



Product: **Database Accelerator (DBXL)**

Title: **How to use the QueryDB Permissions
plugin interface**

Starting with version 2.2, DBXL includes a mechanism for defining custom permissions to the QueryDB and DatabaseEnumeration web services. You can utilize this interface to restrict certain sensitive databases and tables that should not be exposed by the web service at all, or you can permit access to certain databases and tables for only certain users or roles. The logic is yours to define.

CONTENTS

How it works	2
An Example	3
Creating a new assembly project.....	3
Adding Assembly references	4
Adding a reference to the DatabaseService assembly	4
Adding a reference to System.Web	5
Writing the permissions plugin.....	6
Configuring DBXL to recognize the plugin	8
A More Thorough Approach	9
QueryDBPermissions.xml	10



HOW IT WORKS

QueryDB permissions are defined by using any .NET compatible language to implement the following method from the `IQueryDBPermissionProvider` interface:

```
bool HasPermission(string database, string table);
```

This method accepts two parameters: a database name (or alias) and a table (or view) name. Your implementation of this method should inspect these two values and determine whether to grant or deny access to this database/table combination for the current user. The method should return a *true* Boolean value if access should be granted, or *false* if not. Your permission logic can be as simple or complex as you need it to be, and can involve practically any .NET constructs you would like to use.

Once you have designed the plugin the way you want, you should compile it into a DLL, and copy this DLL into DBXL's **bin** folder, and modify DBXL's web.config to instruct DBXL to use the plugin. See page 8 of this document for more details on modifying web.config.

Note that if you use more than one QueryDB permission plugin on your system that DBXL will grant access to a table as long as at least one plugin grants access to that table, and will deny access only if all plugins deny access.



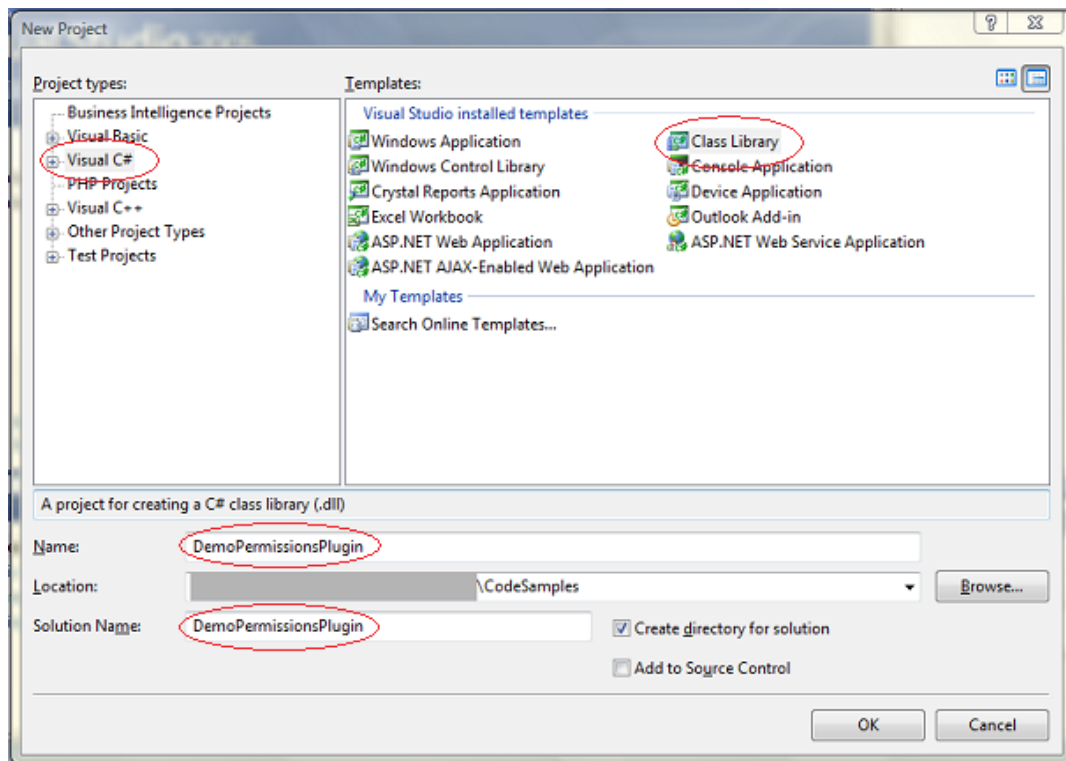
AN EXAMPLE

Here is a simple walkthrough to illustrate how to go about creating a custom permissions plugin for QueryDB. This example uses steps and screenshots from Visual Studio 2005.

