Microsoft Dynamics® AX 2012

Role-based security use patterns for developers

White Paper

This white paper describes how the role-based security artifacts work together. It explains to Microsoft Dynamics AX developers how the security artifacts should be configured when they are applied to various programming objects in the AOT. The challenges of under-permissioning, over-permissioning, and practical ease of use guide this discussion.

Gene Milener, Senior Programming Writer
Mark Skunberg, Senior Software Engineer
May 2013

http://go.microsoft.com/fwlink/?linkid=303934

Table of Contents

1. Introduction

This white paper describes the interactions among the various artifacts of the role-based security system that is new in Microsoft Dynamics AX 2012.

The discussion next moves to an explanation of how Microsoft Dynamics AX partner developers should apply the security artifacts to various programming objects, such as forms, reports, and service operations.

The discussions are framed by the competing risks of giving too little versus too much access to sensitive data.

2. Overview of security artifacts

Permissions

A permission is the simplest security artifact. Sets of permissions are usually grouped into a privilege (or we can say that the permissions are referenced by the privilege).

A permission is the combination of two things:

- A securable object, such as a form or a control in a form, a report, or a service operation
- A degree or level of access to the securable object, such as read or update

In the Application Object Tree (AOT), permissions are found under the **Permissions** node of some programming objects, such as a form or report:

- Various sets of permissions are automatically inferred by the system, saving you the effort of creating them.
- Each permission set is designed for one level of access, such as read or update.

  However, the developer can adjust individual permissions in any set if necessary.
- Developers decide which permission set should be associated with each entry point when they create a privilege under **AOT** > **Security** > **Privileges**.

  An entry point can reference a menu item, a service operation, or certain other item types that are starting points for application functionality.

Hierarchy of permissions

The following list gives the names of the levels that any given permission can represent. The levels are listed in order of their scope.

1. Read
2. Update
3. Create
4. Correct
5. Delete
6. NoAccess

Consider the following examples of this hierarchy:

- Read permission lets the user see data but not change it.
- Create permission lets the user insert new data records; plus, it gives the user Update and Read permissions.

Artifacts under AOT > Security

With the exception of permissions, all the role-based security artifacts can be viewed immediately under **AOT** > **Security**.

Hierarchy of artifact references

The security artifact types follow a progression, from permissions to roles:

1. One role can include other roles.
2. One role is a group of several duties.

3. One duty is a group of several privileges.
4. One privilege is a group of permissions, plus one or more entry points.

Therefore, among security artifacts, the general hierarchy is represented as follows:

Permission < Privilege < Duty < Role < Role...

Duties

A duty defines the functionality for a business process – for example, maintain bank accounts. A duty can only contain privileges, but it can belong to more than one role.

It is rare for a duty to become obsolete, or to be deleted after it starts to be used. Some of the privileges in a duty might be removed from the duty, and other privileges might be added. However, the duty itself, and its relationships to roles, are stable as the months and years pass.

Privileges

A privilege defines the functionality for a specific business function – for example, maintain bank statement. A privilege contains permissions, plus one or more entry points.

Roles

A role is a combination of users (or legal entities) and a set of duties.

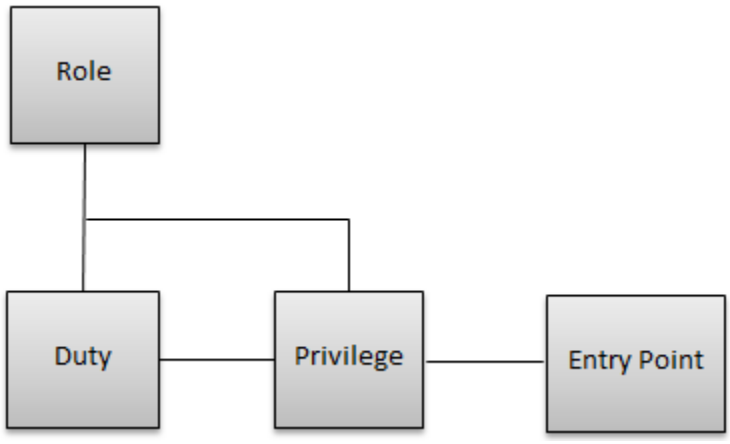
A user's access to data, forms, and other securable objects is controlled by the roles that the user is assigned to, and by the security artifacts that are also assigned to those roles.

For easier maintenance, we recommend that you assign duties rather than privileges to roles. Assign privileges to a duty, and then assign the duty to a role.

Entry points

An entry point references a programming object that is at the start of an application functionality.

Entry points can be directly associated with privileges.



When newly installed, Microsoft Dynamics AX 2012 includes many predefined roles and duties. The Microsoft Dynamics AX partner developer, or the system administrator, can add to these or modify them. The role-based security system is designed for flexibility, but also for ease of use when that flexibility is leveraged.

System administrators can customize these roles and duties by using the form at **System administration** > **Setup** > **Security** > **Security roles**.

Chain of associations

Through its **ObjectType** property, an entry point can reference any of the following securable item types:

- **Class**
- **CueGroup**
- **Form**
- **FormPart**
- **Job**
- **InfoPart**
- **Query**
- **Report**
- **SQLReportLibraryReport**
- **SSRSReport**

Example

An entry point references a menu item that, in turn, references a form. Permissions are defined under the form's node in the AOT:

Permission < Form < Menu item < Entry point (on a privilege)

1. A permission under the form can reference one control in the form, which gives the developer fine granularity in the security design.

   The form has a different permission set for each of the four CRUD access types.

