Developers instructions for extending PxWeb

# Readers guide

## What this instruction covers

This manual covers how PxWeb could be extended by writing your own code and plug it in to PxWeb.

PxWeb provides a set of different interfaces that can be implemented to customize some of its behavior. This document tries to describe some of these interfaces.

## Terminology

* *Paxiom* the object model representing a statistical cube
* *table* a multidimensional table containing statistical measures and metadata about these measures stored in a PX-File or a database using the CNMM.
* *CNMM* the Common Nordic Meta Model which is the database structure that is used and maintained by Statistics Sweden, Norway and Denmark.

# Interfaces

## IAuthorization

**Assembly**: PX.Security

**Namespace**: PX.Security

The aim for IAuthorization interface is to provide the possibility to implement a custom authorization function to one or more databases in PxWeb.

The authorization in PxWeb is separated from the authentication. Which could be describe as the process where the user have to provide evidence that he/she is who that claim to be e.g. with a username and password.

**Notice!**

PxWeb does not come with any build in method for authentication other from what is built in ASP.NET. This mean that you have to provide this by yourself. E.g. if you would like to use Windows authentication you have to configure IIS to use it see <https://technet.microsoft.com/sv-se/library/cc754628(v=ws.10).aspx> or if you would like to have Forms authentication you would have to provide a small login application yourself which sets the authentication cookie for PxWeb, see the implication at <https://msdn.microsoft.com/en-us/library/eb0zx8fc.aspx>

The interface consist a two methods one that take a database Id and checks if the user is authorized to view the database. The other methods takes a database id an menu and a selection and determines if the user is authorized to view a level in the database or if the user is authorized to see a table if the selection is pointing on a table.

**public** **interface** IAuthorization

{

**bool** IsAuthorized(**string** dbid, **string** menu, **string** selection);

**bool** IsAuthorized(**string** dbid);

}

**Example**

This an fictive example that is not very useful in real life but shows how an implementation may look like. What it does is that it will prohibit calls to PX files that starts with the letter P unless the call is coming from an IP address that is in a whitelist of IP addresses.

**namespace** MySuperSecureCode

{

**public** **class** MyPFileProtector : PX.Security.IAuthorization

{

**private** HashSet<string> \_whitelist;

**public** MyPFileProtector()

{

\_whitelist = new HashSet<string>();

*//TODO add IP-addresses to the whitelist from*

}

**public** **bool** IsAuthorized(**string** dbid, **string** menu, **string** selection)

{

*//If the file/table starts with P then we shall protect it*

**if** (selection.StartsWith("P",

StringComparison.InvariantCultureIgnoreCase))

{

**if** (\_whitelist.Contains(

HttpContext.Current.Request.UserHostAddress))

{

*//We are clear*

**return** **true**;

}

**else**

{

*//Not in the whitelist deny access*

**return false**;

}

}

*//If not a P file it should be ok to view it.*

**return true**;

}

**public** **bool** IsAuthorized(string dbid)

{

*//I think that everyone should be able to see the database*

**return true**;

}

}

}

To hook up the authorization implementation one have to manually edit the database.config file for the database that should be using it.

Turn on the protection by setting the isProtected element to True and set the auhotizationMethod to point to your implementation.

Then you also have to drop the assembly containing your implementation in the PxWeb bin folder.

<protection>

<isProtected>True</isProtected>

<authorizationMethod>

MySuperSecureCode.MyPFileProtector, MySuperSecureCode

</authorizationMethod>

</protection>

**Notice!**

The API currently does not support authorization so if you have a database that should be protected make sure that the API endpoint is disabled for that database.

## IMetadIdProvider

**Assembly**: PCAxis.Metadata

**Namespace**: PCAxis.Metadata

The aim for the IMetaIdProvider interface is to provide a way to transforms the MetaId property of the Table, Variable and Value in Paxiom to URL:s to one or more metadata systems.

**public** **interface** IMetaIdProvider

{

**bool** Initialize(**string** configurationFile);

MetadataSystem[] MetadataSystems { **get**; }

MetaLink[] GetTableLinks(**string** metaId, **string** language);

MetaLink[] GetVariableLinks(**string** metaId, **string** language);

MetaLink[] GetValueLinks(**string** metaId, **string** language);

}

**Example**

This is another example implementations which assume that we have a metadata system called EMS and that is accessible from Internet on the URL http://ems.myorg.org and you could address different type of entities e.g. table, variable and values on the form <http://ems.myorg.org/ENTITY-TYPE/ID>. We also assume that MetaId contains the Id to EMS. This is what an implementation may look like.

