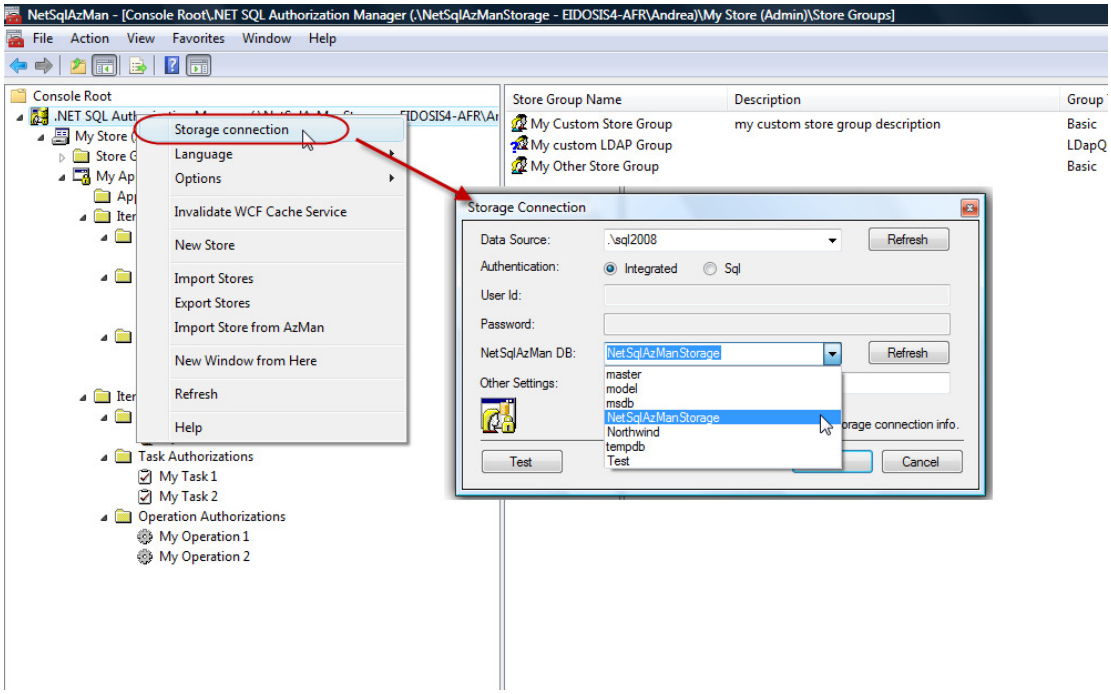
# NetSqlAzMan Snap-In and DataBase Sql Server creation

*You'are going to see how to implement physically all that we've said till now, through the* NetSqlAzMan.*Snap-In. First of all, you have to create a new* Sql Server *database; I suggest the* NetSqlAzManStorage *name but anything else is ok, then execute the appropriate script that you'll find in the* NetSqlAzMan *setup directory (there are both Sql Server 2000 and 2005 versions), be careful to select, as active database, the one just created (by* Query Analyzer *or* Sql Management Studio*) before starting the script.*

*Once the database has been created, the console can be started in 3 different ways:*

- *Start – Programs* - .NET Sql Authorization Manager – NET Sql Authorization Manager Console
- *Start – Run* – NetSqlAzMan.msc
- *Start – Run* – mmc *and then Add/remove Snap-In, then choose* .NET Sql Authorization Manager *from the list and then click on the Add botton.*

*Now we've to say* NetSqlAzMan *which* Storage *is to manage – **picture 9**; this operation can be done by right clicking on the* .NET Sql Authorization manager *node and then choosing the* Storage connection *menu. We then specify the server name and eventually the* Sql Server *instance on which we've created the database and executed the sql script, the authentication type (Sql or Windows), the* Storage *name and more eventual added parameters that will be part of the connection string* ADO.NET *(as ex. "Enlist = false;").*