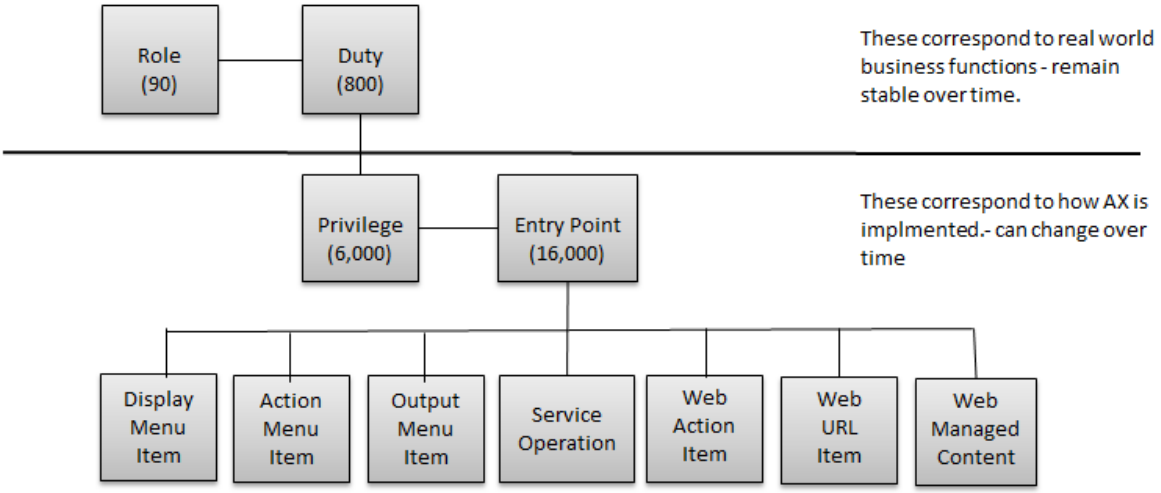
   (Sometimes, there is also a **Correct** access type, which is related to valid time state tables.)

2. On the menu item, the following properties govern which CRUD permission sets of the form will be available to the entry point:

   - **CreatePermissions**
   - **DeletePermissions**
   - **ReadPermissions**
   - **UpdatePermissions**

3. On the entry point, the **AccessLevel** property in essence selects one of the permission sets on the form.

3. Security boundaries and resources



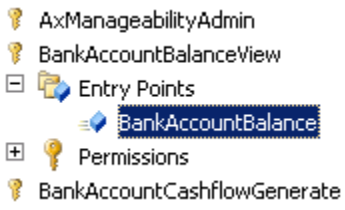
Over time, application functionality will increase and change. Therefore, the developer must occasionally add and remove individual permissions from a given privilege. This churn of permissions is why the "Privilege" box is below the line in the preceding diagram.

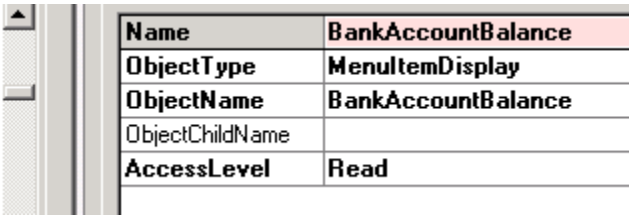
The privilege is assigned to a duty. This assignment is unaffected by the churn of permissions within the privilege. To reduce maintenance burdens, it is better to assign duties rather than privileges to roles. That is why the "Duty" box is above the line in the diagram.

Securable boundaries

Each entry point represents one gate in the security fence that the system builds around your sensitive data. Each entry point references a programming object that can access data, such as a form, report, or service operation.

A given user can see the entry point only if the user has authorization to invoke the entry point. The user obtains authorization by being assigned to a role that also has a privilege that includes the entry point.





Name	BankAccountBalance
ObjectType	MenuItemDisplay
ObjectName	BankAccountBalance
ObjectChildName	
AccessLevel	Read

Securable resources

When a user uses an entry point to invoke a programming object, the security system examines the permissions to ascertain that the user has the necessary authorization to each of the tables and other resources that the programming object accesses. An error occurs at any point where sufficient permission is not found. We call this an "under-permission" error.

Example

When a form opens, the forms engine asks the security system for the table access permission levels for each of the form data sources. The forms engine compares the permission level information to the form data source metadata. The forms engine can then calculate whether it should display and enable all its controls that are connected to the form data source. This enables a single form to open either with full CRUD functionality, or with only a subset of CRUD functionality on some or all data sources.

Form methods

Suppose a user has access to a button in a form. When the user clicks the button, that event invokes a method on the form, and that method calls a method on another class that accesses several tables. Usually, the security system does not check whether the user has authorization to access those tables. Instead, the security system only checks that the user has authorization to click the button control in the form. Here we say that the other tables are accessed "on behalf of the user".

However, the Table Protection Framework (TPF) represents an exception to this usual behavior.

Table Protection Framework

The exception is that the security system always checks permissions for any tables that have their **AOSAuthorization** property set. The **AOSAuthorization** property is part of the TPF.

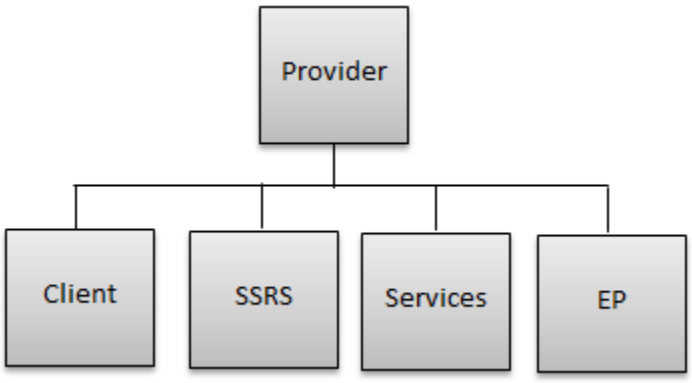
We add a table to the TPF when the table contains extra-sensitive information. The **AOSAuthorization** metadata lets us control whether the application always verifies that the user has explicit access.