CREATING A NEW ASSEMBLY PROJECT

Open a new instance of Visual Studio, and open the **New Project** dialog by selecting **Project...** from the **File** menu.

Under **Project Types**, select **Visual C#**, and then select **Class Library** under **Templates**. Give your project and solution the name *DemoPermissionsPlugin*



A new project should open with a C# class file *Class1.cs* already added to it.



ADDING ASSEMBLY REFERENCES

To get your code to acknowledge the `DatabaseService` assembly, and to use the web-server-related code used in this example. You will need to add references to a few assemblies in your project. This section describes how to do this.

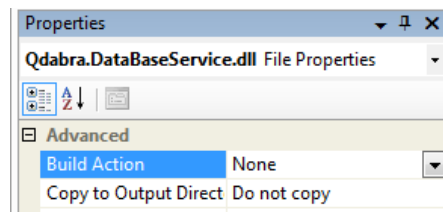
ADDING A REFERENCE TO THE DATABASESERVICE ASSEMBLY

In order for your project to be aware of the `IQueryDBPermissionsProvider` interface, you will need to create a reference to the `DatabaseService` assembly. To ease this process, you can include a copy of the `DatabaseService` DLL in your project.

In the Project Explorer window of Visual Studio, right-click the project name, select **Add**, and select **Existing Item...**. Navigate to a location where you have stored a copy of the `Qdabra.QueryDB.dll` file that is installed with DBXL in the site's **bin** directory, and click OK to import this file into your project.

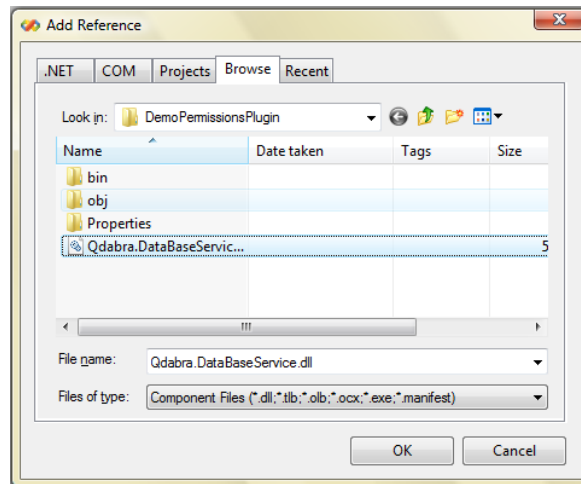
Note: In older versions of DBXL, this file is called `Qdabra.DatabaseService.dll`, and this is the name shown in the screenshots below.

Once the file is imported, click it in the Project Explorer, and in the Properties window, change the file's **Build Action** to *None*. This file is present purely as a reference and should not be marked as any sort of output for your project.



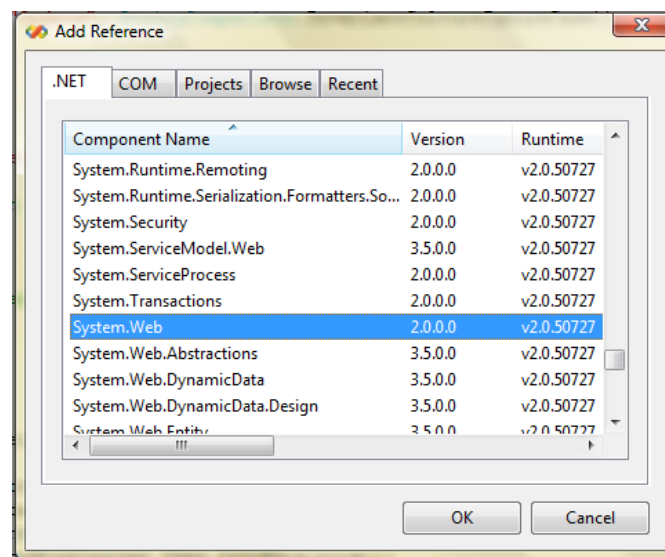
Finally, to actually add the reference to the library, right-click the project name again in the Project Explorer, select **Add Reference...**, click the **Browse** tab, and locate the newly included DLL file *within the directory structure of your project*.