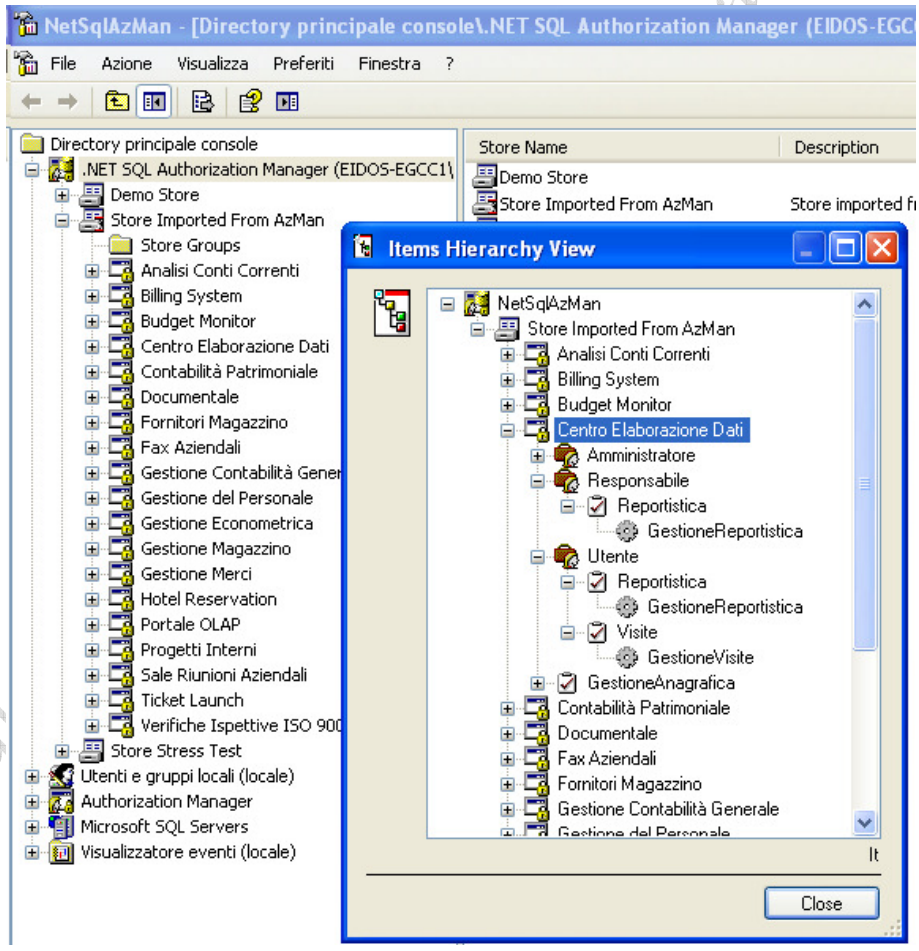



***Picture 9***

**Note 2:**

**NetSqlAzMan can work in two different ways: "Administrator" and "Developer"; the first one doesn't regard local Users/Groups/Well Know SIDs but only the Domain or Forest Active Directory Users/Groups/Well Know SIDs and doesn't allow the Operation manipulation (because considered at developers use); the second one regards both . The "Developer" way has been done just for the development time, when the Deployment environment isn't available or not even known. In this way the developer has to define from which are the Operations that make up each Task (previously defined by the analyst). To change this way it's necessari to use the console and, by right clicking on the.NET Sql Authorization Manager node, choose the Options menu.**

*At this point we're ready to create* Store, Store Group, Application, Application Group, Role, Task, Operation, Authorization *and* Authorization Attribute *by right clicking* – New Store, New Application, *etc…. In* ***picture 10*** *an example of* Store.




***Picture 10***

*When we've finished, we save the console so that next time we don't have to supply again the same connection data about the* NetSqlAzMan Storage.

# NetSqlAzMan API – Il Run-Time Engine

*Once the structure has been created from the* NetSqlAzMan *console, we see how the applications can enquire the* Storage *through the run-time engine API. You create then our application (smart client, web, etc…) with* VS.NET 2005 *and add a reference to the .NET* NetSqlAzMan.dll *Assembly. This assembly should be visibile in the components list of the "add reference"window; if it doesn't, look for it in the* NetSqlAzMan *setup folder.*

*At the beginning of our source code, we add two using/imports rules to declare using the* NetSqlAzMan *and* NetSqlAzMan.Interfaces *namespaces. Now all the necessary code to enquire the storage is just the following, as shown in* ***print 2****:*

***Print 2****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

...

string cs = "Data Source=(local);Initial Catalog = NetSqlAzManStorage;Integrated
Secuirty = SSPI;";

IAzManStorage storage = new SqlAzManStorage(cs);

System.Security.Principal.WindowsIdentity identity =
System.Security.Principal.WindowsIdentity.GetCurrent();

//For each Operation …

//Can I do "My Operation" ?

AuthorizationType authorization = storage.CheckAccess("My Store", "My
Application", "My Operation", identity, DateTime.Now, true);

switch (authorization)

{

    case AuthorizationType.AllowWithDelegation:

        //Yes, I can ... and I can delegate

        break;

    case AuthorizationType.Allow:

        //Yes, I can

        break;

    case AuthorizationType.Deny:
```

```
    case AuthorizationType.Neutral:
      //No, I cannot
      break;
}
```

VB.NET