Trimming

If a table has its **AOSAuthorization** property set to **CreateDelete**, but the user is authorized for only the relatively weak access level of Read, the security system prevents the user from seeing the data in that table. As part of this prevention, Application Object Server (AOS) discards or "trims" the data that SQL selected from the field, so that the results that are returned to the client lack the trimmed data.

4. Auto-inference of permissions

The security systems of Microsoft Dynamics AX assist the developer and reduce the work needed to implement security.



Some auto-inferred permissions are located at AOT navigation paths that resemble the following:

AOT > **Forms** > ***MyForm*** > Permissions

- The system has an auto-inference provider that generates sets of permissions by using reflection on the programming objects (such as forms, reports, and service operations).

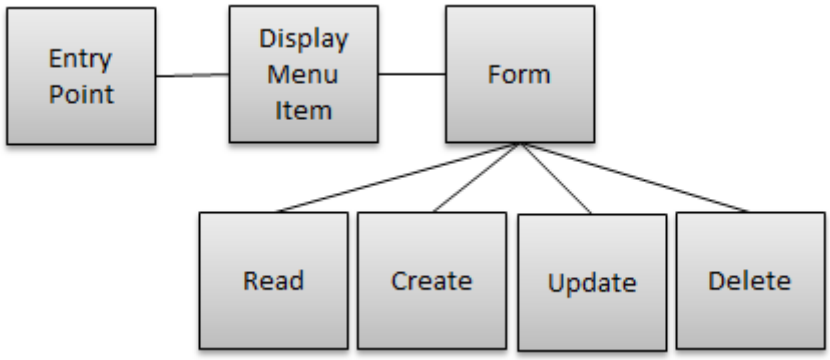
  The developer has the option of assigning these auto-inferred permissions to privileges, instead of having to create the permissions manually.
- The framework utilizes the provider pattern to reflect on X++ artifacts (forms, reports, service operations, and so on) to intelligently determine the security resources that are necessary.
- For simple and moderate cases, the auto-inferred permissions are complete.
- The information generated from this reflection is not saved with the artifact. Rather, it is persisted in the security database store and is visible in the AOT.
- Each provider uses existing metadata and internal rules to infer the correct resources and access level.
- The providers work nicely for all layers, because they automatically recalculate in real time as the developer changes AOT metadata. Therefore, the system does the work for the developer.

5. Security patterns for AOT objects

The diagrams in this section emphasize the relationships between security artifacts and AOT objects.

Forms

For each form, four sets of permissions are auto-inferred: Read, Update, Create, and Delete.



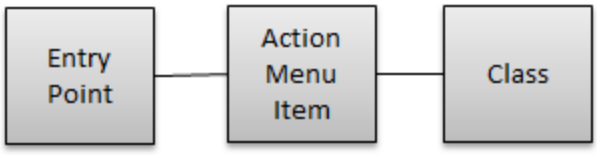
- Typically, the **AccessLevel** property on the entry point is assigned to use the Read or Delete permission set. These are the weakest and strongest sets, respectively.

  The access level is specified at **AOT** > **Security** > **Privileges** > ***MyPrivilege*** > **Entry Points** > ***MyEntryPoint*** > **AccessLevel**.

- **Display** and **Edit** methods (which are bindable to controls) execute without further security verification. Therefore, in some cases, it might be prudent for the developer to force a security check by calling the method **Global::hasTableAccess**. If the method call determines that the user lacks the necessary permissions, the code should not return the data to the user.

Classes

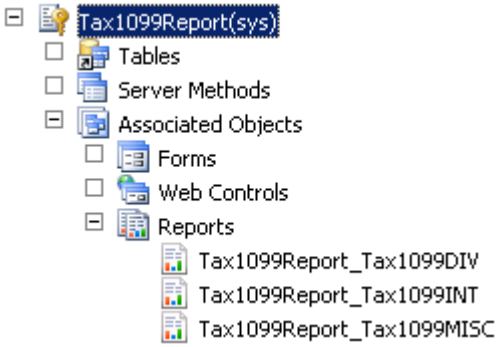
Methods on classes are typically used to start a process, form, or report. **We do not have a class auto-inference provider**, because this would involve interpreting X++ code. That is the developer's responsibility.



- Typically, the entry point access level is set to Delete.
- Use the menu item–linked permission type to include a single form or report.

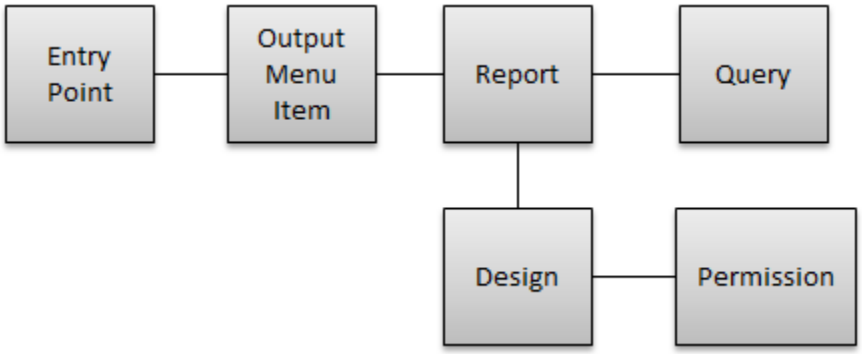
- Use a code permission to include forms, reports, server methods, tables, and web controls.

  See under **AOT** > **Security**> **Code Permissions**.



