Custom Users and Groups Import/Export

By Josh Fletcher, Technical Support Engineer, 4D Inc.

Technical Note 07-12

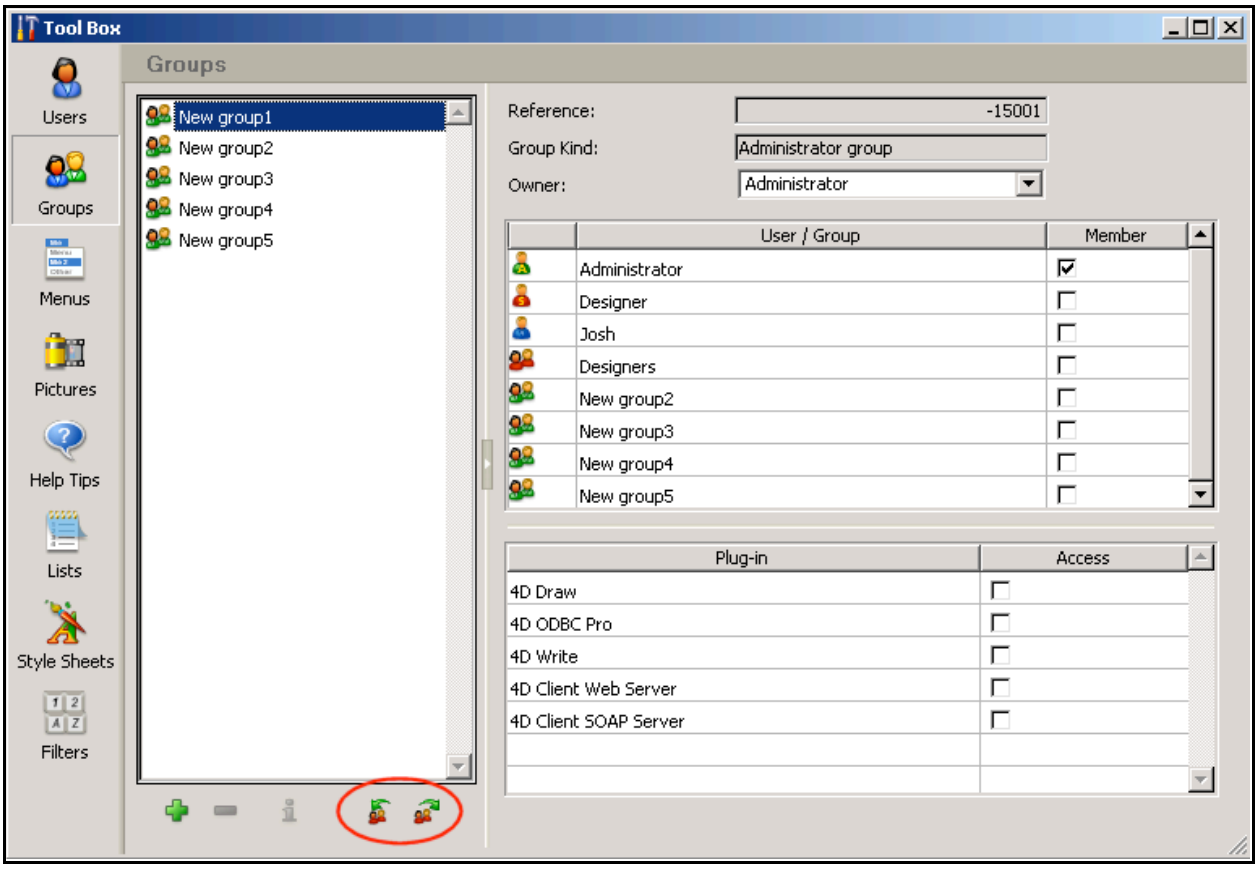
Abstract

This Technical Note presents an example implementation of a custom Users and Groups import/export module. Of particular interest is the ability to import and export plug-in access settings, something that the built-in Users and Groups import/export feature of 4th Dimension (4D) 2004 does not do.

The source code is provided, as well as a demo database.

Introduction

4D 2004 features the ability to import and export both users and groups created by the Administrator account. This feature is exposed in the Tool Box on the "Groups" page, circled below:



The **EDIT ACCESS** command can also be used to expose this dialog outside of the Design environment.