```
Imports NetSqlAzMan
Imports NetSqlAzMan.Interfaces
...
Dim cs As String = "Data Source=(local);Initial Catalog =
NetSqlAzManStorage;Integrated Secuirty = SSPI;"
Dim storage As IAzManStorage = New SqlAzManStorage(cs)
Dim identity As System.Security.Principal.WindowsIdentity =
System.Security.Principal.WindowsIdentity.GetCurrent()
'For each Operation …
'Can I do "My Operation" ?
Dim authorization As AuthorizationType = storage.CheckAccess("My Store", "My
Application", "My Operation", identity, DateTime.Now, True)
Select Case authorization
    Case AuthorizationType.AllowWithDelegation
        'Yes, I can ... and I can delegate
    Case AuthorizationType.Allow
        'Yes, I can
    Case AuthorizationType.Deny Or AuthorizationType.Neutral
        'No, I cannot
End Select
```

*First of all we've to create a* NetSqlAzManStorage *class instance, supplying the Sql Server connection string to the constructor, then returning the user identity, that is working with the application.*

*To return the* WindowsIdentity *for an ASP.NET application we use the* LogonUserIdentity *property of the* HttpRequest *class of the* .aspx *page; if, instead we are working for a Smart-Client application we should use the* System.Security.Principal.WindowsIdentity.GetCurrent() *static method.*

*It's now sufficient to supply the* CheckAccess *method with the following input parameters:*

- StoreName *(*System.String*): the store name*

- ApplicationName *(*System.String*): the application name*
- ItemName *(*System.String*): the Item name (a* Role*, a* Task *or an* Operation*).*
- windowsIdentity *(*System.Security.Principal.WindowsIdentity*): the user identity*
- ValidFor *(*System.DateTime*):* DateTime *that indicates for which point in time the check is requested (typically* DateTime.Now*).*
- OperationsOnly *(*System.Boolean*):* true *to indicate that the* Item *must be necessarily an* Operation*,* false *otherwise (if you want to check the access to a* Task *or a* Role*).*

*The* CheckAccess *method answer will be of* NetSqlAzMan.AuthorizationType *enum type and its meaning will be one of the following:*

- AuthorizationType.AllowWithDelegation*: allowed access with delegation right.*
- AuthorizationType.Allow*: allowed access without delegation right.*
- AuthorizationType.Deny*: denied access.*
- AuthorizationType.Neutral*: neutral access (so denied).*

## *Applicative Delegation*

*If we want to implement in our application an* .aspx *page or a specific* Form Windows *to allow the users to delegate, we should have ,first, the following informations:*

- *The* Item *(*Role, Task, Operation*) delegation object;*
- *Who is the "delegant" user;*
- *Who is the "delegated" user;*
- *The validity time of the "delegation" (from … to);*
- *The authorization type to assign (Allow or Deny).*

*all informations to pass to the* NetSqlAzMan.SqlAzManItem.CreateDelegateAuthorization(…) *method.*

*From the first* NetSqlAzManStorage *class, we can "navigate" in the* NetSqlAzMan *DOM using the* GetXXX() *accessories methods or simplier using an indexer, as shown in* ***prints 3 and 4****.*

***Print 3****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;
```

```
...

string cs = "Data Source=(local);Initial Catalog = NetSqlAzManStorage;Integrated
Secuirty = SSPI;";

IAzManStorage storage = new SqlAzManStorage(cs);

storage.OpenConnection();

IAzManStore store = storage.GetStore("My Store");

IAzManApplication application = store.GetApplication("My Application");

IAzManItem operation = application.GetItem("My Operation"); //Or "My Task" Or
"My Role"

//...

storage.CloseConnection();
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces

...

Dim cs As String = "Data Source=(local);Initial Catalog =
NetSqlAzManStorage;Integrated Secuirty = SSPI;"

Dim storage As IAzManStorage = New SqlAzManStorage(cs)

storage.OpenConnection()

Dim store As IAzManStore = storage.GetStore("My Store")

Dim application As IAzManApplication = store.GetApplication("My Application")

Dim operation As IAzManItem = application.GetItem("My Operation") 'Or "My Task"
Or "My Role"

'...

storage.CloseConnection()
```

***Print 4**:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

...
```

```
string cs = "Data Source=(local);Initial Catalog = NetSqlAzManStorage;Integrated Secuirty = SSPI;";

IAzManStorage storage = new SqlAzManStorage(cs);

IAzManItem operation = storage["My Store"]["My Application"]["My Operation"];
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces

...

Dim cs As String = "Data Source=(local);Initial Catalog = NetSqlAzManStorage;Integrated Secuirty = SSPI;"

Dim storage As IAzManStorage = New SqlAzManStorage(cs)

Dim operation As IAzManItem = storage("My Store")("My Application")("My Operation")
```

*It's always possible and even better recommended to get, at run-time, an* Item *list defined in the* Storage, *but we must consider only that ones with* AllowWithDelegation *permission, as shown in* ***print 5****, because only with this specific permission is possible to delegate.*

***Print 5****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

...

string cs = "Data Source=(local);Initial Catalog = NetSqlAzManStorage;Integrated Secuirty = SSPI;";

IAzManStorage storage = new SqlAzManStorage(cs);

foreach (IAzManItem item in storage["My Store"]["My Application"].GetItems(ItemType.Role))

{

    foreach (IAzManAuthorization auth in item.GetAuthorizations())

    {

        if (auth.AuthorizationType == AuthorizationType.AllowWithDelegation)
```

```
        {
            //On this "item" can delegate
        }
    }
}
```