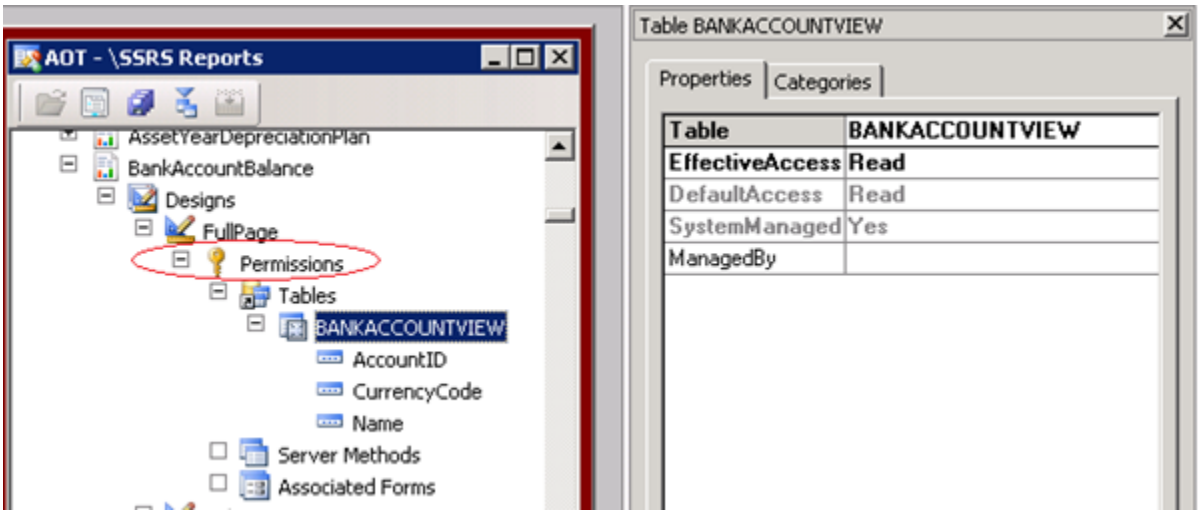
- After the **Linked*** properties are set on the entry point, the security system can examine the programming object and the chosen access level.

Reports – Query pattern

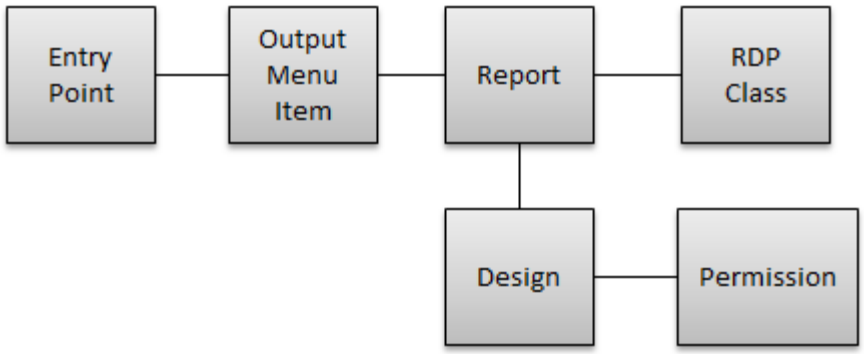


- The entry point access level is set to Read.
- The report auto-inference provider calculates the necessary tables and table access permissions.

- Dialogs for dynamic parameters do not have a security impact.



Reports – Remote data processor pattern



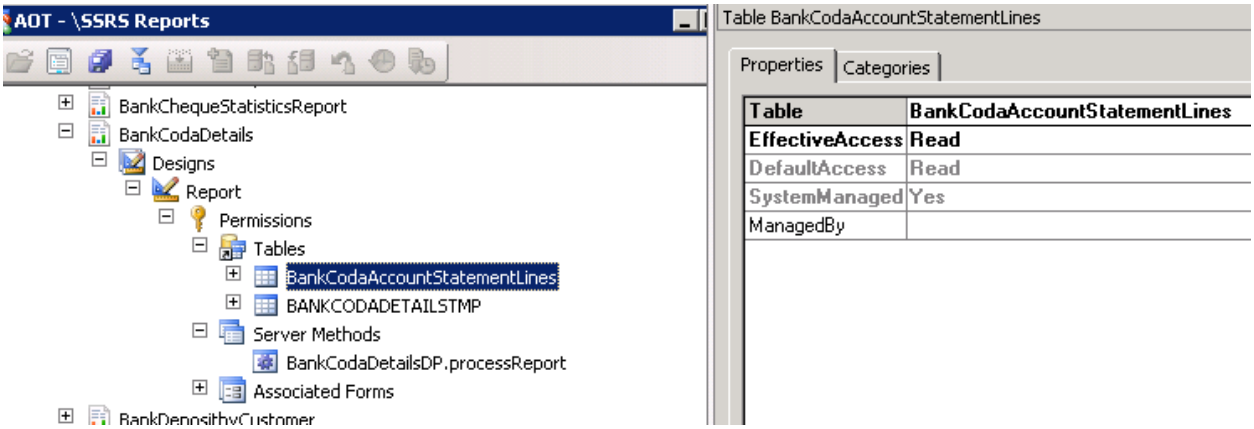
- The remote data processor (RDP) class extends the **SRSReportDataProviderBase** class.
- The entry point access level should be set to Read.
- Define an unchecked server entry point on the **processReport** method.

  This is done by passing the value **false** to the constructor of **SysEntryPointAttribute**, as decoration on the **processReport** method.
- A server entry point ensures that the user has access to this specific method before it can be executed.