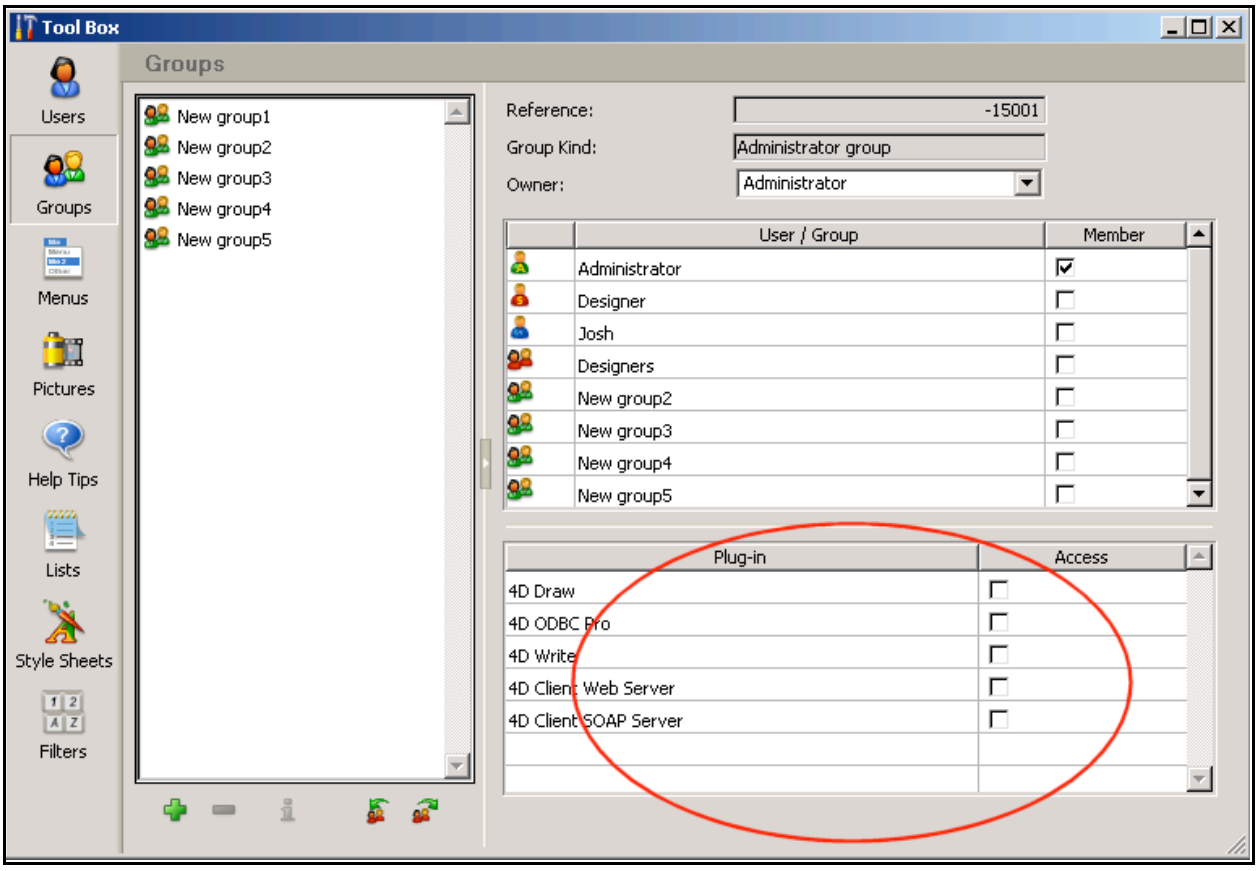
From the 4D 2004 Design Reference:

> *The ability to save groups means that the Administrator can save the access system of a database and transfer it to a modified version of the same database or to a new database. This is extremely useful for restoring the access system for a new version of the database. Because the groups can be reloaded, users of the database do not have to learn a new access system.*
>
> *All the user names, passwords, startup method names, groups, group owners, and group memberships are preserved.*

This feature does not, however, export nor import any plug-in access settings. When importing the users and groups into a different database the plug-in access settings must be manually changed. This can obviously be a tedious process in a database that might have many groups and/or many plug-ins.