VB.NET

```
Imports NetSqlAzMan
Imports NetSqlAzMan.Interfaces
...
Dim cs As String = "Data Source=(local);Initial Catalog =
NetSqlAzManStorage;Integrated Secuirty = SSPI;"
Dim storage As IAzManStorage = New SqlAzManStorage(cs)
For Each item As IAzManItem In storage("My Store")("My
Application").GetItems(ItemType.Role)
    For Each auth As IAzManAuthorization In item.GetAuthorizations()
        If auth.AuthorizationType = AuthorizationType.AllowWithDelegation Then
            'On this "item" can delegate
        End If
    Next
Next
```

*After indicating the delegation object* Item *and the delegant user identity (same procedure shown in* ***print 2****) we should think about the "delegated" user, the one to allow the right to do the* Item *in our stead. Such identity is represented only by the* SID *(*Security IDentifier*) of a* Windows *user.*

SID *are guarded by the* Active Directory *infrastructure to which the application belongs to; to get a* SID *we must have the corresponding* Login *and generally a table of combinations is necessary between the user name, for example (name and surname), and the* Login *(DOMAIN\User). The .NET Framework 2.0 , fortunately, supplies two new and important classes to do such operation:* System.Security.Principal.NTAccount *and* System.Security.Principal.SecurityIdentifier*, that together allow to have an user* SID *, just given its* Login *and/or viceversa;* ***print 6*** *shows an example.*

***Print 6****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

using System.Security.Principal;

...

//Retrieve SID from NTAccount

NTAccount ntAccount1 = new NTAccount("MYDOMAIN", "username");

SecurityIdentifier SIDofNTAccount1 =
(SecurityIdentifier)ntAccount1.Translate(typeof(SecurityIdentifier));

string SSid = SIDofNTAccount1.Value; //S-X-XXXX-XXXX-XXXX-XXXXXXXX


//Retrieve NTAccount from SID

SecurityIdentifier SID2 = new SecurityIdentifier("S-1-1-0"); //Well Know SID of
Everyone

NTAccount ntAccount2 = (NTAccount)SID2.Translate(typeof(NTAccount));

string accountName = ntAccount2.Value; //Everyone
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces

Imports System.Security.Principal

...

'Retrieve SID from NTAccount

Dim ntAccount1 As NTAccount = New NTAccount("MYDOMAIN", "username")

Dim SIDofNTAccount1 As SecurityIdentifier =
DirectCast(ntAccount1.Translate(GetType(SecurityIdentifier)),
SecurityIdentifier)

Dim SSid As String = SIDofNTAccount1.Value 'S-X-XXXX-XXXX-XXXX-XXXXXXXX


'Retrieve NTAccount from SID

Dim SID2 As SecurityIdentifier = New SecurityIdentifier("S-1-1-0") 'Well Know
SID of Everyone

Dim ntAccount2 As NTAccount =
DirectCast(SID2.Translate(GetType(SecurityIdentifier)), NTAccount)
```

```
Dim accountName As String = ntAccount2.Value 'Everyone
```

*Then the* SID *will be passed as parameter to the* NetSqlAzMan.SqlAzManSID *constructor class.*

*The last two parametrs are* System.DateTime? *type (can be null) and represent the start and end date/time of delegation validity. If one of them or both have a null value, that means that the delegation has no expiry date or without validity start or indetermined time.* ***Print 7*** *shows an example of applicative delegation.*

***Print 7****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

using System.Security.Principal;

...

IAzManItem item = ...;

WindowsIdentity delegatingIdentity = WindowsIdentity.GetCurrent();

NTAccount delegatedUserNTAccount = new
NTAccount("MYDOMAIN","delegatedusername");

SecurityIdentifier delegatedUserSID =
(SecurityIdentifier)delegatedUserNTAccount.Translate(typeof(SecurityIdentifier))
;

IAzManSid delegatedUserAzManSID = new SqlAzManSID(delegatedUserSID);

DateTime? validFrom = DateTime.Now;

DateTime? validTo = new DateTime?(); //No expiration date

IAzManAuthorization del = item.CreateDelegateAuthorization(delegatingIdentity,
delegatedUserAzManSID, RestrictedAuthorizationType.Allow, validFrom, validTo);
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces

Imports System.Security.Principal
```

```
...

Dim item As IAzManItem = ...

Dim delegatingIdentity As WindowsIdentity = WindowsIdentity.GetCurrent()

Dim delegatedUserNTAccount As NTAccount = New NTAccount("MYDOMAIN",
"delegatedusername")

Dim delegatedUserSID As SecurityIdentifier =
DirectCast(delegatedUserNTAccount.Translate(GetType(SecurityIdentifier)),
SecurityIdentifier)

Dim delegatedUserAzManSID As IAzManSid = New SqlAzManSID(delegatedUserSID)

Dim validFrom As Nullable(Of Date) = DateTime.Now

Dim validTo As Nullable(Of Date) = New Nullable(Of Date) 'No expiration date

Dim del As IAzManAuthorization =
item.CreateDelegateAuthorization(delegatingIdentity, delegatedUserAzManSID,
RestrictedAuthorizationType.Allow, validFrom, validTo)
```

*The* NetSqlAzMan.SqlAzManItem.CreateDelegateAuthorization(…) *method returns an instance of* IAzManAuthorization *type that you can use to add one or more attributes as shown in* ***print 8***.