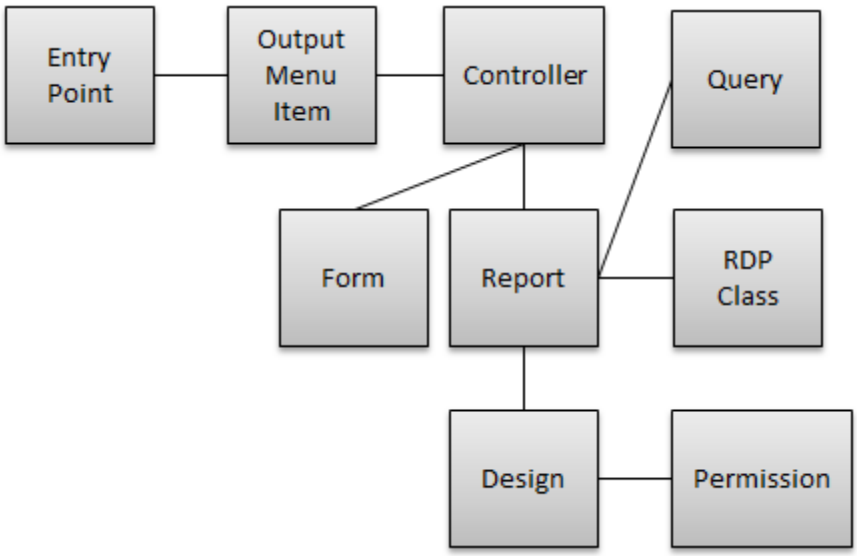
```
[SysEntryPointAttribute(false)]
public void processReport()
{
}
```

- The report auto-inference provider defines access to the server entry point, the necessary tables, and the table access levels.
- These auto-inferred permissions are just one set. Unlike forms, reports do not have Read, Update, Create, and Delete permission sets auto-inferred for them.

- Dialogs for dynamic parameters do not have a security impact.



Reports – Controller pattern



- The controller class extends the **SysReportRunController** class.
- The entry point access level is set to Read.
- We **do not have an auto-inference provider**, because this would involve interpreting X++ code.
- If the report is implemented with an RDP, you will need to define the **processReport** method as an unchecked server entry point.
- Use the menu item–linked permission property, or a code permission, to include the report and a possible form dialog.

Service operations

The service operation entry point method needs to be marked as checked by decorating the method with the **SysEntryPointAttribute(true)** attribute. Microsoft Dynamics AX will enforce that all callers have been granted access to this method.

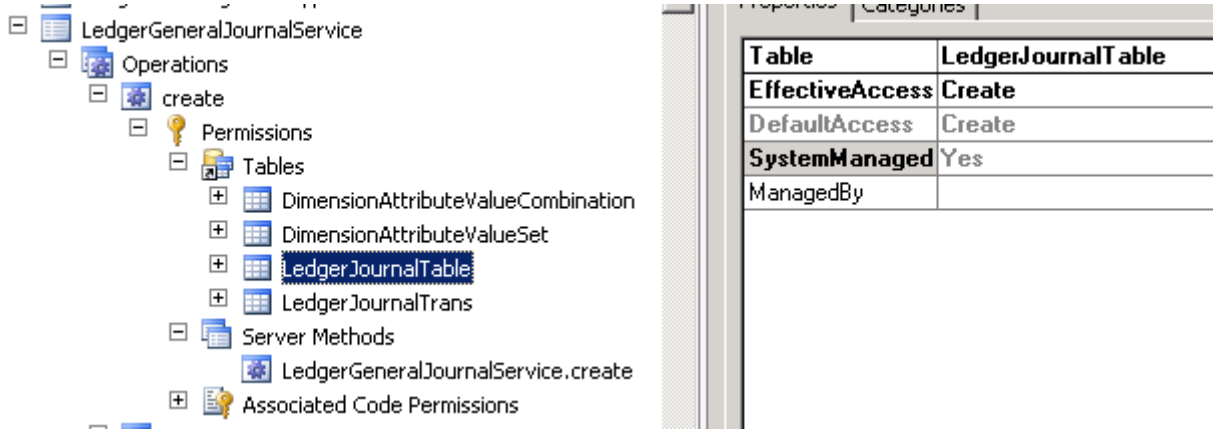
Document services pattern

- For the permission, the application developer only has to set the **SysEntryPointAttribute** attribute. The Application Integration Framework (AIF) auto-inference provider will automatically define access to this server entry point, and will automatically determine all the appropriate tables and access levels.
- The **find** and **read** operation methods should be marked as checked to verify that the user has access to all the tables that are returned to the user in the graph.
- The **create** and **update** operations should be marked as checked to ensure that the deserialization of the message does not contain elements that the user lacks authorization to.
- After deserialization, AIF will elevate privileges for the user, so that the application business logic can run unchecked.
- Currently, the **findKeys** and **delete** operations have to be marked as checked, because the design wanted to ensure that the user has access to all the tables.

Checked example

```
[AifDocumentCreateAttribute, SysEntryPointAttribute(true)]
public AifEntityKeyList create(VendTable _vendTable)
{
}
```

The following screenshot was captured under **AOT** > **Services**.



Custom X++ services pattern

The AIF auto-inference provider defines access levels to this server entry point.

To determine the range of permission verification, the application developer must apply the **SysEntryPointAttribute** attribute on the X++ method and consider table access levels. The attribute is constructed with a Boolean parameter value:

- **true** – Security is *checked* during execution.
- **false** – Security goes *unchecked* during execution.

Recommendation: Most custom services can be designed as *unchecked*. This choice means that the developer has no additional security work.