4D Plug-In Access Settings

4D provides the ability to assign plug-in access to a specific group. These settings are typically configured in the Tool Box, as below:



Configuring the plug-in access for a database allows the developer and/or database administrator to limit the number of users who can access a particular plug-in. This might be done for security reasons, or to prevent licenses from inadvertently being used (important for the client/server environment).

The ability to export and/or import these settings is just one example of how the user and groups import/export process can be customized.

Customization – Step One – Users and Groups

Access to Design mode might be restricted or completely unavailable (as in a compiled database) so the Tool Box may not always be a viable option for importing or exporting the users and groups. Of course the **EDIT ACCESS** command can be used to expose the Tool Box in custom menus mode but even this does not allow much customization.

The first step to customizing the process is to get the users and groups programmatically. This is accomplished with the use of the 4D commands **USERS TO BLOB** and **BLOB TO USERS**.

USERS TO BLOB

Here is the entry for **USERS TO BLOB** from the 4D Language Reference (edited for brevity):

> *The USERS TO BLOB command stores in the BLOB…the list of all user accounts and database groups created by the Administrator.*
>
> *The generated BLOB is automatically encrypted and can only be read using the BLOB TO USERS command. You can store this BLOB in a file on your hard disk or in a field.*

With this command the users and groups can easily be exported to any format and/or location desired using the many commands in 4D that can manipulate BLOBs.

BLOB TO USERS

Here is the entry for **BLOB TO USERS** from the 4D Language Reference (again, edited for brevity):

> *The BLOB TO USERS command adds the user accounts present in the BLOB users in the database. The BLOB…is encrypted and must have been created using the USERS TO BLOB command.*

The reflection of **USERS TO BLOB**, **BLOB TO USERS** allows the users and groups to be imported from any source, provided the BLOB can be restored to the appropriate form.

Step One Complete

With the ability to programmatically access the users and groups in place the import/export process can be customized at will. Additionally the information can be stored in any form desired.

The example code included with this Technical Note stores the information in an XML file (this will be discussed in-depth later).

Also note that many of the commands in the "Users and Groups" theme of the 4D language can be used to customize the import/export process (refer to the 4D Language Reference for a complete list of commands).

In this Technical Note the commands related to plug-in access are explored.

Customization – Step Two – Plug-In Access

In order to import and/or export plug-in access settings two commands from the 4D language need to be used: **SET PLUGIN ACCESS** and **Get plugin access**.

SET PLUGIN ACCESS

The **SET PLUGIN ACCESS** command provides programmatic access to set the group that is allowed to use each "serialized" plug-in that is installed in the database. This can be used to manage how plug-in licenses are used.

The list of serialized plug-ins can be found in the 4D Language Reference.

Note that only one group at a time can be assigned to any given plug-in. When this command is successfully executed, if another group had the plug-in access rights, it loses this privilege.

Also note that this command will only change the plug-in access setting if the plug-in is installed, licensed, and not disabled. If any of these conditions are not met, the plug-in access setting will not be set (and the OK variable will be set to 0).

Get plugin access

The **Get plugin access** command returns the name of the group that is authorized to use the serialized plug-in whose number was passed. If there is no group associated with the plug-in, the command returns an empty string ("").

The list of serialized plug-ins can be found in the 4D Language Reference.

Note that **Get plugin access** only returns the group name if the plug-in is installed, licensed, and not disabled. If any of these conditions are not met the

command will not return the group name, regardless of whether or not the setting exists.

What's next?

The four commands presented are all that are needed to start building a customized users and groups import/export module that includes plug-in access settings. Refer to the "The Source Database" section of this Technical Note for information about the implementation presented in the example database.

Customization – Step Three, and Beyond…

The example database in this Technical Note only deals with exporting users and groups along with plug-in access settings. However, there are many other commands in the **Users and Groups** theme of the 4D language that allow for further customization. For instance group membership can be changed programmatically with the command **Set user properties**.

The reader is encouraged to explore the possibilities that the commands in this theme offer. For more information refer to the 4D Language Reference.

Implementation - The Source Database

This section deals with the implementation of the custom users and groups import/export module provided in the example database.

Overview

The example database included with this Technical Note ("UG_Source.4DB") implements a Users and Groups import/export module that can be used to import and export both 4D Users and Groups as well as plug-in access settings.