***Print 8****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

using System.Security.Principal;

...

IAzManAuthorizationAttribute attr = del.CreateAuthorizationAttribute("My Key",
"My Value");
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces

Imports System.Security.Principal

...

del.CreateAuthorizationAttribute("My Key", "My Value")
```

*Once the delegation has been carried out, the "delegated" user can access the application functionalities represented by* Item *for which he's been delegated. (the permission assigned by the delegation is* Allow*).*

*When you work with delegations is very important to know if the user has already delegated others or not, simply to avoid that the same user can do again the delegate operation on the same delegated one. To do so is necessary to read the given authorizations to a certain* Item *using such methods as* NetSqlAzMan.SqlAzManItem.GetAuthorizationsOfOwner(…) *supplying the "delegant" user* SID *or, the* NetSqlAzMan.SqlAzManItem.GetAuthorizations(…) *method for delegations on specific user, passing the "delegant"* SID *and the "delegated"* SID. *An example of use about these two methods in* ***print 9****.*

***Print 9****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

using System.Security.Principal;

...

IAzManItem item = ...;

IAzManSid delegatingUserSid = ...;

IAzManSid delegatedUserSid = ...;


IAzManAuthorization[] authorizations =
item.GetAuthorizationsOfOwner(delegatingUserSid);

foreach (IAzManAuthorization auth in authorizations)

{

    if (auth.SID.StringValue.Equals(delegatedUserSid.StringValue))

    {

        //delegatedUserSid is already a delegate

    }

}
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces
```

```
Imports System.Security.Principal

...

Dim item As IAzManItem = ...

Dim delegatingUserSid As IAzManSid = ...

Dim delegatedUserSid As IAzManSid = ...


Dim authorizations As IAzManAuthorization() =
item.GetAuthorizationsOfOwner(delegatingUserSid)

For Each auth As IAzManAuthorization In authorizations

    If auth.SID.StringValue.Equals(delegatedUserSid.StringValue) Then

        'delegatedUserSid is already a delegate

    End If

Next
```

*At last, to remove a delegation, executed from a "u1" to "u2" user it's available the* NetSqlAzMan.SqlAzManAuthorization.DeleteDelegateAuthorization(…) *method; the supplying parameters are the "u1" user* WindowsIdentity *and the "u2" user* SID *(**print 10**).*

***Print 10**:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

using System.Security.Principal;

...

IAzManItem item = ...;

WindowsIdentity u1 = WindowsIdentity.GetCurrent();

IAzManSid u2 = ...;

item.DeleteDelegateAuthorization(u1, u2);
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces
```

```
Imports System.Security.Principal

...

Dim item As IAzManItem = ...

Dim u1 As WindowsIdentity = WindowsIdentity.GetCurrent()

Dim u2 As IAzManSid = ...

item.DeleteDelegateAuthorization(u1, u2)
```

## Manipulating the NetSqlAzMan Storage by .NET code

*The* NetSqlAzMan API *are more useful to manipulate the Sql Server* Storage *directly from .NET code. It's enough to think that the administrative console, internally, uses exactly these API; that means that all we can do from the console, we can do it from source code too.*

*It would be long and extremely laborious describing every single method of each class present in the* NetSqlAzMan *DOM , for further details refer to the* chm *supplied with the product.*

*In the example shown in* ***print 11****, we use the DOM to create at run-time a* Store*, an* Application*, a* Role*, a* Task*, an* Operation *and to make this last one as* Item *member of the* Task *and the* Task *as* Item *member of the* Role*; in this example an* Authorization *and an* AuthorizationAttribute *have been created, too. Remember that the user that will execute such code must belong to the Sql* Database Role*:* NetSqlAzMan_Administrators*.*

***Print 11****:*

C#

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

using System.Security.Principal;

...

string cs = "Data Source=(local);Initial Catalog = NetSqlAzManStorage;Integrated
Secuirty = SSPI;";

IAzManStorage storage = new SqlAzManStorage(cs);

WindowsIdentity identity = WindowsIdentity.GetCurrent();

IAzManSid targetUserSID = ...;


try

{
```

```
    storage.OpenConnection();

    storage.BeginTransaction();

    IAzManStore newStore = storage.CreateStore("A New Store", "Store
description");

    IAzManApplication newApplication = newStore.CreateApplication("A New
Application", "Application description");

    IAzManItem role1 = newApplication.CreateItem("Role 1", "Role description",
ItemType.Role);

    IAzManItem task1 = newApplication.CreateItem("Task 1", "Task description",
ItemType.Task);

    IAzManItem op1 = newApplication.CreateItem("Operation 1", "Operation
description", ItemType.Operation);

    task1.AddMember(op1);

    role1.AddMember(task1);

    IAzManAuthorization auth = op1.CreateAuthorization(new
SqlAzManSID(identity.User), WhereDefined.LDAP, targetUserSID, WhereDefined.LDAP,
AuthorizationType.Deny, new DateTime(2006, 1, 1), new DateTime(2006, 12, 31));

    IAzManAuthorizationAttribute attr = auth.CreateAuthorizationAttribute("Some
Key", "Some Value");

    storage.CommitTransaction();

}

catch

{

    if (storage.TransactionInProgress)

        storage.RollBackTransaction();

}

finally

{

    storage.CloseConnection();

}
```