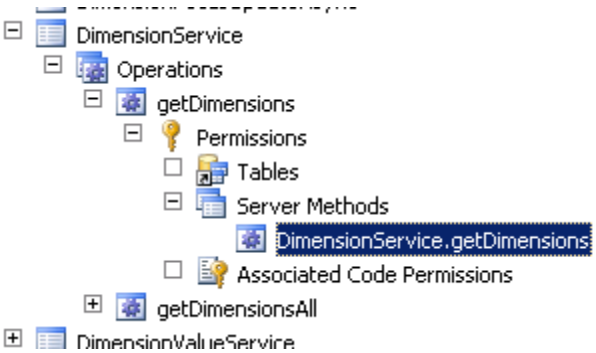
- The *unchecked* mode means that the developer should add explicit logic to the service implementation to verify that the user has explicit permission to access any sensitive fields.

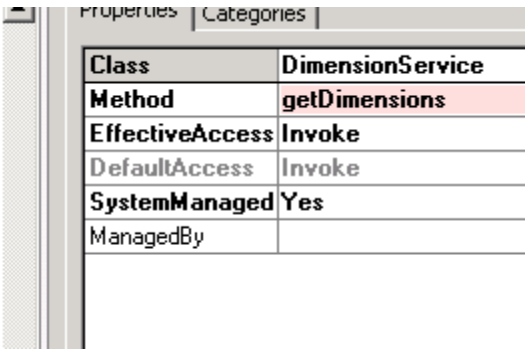
  The method **Global::hasTableAccess** can be useful here.
- The *checked* option can be considered when both the following conditions are met:
  - The number of tables that the service uses is small, and is stable over the months and years.
  - The service operation exposes a data contract class with sensitive input information.

Unchecked example

In *unchecked* mode, the attribute is constructed with a parameter value of **false**.

```
[AifDocumentCreateAttribute, SysEntryPointAttribute(false)]
public List getDimensions(AccountStructureContract _accountStructureContract)
{
}
```





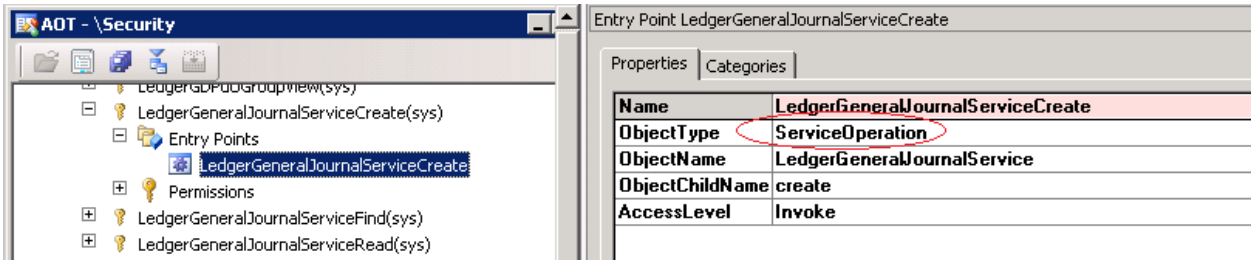
Security operation permissions pattern

For this privilege pattern, we create one privilege for each service operation. We name the privilege by using the pattern *ServiceName+OperationName*.

For service operations that are consumed *outside* the Microsoft Dynamics AX application, we add these privileges to the ServiceOperations *duty*. This allows system administrators to configure the privileges as appropriate for their integration scenarios.

For service operations that are consumed *inside* the Microsoft Dynamics AX application, we must expose the operations at the most appropriate place in the security model.

The following AOT screenshot shows a service operation being exposed through an entry point that is under a security *privilege*.



6. Advanced topics

Table Protection Framework

X++ application code can access tables on behalf of the user, so that the user does not need explicit access to every table. TPF was introduced many releases ago to control which users can access tables that contain high business impact (HBI) data. The kernel verifies that the user has explicit access to these tables *everywhere* they are accessed.

- By setting the **AOSAuthorization** property on a table, the developer lists the range of CRUD operations that security authorization checks are always performed for. The value **CreateReadUpdateDelete** is the most secure setting.

- On a field, if **AOSAuthorization** is set to **Yes**, the security check is performed before the data from the field can be sent to the user's client. If the check determines that the user is not authorized to see the field, the field is "trimmed" from the results before those results are sent to the client.

- We expose TPF tables at the **role level** with the least access necessary.
- If you want to ensure that only an authorized group of users can create, update, and delete information, but that everyone else can read the information, you set the table's **AOSAuthorization** property to **CreateUpdateDelete**. This causes the system to check for table permissions when the user tries to create, update, or delete information in the table.
- By default, granting a role access to a table grants access to all the fields in that table.
- For new tables and fields, consider whether any should be treated as HBI fields and made extra secure. To achieve the extra security, do the following:
  - Sett the **AOSAuthorization** property at the table and HBI field levels.
  - Withhold security permission to the HBI field from appropriate roles.

Server entry points

A server entry point (SEP) protects a server method. Any user who accesses this method must have explicit access to invoke the method.

- The method can be defined as checked or unchecked.
  - **Checked** – Verifies that the user has the appropriate CRUD access to *every* table that is invoked by the method.

    To avoid an oppressive amount of careful labor and maintenance, use checked mode sparingly.

The query web service provides a good example of when to use this option:

- The request for data comes from outside the Microsoft Dynamics AX system.
- Users should be able to query only the information that they have been granted access to.

- **Unchecked** – Authorizes the user to access all the tables that the method's code operates on, even when the user lacks explicit permissions to the tables. **Recommended**.

- A server entry point is required for service operations and RDP reports.
- Typically, an SEP is created for a server business process, so that a user does not need explicit access to long lists of sensitive tables.
- The SEP itself must be associated with the privileges that are necessary to access the method.