The information is exported in XML format using 4D's DOM commands. In particular the 4D Users and Groups are exported as a Base64 encoded BLOB and the plug-in access settings are exported as a simple list of XML elements with the access group as the element value.

The source database contains approximately 40 project methods however it is important to note that most of this code is simply used to present a relatively robust interface. In terms of understanding the concept of customizing the users and groups import/export process there are really only 4 methods of interest:

- *UG_EXPORT_UGToXMLTree* - Saves the users and groups to an XML tree
- *UG_EXPORT_PluginAccessToXMLTree* - Saves the plug-in access settings to an XML tree
- *UG_IMPORT_XMLTreeToUG* - Loads the users and groups from an XML tree
- *UG_IMPORT_XMLTreeToPluginAccess* - Loads the plug-in access settings from an XML tree

Each of these methods is discussed below.

Finally, note that every method in the source database is documented via comments.

UG_EXPORT_UGToXMLTree

This method is straightforward. It calls a subroutine that uses the **USERS TO BLOB** command, encodes the resulting BLOB with the **ENCODE** command, and places the encoded BLOB into the passed XML tree. Here is the pertinent code from the method:

```
$encodedUsers:=UG_EXPORT_CreateEncodedUGBlob
$elemRef:=DOM Create XML element($rootRef;<>UG_XPATH_USERGROUPSBLOB)
DOM SET XML ELEMENT VALUE($elemRef;$encodedUsers)
```

The XPATH for the BLOB (assigned to <>UG_XPATH_USERGROUPSBLOB) is defined in the *UG_Startup* method.

Here is the pertinent code from the *UG_EXPORT_CreateEncodedUGBlob* subroutine (edited for brevity):

```
USERS TO BLOB($users)
ENCODE($users)
$0:=$users
```

The **ENCODE** command is used to ensure that the information placed in the XML tree will not break an XML parser (the BLOB is encoded in Base64 format). Note that the CDATA portion of the XML element could have been used to accomplish the same thing.

UG_EXPORT_PluginAccessToXMLTree

This method exports the plug-in access settings to the provided XML tree. The settings are retrieved with the **Get plugin access** command.

Important: **Get plugin access** will only work if the plug-in is installed, licensed, and not disabled.

This method uses several interprocess variables that are defined in the *UG_Startup* method. Refer to the section on that method for more information.

Here is the pertinent code from the method (edited for brevity):

```
For ($i;1;<>UG_TOTALPLUGINS)
  $groupWithAccess:=Get plugin access(<>UG_PLUGINIDS{$i})

  If (OK#1)
```

```
      $groupWithAccess:=""
    End if

    If ($groupWithAccess#"")
      $elemRef:=DOM Create XML element($rootRef;<>UG_XPATHS{$i})
      DOM SET XML ELEMENT VALUE($elemRef;$groupWithAccess)
      $anyExported:=True
    End if
  End for
```

Notice that a loop is used to process each serialized plug-in, instead of having to write a separate block of code for each plug-in. As mentioned, the arrays used in the loop are defined in the *UG_Startup* method.

This method returns a warning message if no plug-in access settings were exported. The idea behind this is that the method should not be called unless the plug-in access settings actually need to be exported, so if no settings were successfully exported something could be wrong with the plug-ins. This is not technically an error so no error message is returned. If any single plug-in access setting has been exported, no error or warning is returned.

UG_IMPORT_XMLTreeToUG

This method is straightforward. It calls a subroutine – which uses the 4D DOM commands to extract the encoded BLOB from the XML tree and decodes it with the **DECODE** command – and uses the **BLOB TO USERS** command to import the users and groups. Here is the pertinent code from the method:

```
$result:=UG_IMPORT_GetUGBlob ($rootRef;->$users)
If ($result=UGM_SUCCESS )
  BLOB TO USERS($users)
End if
```

Here is the pertinent code from the *UG_IMPORT_GetUGBlob* subroutine (edited for brevity):

```
$elemRef:=DOM Find XML element($rootRef;<>UG_XPATH_USERGROUPSBLOB)
If (OK=1)
  DOM GET XML ELEMENT VALUE($elemRef; $pUsers->)
  If (OK=1)
    DECODE($pUsers->)
  End if
End if
```

The XPATH for the BLOB element (assigned to <>UG_XPATH_USERGROUPSBLOB) is defined in *UG_Startup*.