VB.NET

```
Imports NetSqlAzMan

Imports NetSqlAzMan.Interfaces
```

```
Imports System.Security.Principal

...

Dim cs As String = "Data Source=(local);Initial Catalog =
NetSqlAzManStorage;Integrated Secuirty = SSPI;"

Dim storage As IAzManStorage = New SqlAzManStorage(cs)

Dim identity As WindowsIdentity = WindowsIdentity.GetCurrent()

Dim targetUserSID As IAzManSid = ...


Try

    storage.OpenConnection()

    storage.BeginTransaction()

    Dim newStore As IAzManStore = storage.CreateStore("A New Store", "Store
description")

    Dim newApplication As IAzManApplication = newStore.CreateApplication("A New
Application", "Application description")

    Dim role1 As IAzManItem = newApplication.CreateItem("Role 1", "Role
description", ItemType.Role)

    Dim task1 As IAzManItem = newApplication.CreateItem("Task 1", "Task
description", ItemType.Task)

    Dim op1 As IAzManItem = newApplication.CreateItem("Operation 1", "Operation
description", ItemType.Operation)

    task1.AddMember(op1)

    role1.AddMember(task1)

    Dim auth As IAzManAuthorization = op1.CreateAuthorization(New
SqlAzManSID(identity.User), WhereDefined.LDAP, targetUserSID, WhereDefined.LDAP,
AuthorizationType.Deny, New DateTime(2006, 1, 1), New DateTime(2006, 12, 31))

    Dim attr As IAzManAuthorizationAttribute =
auth.CreateAuthorizationAttribute("Some Key", "Some Value")

    storage.CommitTransaction()
Catch

    If storage.TransactionInProgress Then

        storage.RollBackTransaction()

    End If

Finally

    storage.CloseConnection()
```

```
End Try
```

*The* API *use* LINQ –Language Integrated Query *to read and write from* Sql Server *and this involves a performance increase and all the security provided by* LINQ *(for ex. no sql injection)*

## *ENS (Event Notification System)*

*All the* NetSqlAzMan *classes fire events and this fact is very useful if, for example, we want to trace a log about all the operations carried out by users on the* Storage*. Moreover every events of every class you could possibly imagine have been collected and centralized in a unique class with static events:* SqlAzManENS*.*

*ENS is a powerful System of Events Notification to use both with administrative scope and in order to manage all the events that happen in the* Storage *through your application. In* ***print 12*** *an example how using the* NetSqlAzManENS *class.*

***Print 12****:*

C#

```
using NetSqlAzMan;
using NetSqlAzMan.Interfaces;
using NetSqlAzMan.ENS;
...
SqlAzManENS.StoreCreated += new StoreCreatedDelegate(SqlAzManENS_StoreCreated);
...
void SqlAzManENS_StoreCreated(IAzManStore storeCreated)
{
    System.Diagnostics.Debug.WriteLine(storeCreated.Name + " store created");
}
```

VB.NET

```
Imports NetSqlAzMan
Imports NetSqlAzMan.Interfaces
Imports NetSqlAzMan.ENS
...
```

```
Dim WithEvents ens As SqlAzManENS 'class-level field

...

Private Sub ens_StoreCreated(ByVal storeCreated As
NetSqlAzMan.Interfaces.IAzManStore) Handles ens.StoreCreated

    System.Diagnostics.Debug.WriteLine(storeCreated.Name + " store created")

End Sub
```

## Building Applications with NetSqlAzMan

Requirements:

- ✓ .NET Framework 3.5
- ✓ NetSqlAzMan 3.5.0.0 (or upper) installed
- ✓ Visual Studio .NET 2008

For each tutorial:

- ➢ Create a new Windows/Web application
- ➢ Add a reference to the NetSqlAzMan.dll assembly

## Tutorial 1: CheckAccess inside Windows/Web Applications

- ➢ *Add using clauses:*

```
using NetSqlAzMan;

using NetSqlAzMan.Interfaces;

...
```

- ➢ *Create a SqlAzManStorage instance:*

```
string cs = "Data Source=(local);Initial Catalog = NetSqlAzManStorage;Integrated
Secuirty = SSPI;";

IAzManStorage storage = new SqlAzManStorage(cs);
```