Trimming

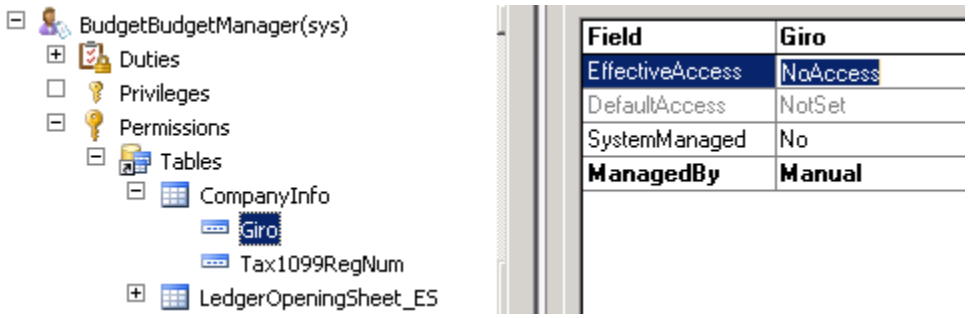
If a user does not have explicit access to a HBI field, the Microsoft Dynamics AX kernel will remove or "trim" the field from the buffer before the buffer leaves the server on its way to the client.

- When a role has explicit permissions to a TPF table, the role has access to all the HBI fields in the table.

  However, the administrator can override this behavior.

- When only a specific set of roles should have access to the HBI fields, the developer must explicitly remove the HBI fields from the other roles.
- The following screenshot is from under **AOT** > **Security** > **Roles** > **BudgetBudgetManager**.

  Notice that **EffectiveAccess** is set to **NoAccess**.



- The rich client and Enterprise Portal for Microsoft Dynamics AX (EP) automatically adjust the user interface according to the **EffectiveAccess** setting.
- The kernel performs updates that might skip a secure subset of all the fields that the method attempts to update. Therefore, the developer must pay attention to scenarios where one field is validated based on the value in another field.
- During development of Microsoft Dynamics AX 2012, our detailed code analysis across our entire internal code base, for all TPF tables and HBI fields, discovered minimal need for code refactoring.

  Generally, if the application uses an HBI field, the user should be granted access to it.

  Under-permissioning in this case can lead to a lot of ongoing security maintenance.

Cross-company queries

Cross-company queries are used to read data from more than one company. The kernel verifies that the user has explicit access to the data for each queried company, in all the tables involved. The Microsoft Dynamics AX kernel silently discards results when the user lacks sufficient access.

- Because these queries are often deep inside the application, auto-inference will not find the tables and grant access.
- **Recommendation:** Manually add the tables to the few privileges that reference the appropriate entry points.
- Consider which roles the user needs in the other companies. For example, for centralized cross-company payment, we created roles to only read invoice information.

Policies

A security policy is like a Where clause in an SQL **Select** statement. When a Where clause is added to a **Select** statement, the result is usually a reduction in the count of data rows that are returned (the count is never increased).

When a policy is assigned to a role, the users in that role are restricted to seeing only the data that is not filtered out by the policy. The filtering occurs at the AOS tier, before any data is returned to the client.

For more information, see http://www.microsoft.com/en-us/download/confirmation.aspx?id=3110.

Securable controls

Unbound controls can be buttons, command buttons, dynamic controls, and list views. Unbound controls can let the user change data even when the form is open for read-only mode. We use securable control options to restrict access to these controls. We also use securable controls to grant access to sensitive information only in specific scenarios.

Consider the following points:

- Use the control's **NeededPermission** property to define when the control should be enabled.
  - The client forms engine will grant access via the form permissions and configure the control correctly for CRUD settings. **Recommended**.
  - The manual option lets developers control how the control is granted access to the security model.
- **Recommendation:** Test in inquiry mode, and see whether the unbound controls can modify data.

- Dimensions will dynamically add controls to empty tab pages or groups. We make the container (that is, the tab page or group) a securable control, because the setting cascades to all contained controls.

7. Additional security concepts

Form auto-authorization

When a control in a form reads from a lookup table, the client forms engine usually does not call for a security check to determine whether the user is authorized to see the table's data. The exception is when the lookup table has been added to the TPF through assignment to the table's **AOSAuthorization** property.

The Microsoft Dynamics AX team chose this security design for the following reasons:

- When users have Create, Update, or Delete permission to a form, by default they should have access to all the lookups.
- It reduces the complexity and maintenance cost of the security model.
- It prevents users from getting read access to the entire lookup table.

Process cycles

A process cycle is a set of duties.

The main use of process cycles is to help system administrators find relevant duties more quickly when they are modifying role configurations.

Process cycles have names that match major areas of business, such as RetailMerchandising.

Process cycles represent general business concepts that apply to many or most businesses and organizations. Process cycles are even more stable than duties.

8. Over-permissioning

Earlier portions of this white paper concerned issues around under-permissioning. We made recommendations about which permissions level should be granted when you develop with various programming objects. Some of these recommendations were explained by listing the problems that weaker permission levels might cause.

Now we take the opposite perspective and look at over-permissioning. The following sections explain various scenarios and patterns where too much access has been granted to roles that do not need the access and should not have it.

Pattern: One menu item reused many times

A given action menu item is invoked from many different places in the application. If some of those places need stronger security permission than others, the application is over-permissioned.

A simple solution is to create a separate action menu items for each security context. Expose each menu item to the security model as different entry points.

Pattern: One form with many contexts

A given form may be invoked by many menu items. Each menu item provides a different context to the form.

An example is accounting distributions, where one accounting distribution form is used for many document contexts.

A user can have both Read and Delete permissions to a given table:

- In the role-based security model, the role can have Read permission on a given table from one privilege, plus Delete permission on the same table from another privilege.
- Alternatively, a given user might be in multiple roles, where one role has the Read privilege and the other role has the Delete privilege.