UG_IMPORT_XMLTreeToPluginAccess

This method Retrieves the plug-in access settings from the specified XML tree and uses them to set plug-in access in the database. Here is the pertinent code:

```
For ($i;1;<>UG_TOTALPLUGINS)
  $elemRef:=DOM Find XML element($rootRef;<>UG_XPATHS{$i})
  If (OK=1)
    DOM GET XML ELEMENT VALUE($elemRef;$groupWithAccess)
    If (OK=1)
      If ($groupWithAccess#"")
        SET PLUGIN ACCESS(<>UG_PLUGINIDS{$i};$groupWithAccess)
        End if
      End if
    End if
End for
```

Notice that a loop is used to process each serialized plug-in, instead of having to write a separate block of code for each plug-in. As mentioned, the arrays used in the loop are defined in the *UG_Startup* method.

Note: this command is considered successful if at least one plug-in access setting was found and set. If any call to SET PLUGIN ACCESS fails this is considered complete failure. A warning is returned if no plug-in access settings were set.

UG_Startup

In order to create cleaner code some arrays are declared in this method to store the serialized plug-in IDs and the XPATH for each plug-in setting that can be exported in the XML file.

In other words instead of having to write a block of code like this for each plug-in:

```
$groupWithAccess:=Get plugin access( <plug-in ID constant> )

  If (OK#1)
    $groupWithAccess:=""
  End if

  If ($groupWithAccess#"")
    $elemRef:=DOM Create XML element($rootRef; <XPATH for this plug-in> )
    DOM SET XML ELEMENT VALUE($elemRef;$groupWithAccess)
    $anyExported:=True
  End if
```

By using arrays a, for loop can be used instead. This also eliminates the use of "magic numbers" in the code. Of course defining all these variables in one location makes it far easier to alter them as well.

Limitations and Enhancements

This section discusses the current limitations of the import/export module presented in the example database as well as areas for improvement.

- Because the 4D Users and Groups are exported in BLOB format no "visible" or readable representation can be made from the XML file. If viewing the XML file the users and groups will be a block of ASCII text because of the Base64 encoding.

  If it is desirable to have the actual User or Group names appear in the XML file this is possible with the other commands available in the Users and Groups theme of the 4D language. The implementation of such a feature is left to the reader.

- The import and export functionality is exposed via two forms but could easily be combined into a single, multi-page form if desired.

- The XPATH's for the XML elements are hard-coded in the database currently in a Text array that is created in the database startup code. This is done so that the settings can be imported and exported in a loop rather than line by line.

  Similarly the IDs of the serialized plug-ins are hard-coded as constants in 4D. One possible enhancement would be to define a DTD for the XML file that can account for serialized plug-ins being added (or removed) as well as defining the list of possible IDs. In this way the custom import/export module could be made more dynamic and robust.

Other Design Decisions

This information is not critical to the Technical Note topic, but worth noting so that the design in the example database is understood.

The database uses a "messaging" system instead of number-based return codes. That is, each method returns a string message rather than an error code. The messages are treated as "constants" in that they are only accessed by calling the appropriately named project method which, in turn, returns the message text. This was done for a few reasons:

- Abstracting the error codes eliminates "magic numbers" in the code. In this way the compiler can catch misspelling errors whereas if a string literal (or number) was used the compiler would not care. Also the code formatting in the method editor can point out misspelled method names (they will look like process variables rather than method calls).
- The implementation of custom constants, while possible, is not actually a documented 4D feature. The method-based system used in the database will always work whereas the implementation of constants in 4D could change without warning.

- 4D’s code completion feature can be used to complete the method name just as if a custom constant was in use (much easier than cutting and pasting a variable name).