- ➢ *Get user Windows Identity:*

```
System.Security.Principal.WindowsIdentity identity =
System.Security.Principal.WindowsIdentity.GetCurrent();
```

- ➢ *Invoke SqlAzManStorage.CheckAccessMethod:*

```
//Can I do "My Operation" (or my Task … or my Role) ?

AuthorizationType authorization = storage.CheckAccess("My Store", "My
Application", "My Operation", identity, DateTime.Now, false);
```

- ➢ *Use authorization result for your business logic:*

```
switch (authorization)

{

    case AuthorizationType.AllowWithDelegation:

      //Yes, I can ... and I can delegate
```

```
        break;
    case AuthorizationType.Allow:
        //Yes, I can
        break;
    case AuthorizationType.Deny:
    case AuthorizationType.Neutral:
        //No, I cannot
        break;
}
```

## Tutorial 2: CheckAccessHelper

*NetSqlAzMan is able to generate for you the CheckAccess client code into an helper class called CheckAccessHelper.*

*To generate CheckAccessHelper class:*

- *Open NetSqlAzMan MMC console or Web console*
- *Choose your application*
- *Right click – Generate Check Access Helper*
- *If you want to allow check access on Roles and Tasks to check relative options*
- *Copy and Paste generated code into a file class of your project*

*To use CheckAccessHelper:*

- *Create an instance of CheckAccessHelper class:*

```
string cs = "data source=eidosis4-afr;Initial
Catalog=NetSqlAzManStorage;Integrated Security = SSPI;";
          My_Application.Security.CheckAccessHelper helper = new
My_Application.Security.CheckAccessHelper(cs, WindowsIdentity.GetCurrent());
```

- *Open the connection:*

```
helper.OpenConnection();
```

- *Invoke CheckAccess on your items:*

```
bool result =
helper.CheckAccess(My_Application.Security.CheckAccessHelper.Operation.Op1);
```

- *Close the connection:*

```
helper.CloseConnection();
```

- *Use result for your business operations (i.e. to enable/disable some UI elements)*

```
//Use result for your business logic
    if (result == true)
    {
        //Allow or AllowWithDelegation
    }
    else
    {
        //Deny or Neutral
    }
```

## Tutorial 3: UserPermissionCache

*UserPermissionCache class is able to cache all authorizations of a given Application for a given user (Windows or DB).*

- To use it create an instance of a SqlAzManStorage class:

```
IAzManStorage storage = new SqlAzManStorage("data source=.;Initial
Catalog=NetSqlAzManStorage;Integrated Security = SSPI;");
```

- Create an instance of a UserPermissionCache class:

```
NetSqlAzMan.Cache.UserPermissionCache userPermissionCache = new
NetSqlAzMan.Cache.UserPermissionCache(storage, "My Store", "My Application",
WindowsIdentity.GetCurrent(), true, true);
```

- (If your application is a Web Application you can put UserPermissionCache instance into a Session variable or in the ASP.NET Cache to cache authorizations for all the session time. This step avoid UserPermissionCache rebuilding.)

- Invoke CheckAccess method:

```
AuthorizationType auth = userPermissionCache.CheckAccess("My Operation",
DateTime.Now);
```

- Use auth result for your business logic.

Andrea Ferendeles
aferende@hotmail.com

## Tutorial 4: StorageCache

*NetSqlAzMan.Cache.StorageCache class is able to cache an entire Storage (Stores, Applications, Items, Authorizations, Attributes …) by retrieving all Storage data in a single set of T-SQL SELECT operations. After this, the connection is closed.*

*StorageCache expose four CheckAccess methods that calculate permissions over cached elements and without opening connection to the SQL Server Storage.*

*StorageCache can be used to:*

- *Increase CheckAccess performance (no SQL data retrieve)*
- *Client-side CheckAccess*

*To enable StorageCache cache:*

- *Create an instance of the NetSqlAzMan.Cache.StorageCache class:*

```
NetSqlAzMan.Cache.StorageCache sc = new StorageCache("data source=.;initial
catalog=NetSqlAzManStorage;user id=sa;password=");
```

- *Invoke BuildStorageCache() method to cache all Storage elements:*

```
sc.BuildStorageCache("My Store", "My Application");
```

- *(If you want to cache all elements of a single Store or a single Application invoke BuildStorageCache(storeNameFilter) / BuildStorageCache(storeNameFilter, applicationNameFilter).)*
- *Invoke CheckAccess() method:*

```
WindowsIdentity wid = WindowsIdentity.GetCurrent();
string user = wid.GetUserBinarySSid(); //using NetSqlAzMan.Cache needed
string[] groups = wid.GetGroupsBinarySSid(); //using NetSqlAzMan.Cache needed
AuthorizationType au = sc.CheckAccess("My Store", "My Application", "My
Operation", user, groups, DateTime.Now, false);
```

- *Use result for your business logic:*

## Tutorial 5: WCF Cache Service

*NetSqlAzMan Cache Service is a WCF (Windows Communication Foundation) service, hosted by a Windows NT Service over Http/Net.Tcp protocol.*