ADDING A REFERENCE TO SYSTEM.WEB

Since the code used in this example uses classes in the System.Web .NET assembly, you will need to add a reference to this as well. Right-click the project name in the Project Explorer again, and again select **Add Reference....** This time, click the .NET tab, and select System.Web from the list.



WRITING THE PERMISSIONS PLUGIN

Now that we have set up the necessary assembly references, we're ready to start coding.

Open up the *Class1.cs* file. After all of the **using** statements at the top, add **using** statements to indicate the *System.Web* and *DatabaseService* namespaces.

```
using System.Web;
using Qdabra.DataBaseService;
```

Rename the class from *Class1* to a more descriptive name: *QueryDBPermissions*, and indicate that it implements the *IQueryDBPermissionProvider* interface.

```
public class QueryDBPermissions : IQueryDBPermissionProvider
```

Now, in between the opening and closing brackets of the class, paste the following code:

```
#region IQueryDBPermissionProvider Members

public bool HasPermission(string database, string table)
{
    // Allow all users to access only the ExpenseReports and
    // Orders tables in #TestDatabase#
    if (database.Equals("#TestDatabase#", StringComparison.InvariantCultureIgnoreCase))
    {
        if (table.Equals("ExpenseReports", StringComparison.InvariantCultureIgnoreCase) ||
            table.Equals("Orders", StringComparison.InvariantCultureIgnoreCase))
        {
            return true;
        }

        if (IsUserInRole("SystemAdmin"))
        {
            // Allow users in the SystemAdmin role to access anything in #TestDatabase#
            return true;
        }
    }

    // Deny access for any request that doesn't pass the above checks
    return false;
}

#endregion

private bool IsUserInRole(string role)
{
    HttpContext currentHttpContext = System.Web.HttpContext.Current;
    return (currentHttpContext != null &&
        currentHttpContext.User.Identity.IsAuthenticated &&
        currentHttpContext.User.IsInRole(role));
}
```

The above code grants access to all users to query the tables named *Expense Reports* and *Orders* in the database with the alias *TestDatabase*, and nothing else. Additionally, it allows any user in the *SystemAdmin* role to query any table or view in the database with alias *TestDatabase*, and nothing else.



Note that in actual practice, it is not advisable to hardcode database and table names into your code this way, and that it would better to devise a way to define your permissions that does not require rebuilding code.

At this point you should be able to build your C# project and locate the *DemoPermissionsPlugin.dll* file under your project's *bin* folder. Please copy this file to the *bin* folder of your QdabraWebService directory on your web server.



CONFIGURING DBXL TO RECOGNIZE THE PLUGIN

The last step to installing the plugin is to modify DBXL's web.config file to indicate that it should use this plugin. Please open DBXL's web.config file in a text editor and locate the **<enumdb>** section.

Now you will need to add an entry to the **<providers>** element inside the **<enumdb>** section. Please add the **<providers>** element inside the **<enumdb>** section if it is not present:

```

    <!--<database
  </databases>
  <providers>
  </providers>
</enumdb>

```

Now you will need to add a **<provider>** element inside the **<providers>** element. This element should have two attributes: **name**, which can be any arbitrary name for the permission plugin, and **type**, which should indicate the class and assembly name of the plugin provider. Since the assembly in this case is *DemoPermissionsPlugin*, and the class is *QueryDBPermissions* in the *DemoPermissionsPlugin* namespace, you should have a result like this:

```

<providers>
  <provider name="DemoPermissions"
    type="DemoPermissionsPlugin.QueryDBPermissions, DemoPermissionsPlugin" />
</providers>

```

If you create your own plugin with a different assembly name, namespace or class, a guide for the format to use in the type= attribute is:

NamespaceName.ClassName, AssemblyName

If you carried out all of these steps successfully, your permissions plugin should be in place and ready to go. In order to test it, you will need to add the *TestDatabase* alias to the **<databases>** element in DBXL's web.config, and create tables with the names *ExpenseReports* and *Orders*, (and another table that should be off limits). Now all you have left to do is make a simple InfoPath form to query the web service and make sure that the plugin is working.