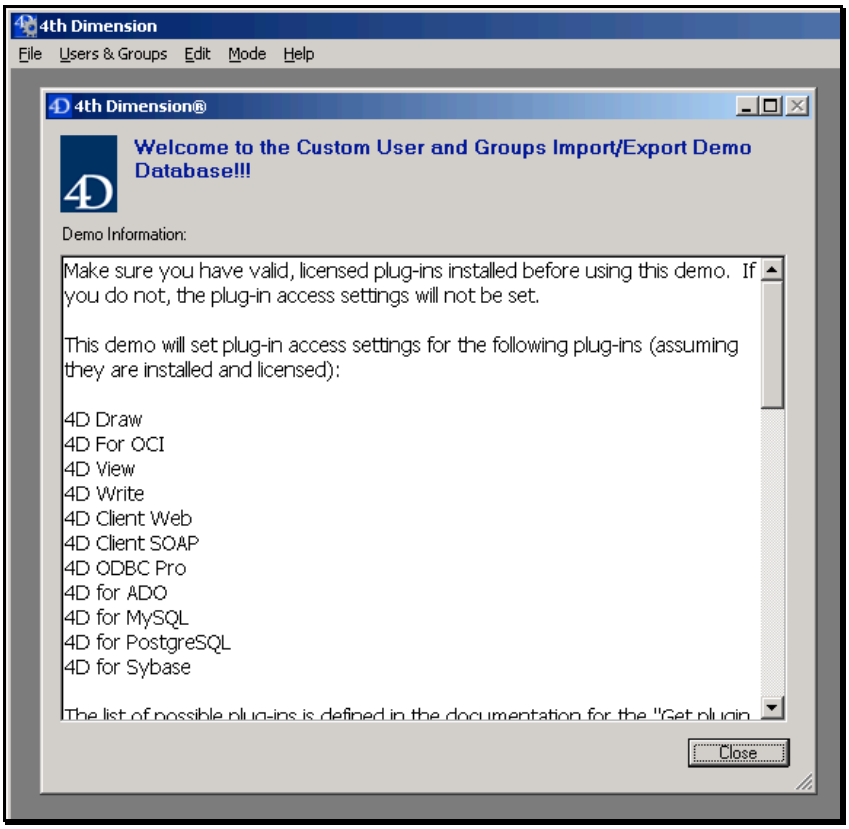
The database uses the following naming convention for method prefixes:

*UGM_*	These methods are used for error reporting.
*UGM_EX_*	Messages related to exporting.
*UGM_IM_*	Messages related to importing.
*UG_UTIL_*	Helpful methods not critical to the concepts in this Tech Note.
*UG_EXPORT_*	The export code.
*UG_IMPORT_*	The import code.
*UG_M_*	Methods that respond to a menu item.
*UG_P_*	Methods used as processes.
*UG_*	Everything else.

The Demo Database

The demo database included with this Technical Note is essentially the same as the source database but includes some extra code to present instructions to the user and also includes a sample export file. The sample export file contains several 4D Users, a Group for every serialized plug-in, as well as plug-in access settings. Note that, although the example export file contains a plug-in access setting for every serialized plug-in they may not all be imported depending in the plug-ins that are correctly configured.

To use the demo database launch "UG_Demo.4DB" in 4D:



Here are the demo instructions:

Make sure you have valid, licensed plug-ins installed before using this demo. If you do not, the plug-in access settings will not be set.

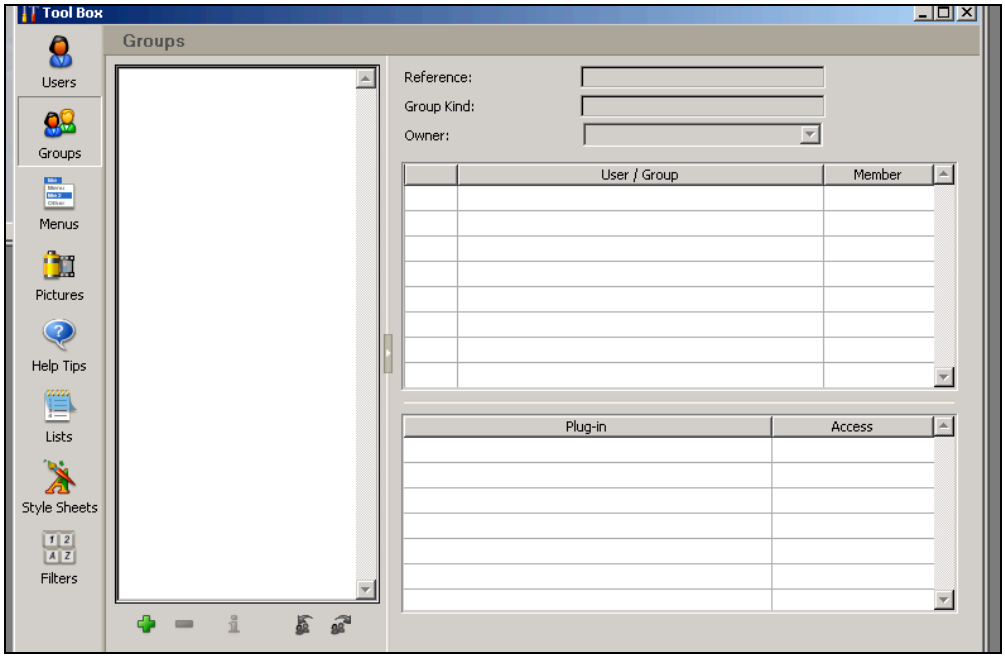
This demo will set plug-in access settings for the following plug-ins (assuming they are installed, licensed, and not disabled):