**namespace** EMS

{

**public** **class** EMSMetaIdProvider : PCAxis.Metadata.IMetaIdProvider

{

**public** **bool** Initialize(**string** configurationFile)

{

*//We dont use a metadata.config file*

*return true*;

}

**public** PCAxis.Metadata.MetadataSystem[] MetadataSystems

{

**get** { **return** **new** PCAxis.Metadata.MetadataSystem[]

{ **new** PCAxis.Metadata.MetadataSystem("ems", "EMS") }; }

}

**public** PCAxis.Metadata.MetaLink[] GetTableLinks(**string** metaId,

**string** language)

{

**return** **new** PCAxis.Metadata.MetaLink[] {

**new** PCAxis.Metadata.MetaLink() {

Link = "http://ems.myorg.org/table/" + metaId,

LinkText = "Go to EMS",

System = "ems",

Target = "\_blank" } };

}

**public** PCAxis.Metadata.MetaLink[] GetVariableLinks(**string** metaId,

**string** language)

{

**return** **new** PCAxis.Metadata.MetaLink[] {

**new** PCAxis.Metadata.MetaLink() {

Link = "http://ems.myorg.org/variable/" + metaId,

LinkText = "Go to EMS",

System = "ems",

Target = "\_blank" } };

}

**public** PCAxis.Metadata.MetaLink[] GetValueLinks(**string** metaId,

**string** language)

{

**return** **new** PCAxis.Metadata.MetaLink[] {

**new** PCAxis.Metadata.MetaLink() {

Link = "http://ems.myorg.org/value/" + metaId,

LinkText = "Go to EMS",

System = "ems",

Target = "\_blank" } };

}

}

}

Enable the IMetaIdProvider by configuring the database.config file for the database that should use the provider. In the metadata section set the useMetadata element to True. If you have a configuration file set the metaSystemConfigFile element to relative location to that file. Also set the metaLinkMethod to point to the IMetaIdProvder implementation. Finally place the assembly containing the implementation in the bin folder of PxWeb.

**Example**

This is what the configuration may look like for the example above.

<metadata>

<useMetadata>True</useMetadata>

<metaSystemConfigFile></metaSystemConfigFile>

<metaLinkMethod>EMS.EMSMEtaIDProvider, EMS</metaLinkMethod>

</metadata>

## IActionLogger

**Assembly**: PCAxis.Web.Controls

**Namespace**: PCAxis.Web.Controls.Managment

The aim of IActionLogger interface is to provide a way to customize how to log user interaction in PxWeb for the purpose user statistics.

The way it works is that the different web controls that PxWeb is made up of are firing events to notify what they are doing. E.g. the CommandBar might fire an event to notify that the user has selected to download the table as an PX file. These events are caught by PxWeb. PxWeb then handles the event by calling the LogEvent on the IActionLogger.

**public** **interface** IActionLogger

{

**void** LogEvent(ActionContext context, **string** userid, **string** lang,

**string** database, PxActionEventArgs e)

}

Implementing the interface is pretty straight forward the you get all the information passed in as parameters the it is up to you to implement what information you want to persist and how to persist it e.g. to a database or to a file etc.

The information that is passed is the

* context - which either is the selection page or the presentation page.
* userid - this is currently a hardcoded string **userid**.
* lang - the current selected language for the user.
* database - the database Id form which the table have been extracted from.
* e – the PxActionEvent that the web control fired which contains the
  + action type
    - Operation – if an operation has been applied on the table.
    - Presentation – if the user has selected to display the table in various form e.g. a table or a chart etc.
    - Save as – if the user has selected to download the table as a file.
  + action name - which depends on the type e.g. if the type is SaveAs the action name would be the file format.
  + tableId – the table Id for the extracted table.
  + Number of selected cells – the number of cells in the extracted table.
  + Number of contents – the total number of contents selected.

When you have your implementation ready and want to use it in PxWeb you have to add com configuration in Web.config in the appSettings section

**Example**

<add key="visitorStatisticsLogger"

value="PXWeb.Ssd.VisitorStatistics.VisitorStatisticsLogger,

PXWeb.Ssd,

Version=1.0.0.0,

Culture=neutral"/>

The default implementation of the IActionLogger logs the

* Context
* User Id
* Language
* Database
* Action type
* Action name
* Table ID
* Number of cells
* Number of contents

Using log4net to a logger called PCAxis.Web.Controls.PxDefaultLogger This logger uses per default a rolling log file appender called visitorStatisticsAppender. But since it uses log4net you could easily change it e.g. to log the information in a database by changing the log4net configuration. So before you start implementing your own IActionLogger you might what to ta a look on the official documentation of log4net at <https://logging.apache.org/log4net/release/config-examples.html> four various kinds of configurations options that might suite your needs.

**Notice!**

PxWeb will only log the UI interaction by utilizing the IActionLogger implementation.

API usage is only logged with log4net with the api-usage logger.

## ISearchIndex

**Assembly**: PCAxis.Search

**Namespace**: PCAxis.Search

The aim of the ISearchIndex interface is to give a way to do partial updates of the search index for an database.

The interface has only one method that has to be implemented. Which takes three parameters. The idea is that PxWeb calls this method passing in a date, a databaseId and a language and in return it expects a list of tables that has been updated since the date specified for the given database and language.

**public** **interface** ISearchIndex

{

List<TableUpdate> GetUpdatedTables(DateTime dateFrom, **string** database,

**string** language);

}

**Notice!**

PxWeb does not currently support doing partial updates of the search index though it might do that in the future.

### TableUpdate

**Assembly**: PCAxis.Search

**Namespace**: PCAxis.Search

This is the return type of the GetUpdatedTables method which has two properties. One for the Path (e.g. the relative path to the PX file) to the table and one for the Id (e.g. the name of the PX file) of the table.

**public class** TableUpdate

{

**public string** Path { **get**; **set**; }

**public string** Id { **get**; **set**; }

}