A MORE THOROUGH APPROACH

This tutorial includes a sample Visual Studio 2005 project that implements a much more general approach to QueryDB permissions than the preceding example.

This example utilizes an external XML file, which can be easily customized to define the permissions you want in a declarative manner.

Since it is fairly more complex than the preceding example, the details of the code will not be reproduced here, but the code itself is liberally documented with comments.

In order to use this sample with your current DBXL instance, you should replace the QueryDB dll in the project, update the assembly references as needed, and rebuild the plugin.

1. Copy the **Qdabra.QueryDBPluginSample.dll** file that you have built into the **bin** folder of your installation of DBXL.
2. Copy the **QueryDBPermissions.xml** file into a subfolder called **xml** in your DBXL main folder. (You will most likely need to create this folder yourself.)
3. Modify the DBXL **web.config** file to indicate the plugin's assembly and class as described in the *Configuring DBXL to recognize the plugin* section on page 8 of this document. In this case, the value to put in the type= attribute would be **QueryDBPluginSample.QueryDBPlugin, Qdabra.QueryDBPluginSample**.
4. Modify the **QueryDBPermissions.xml** file from the **xml** folder in step 2 to indicate the permissions you want to enforce (see the next section for details).



QUERYDBPERMISSIONS.XML

The example permission plugin included with this tutorial looks for a file called **QueryDBPermissions.xml** in the **xml** subfolder of DBXL's folder on the server. This XML file should have this general structure:

```
<databasePermissions>
  <database name="#Database1#">
    <table name="Table1">
      <allow users="THarris, SKing" />
      <deny users="*" />
    </table>
    <table name="Table2">
      <deny roles="Restricted Users" />
      <allow users="*" />
    </table>
  </database>
  <database name="#Database2#">
    <table name="Table1">
      <allow users="*" />
    </table>
  </database>
</databasePermissions>
```

A description of the relevant nodes in this structure follows:

<database>

The file should contain one of these for each database to which you want to grant access. The **name** attribute should indicate the name, or alias surrounded in # symbols, of the database, to match the name that will be used in QueryDB queries. The **name** attribute is not case sensitive.

Note:

If the file contains more than one <database> node for the same database, only the first in the file will be used.

This permission plugin is designed to deny access to any database that is not indicated in the **QueryDBPermissions.xml** file.

<table>

The file should contain one of these for each table or view to which you want to grant access, inside the corresponding <database> node. The name attribute should indicate the name of the table or view.

The <table> node should contain a series of <deny> and <allow> nodes to indicate who can and cannot access the table.

Note:



If the file contains more than one <table> node for any given table, only the first one will actually be used.

This permission plugin is designed to deny access to any table or view that is not indicated in the QueryDBPermissions.xml file.

<allow> and <deny>

The <allow> and <deny> nodes inside a <table> node are evaluated one at a time, from top to bottom, to determine whether to grant or deny the current user permission to the indicated table.

As soon as an <allow> or <deny> node that matches the current user is found, the plugin will grant permission to the table if it is an <allow> node, or deny permission if it is a <deny> node. If no <allow> or <deny> node that matches the current user is found, the plugin will deny access to the table.

Both of these nodes can have either or both of the following attributes:

users – Indicates a comma-separated list of usernames to which to grant or deny access. The following node grants access to the users with the usernames THarris and SKing:

```
<allow users="THarris, SKing" />
```

roles – Indicates a comma-separated list of roles to which to grant or deny access. The following node denies access to all users in the "Restricted Users" role:

```
<deny roles="Restricted Users" />
```

Additionally, the * symbol can be used in either attribute to indicate that an <allow> or <deny> rule should be applied to all users:

```
<allow users="*" />
```