*To install NetSqlAzMan Cache Service:*

- *Download NetSqlAzMan Cache Service from http://netsqlazman.codeplex.com*
- *Install the service*
  *P.S.1: If use Integrated Security = true in the Sql connection string, service user must be granted must be in the NetSqlAzMan_Readers sql role.*
  *P.S.2: If you have LDAP Store/Application groups in your Storage, service user must be a DOMAIN user able to read from your Active Directory Domain.*
- *Open NetSqlAzManCacheService.exe.config file and change configuration options:*

```
<connectionStrings>
       <add
name="NetSqlAzMan.Cache.Service.Properties.Settings.NetSqlAzManStorageCacheConnectionString"
          connectionString="Data Source=(local);Initial Catalog=NetSqlAzManStorage;Integrated
Security=True"
          providerName="System.Data.SqlClient" />
    </connectionStrings>
    <appSettings>
      <add key="expirationValue" value="0 1 0 0" /> <!-- days hours minutes seconds -->
      <add key="StoreNameFilter" value="" />       <!-- leave empty for all Stores -->
      <add key="ApplicationNameFilter" value=""/>   <!-- leave empty for all Applications -->
    </appSettings>
…
<!-- NET TCP SERVICE -->
          <service behaviorConfiguration="NetSqlAzMan.Cache.Service.NETTCPCacheServiceBehavior"
               name="NetSqlAzMan.Cache.Service.CacheService">
            <endpoint address="" binding="netTcpBinding"
contract="NetSqlAzMan.Cache.Service.ICacheService" />
                <endpoint address="mex" binding="mexTcpBinding" contract="IMetadataExchange" />
            <host>
              <baseAddresses>
                <add baseAddress="net.tcp://localhost:8000/NetSqlAzMan.Cache.Service/CacheService/"
/>
                <add baseAddress="http://localhost:9000/NetSqlAzMan.Cache.Service/CacheService/" />
              </baseAddresses>
            </host>
          </service>
```

- *Start the service*
- *Check Application Log Events for cache build results.*

*To use NetSqlAzMan Cache Service:*

- *Create a new Web/Windows client application*
- *Add a Service Reference to the mex Address (default is:* `http://localhost:9000/NetSqlAzMan.Cache.Service/CacheService/` *) and call Service reference "sr".*
- *Create an instance of the WCF service proxy class:*

```
sr.CacheServiceClient csc = new sr.CacheServiceClient();
```

- *Open service connection*

```
csc.Open();
```

- *Invoke CheckAccess methods:*

```
- csc.CheckAccessForWindowsUsersWithAttributesRetrieve(...);
- csc.CheckAccessForWindowsUsersWithoutAttributesRetrieve(...);
- csc.CheckAccessForDatabaseUsersWithAttributesRetrieve(...);
- csc.CheckAccessForDatabaseUsersWithoutAttributesRetrieve(...);
```

- *Invoke InvalidateCache if you want to force cache re-building (i.e. if authorizations are changed on SQL Storage):*

```
csc.InvalidateCache();
```

- *Close service connection (VERY IMPORTANT !)*

```
csc.Close();
```

## Conclusions

*I think we all agree if we say that a lot of applications need or have needed, in the past, of a centralized system to manage authorizations. Who of you has needed, almost once, to have such kind of management for an application? How is it gone? All by hand? And every time do it again? All custom in the DB ? Draw the conclusions.*

*There are a lot of other features in NetSqlAzMan, among them:*

- *Asynchronous* CheckAccess *(*BeginCheckAccess */* EndCheckAccess*)*
- Store Attributes, Application Attributes, Item Attributes.
- NetSqlAzMan *custom* Exceptions.
- *All the operations on* Storage *can be transactional*
- *(*Storage.BeginTransaction/CommitTransaction/RollBackTransaction*)*
- *Import/Export in XML format directly from the console.*
- *Import from MS Authorization Manager.*
- LDAP *query test for dynamical groups from the console (*Store Group *and* Application Group*).*
- *All the operations executed in the console are logged in the machine Application Log.*

*The only fact that is.NET 3.5 native and supports Sql Server for its authorizations* Storage, *let's consider this product surely interesting and worth to test it.*

*At last, I remember you that the source code (C#.NET) is available, both for the console and the run-time engine, as well the setup package (see riff.).*

## Thanks to

*A special thank to "Catho", who gave me precious suggestions during* NetSqlAzMan *design time, besides doing a meticulous testing job. Thanks "Catho".*

*Another special thank to* Giacinta *(my wife) and to* Nagireddy Tamalapudi *for English translation of this document.*

*Thanks guys !!!*

**Andrea Ferendeles.**

## References

**[1]** *SQLAudit home site: http://sqlaudit.sourceforge.net (documentation and quickstart).*

**[2]** *NetSqlAzMan home site: http://netsqlazman.codeplex.com (sources, installer, documentation).*