- 4D Draw
- 4D For OCI
- 4D View
- 4D Write
- 4D Client Web
- 4D Client SOAP
- 4D ODBC Pro
- 4D for ADO
- 4D for MySQL
- 4D for PostgreSQL
- 4D for Sybase

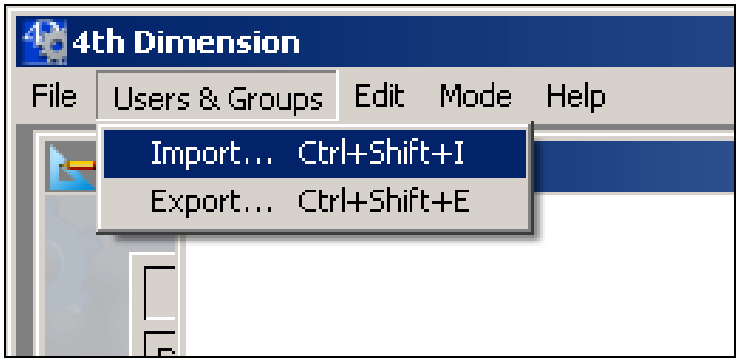
The list of possible plug-ins is defined in the documentation for the Get plugin access command.

Demo steps:

- First add a Designer password so that the password access system will be activated. Also notice there are no groups yet defined:

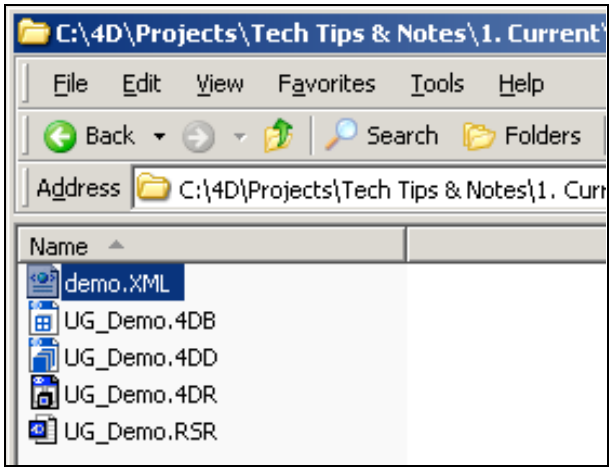


- Restart the database.
- Login as Administrator (there should be no password).
- Open the "Users & Groups" menu and select "Import...".



- This opens the import dialog:

- Browse for the file "demo.XML" (it is next to the structure).



- Check the "Import Plug-In Access?" check box.

- Click the "Import" button.

Once the import is complete there should be a message next to the "Explorer Status :" label. The color of the message text indicates the success of the import (or export) as follows:

- **Green**: successful or benign messages.
- **Orange**: warnings.
- **Red**: errors.

This is what a successful import will look like:

If the import completes successfully go to the Tool Box in Design Mode and verify which plug-in access settings were imported. There will be some new users and groups as well.

If no plug-in access settings were imported a warning message is displayed. This means either the plug-ins are not installed, are disabled, or are not licensed.

Of course if the import failed there will be an error message.

Conclusion

This Technical Note described how to build a customized users and groups import/export module that exports both 4D Users and Groups as well as plug-in access settings. The source code was provided in an example database as well as a demo database.