During run time, the security system calculates a user's final access level by a *union* of all the user's permissions. Therefore, even when the user clicks a menu item that is intended for Read only, the user has Delete access to the table while using the form.

How does the developer prevent a given form from offering full access when the form was called by a menu item that is intended to merely read the data? You can force the Microsoft Dynamics AX system to use the security permissions that are *granted with the calling menu item*. Place the following line of code in the form's **init** method, after the call to **super**:

```
FormSecurity::setFormDataSourceMaxAccessRight(this);
```

Pattern: Many forms sharing the same table

This is somewhat similar to the preceding pattern.

An example is financial journal lines. The data for the financial journal lines is kept in one table, but each of the journals has a different journal line form.

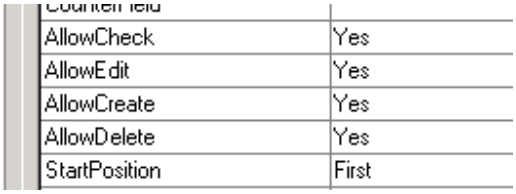
The solution is, again, to force the Microsoft Dynamics AX system to use the security permissions that are granted by the calling menu item. Place the following line of code in the form's **init** method, after the call to **super**:

```
FormSecurity::setFormDataSourceMaxAccessRight(this);
```

Pattern: Form data source access elevation

The developer should ensure that the form's data sources have their **Allow*** properties set to allow only the minimum access that is necessary for the form to function. For example, if the form never needs to delete data from a given data source, the **AllowDelete** property on that data source should be set to **No**.

The forms auto-inference provider uses these property values to determine the level of access for the form permission. Permissions that are unnecessarily elevated can expose more functionality than necessary in other forms. This is because the security model unions table access level together and uses the highest level everywhere.



Pattern: One form with many contexts, each of which exposes common polymorphic behavior

One form can be opened from a variety of different contexts and can provide similar behavior across all those contexts.

Example

The **LedgerJournalTable** form exposes a single post menu item button. Depending upon the context, the form must define the appropriate posting menu item functionality.

Solution

Create different menu items for the various common behaviors. Based on the calling context, dynamically set the menu item name:

```
post.menuItemName(menuitemActionStr(LedgerJourPostLJTransDaily));
```

Pattern: Nested contexts

A form that can be opened from a variety of different contexts invokes another form that can also be opened from a variety of contexts.

Example

Miscellaneous charges can contain multiple contexts that invoke accounting distributions, which can also contain multiple contexts:

PurchTable form > **MarkupTrans** form > **AccountingDistribution** form

Reference

```
MarkupTrans.enableAccountingDistributionButton
```

Solution

1. Create separate menu items for each of the source and destination contexts.
2. Based on the input context, set the menu item button name for the destination context:

   ```
   buttonDistributeAmount.menuItemName(menuitemDisplayStr(AccountingDistMarkupTransPO));
   ```

3. Define the security model so that the entry point access levels between the source and destination menu items are aligned.
4. Reduce the form data source access by only the table access right for a specific menu item:

```
FormSecurity::setFormDataSourceMaxAccessRight(this);
```

Pattern: Securable controls

A form that can be opened from a variety of different contexts has securable controls. The security system unions together all the access levels of the securable controls from all contexts. By default, the user is granted the maximum access by the union.

Problem

Relying on the **NeededPermission** property of the controls can result in over-permissioning.

Solution

Use the following method to enable the controls according to feature requirements:

```
FormSecurity::getMenuItemAccessRight
```

9. Best practices

Best practices for avoiding under-permissioning and over-permissioning

The easiest way to unblock a security barrier might be to grant full access. But that solution likely causes other unintended over-permissioning issues. The developer should balance the under-permissioning and over-permissioning scenarios.

1. Grant the minimum access necessary for the scenario.
2. Solve problems at the entry point level and privilege level, so that administrators can easily use them as building blocks to configure new duties and roles.
3. Solve problems with metadata, not in X++ code. In X++ code, observe the following guidelines:
   - Do not elevate code access.
   - Do not refer to specific roles, duties, privileges, or entry points.
4. Prefer granting access to SEPs rather than to TPF tables.
5. Consider the ongoing maintenance cost of your solution.
6. Consider how the functionality is exposed to the security model.
   - The AOT Security add-in explains these relationships.

     To access the add-in, right-click an AOT element, and then click **Add-Ins** > **Security tools**.
   - To reduce complexity, have separate roles for read functionality and write functionality. Craft separate duties that provide the minimally necessary privileges to support read or write.

Best practice checks for security

The security best practices are integrated into the standard best practice checks.

Examples

1. A privilege must have at least one entry point.
2. Every privilege must be contained in at least one duty.
3. Every duty must be contained in at least one role.
4. Every duty must be contained in one process cycle.

   Note: A process cycle is a different way to group duties. Process cycles to let administrators quickly find a duty when they are adjusting role definitions.

Test a security privilege

This section describes how to test a security privilege.

1. Create a role by using the **AOT** > **Security** > **Roles** node.
2. In the AOT, assign the appropriate duty or privilege to the new role.
3. Create a test user account (such as axtest3).

   System administration > **Common** > **Users**
4. Assign the user to a role.

   System administration > **Setup** > **Security** > **Assign users to roles**

   Always include the System user in the role.

5. Start the application with a command line or shortcut that is similar to the following (lines are wrapped here only for display):

```
%windir%\system32\cmd.exe  /c runas  /savecred  /user:mywindowsdomain\axtest3
"C:\Program Files (x86)\Microsoft Dynamics AX\6.0\Client\Bin\Ax32.exe"
```

Debugging

Unlike in some earlier versions of Microsoft Dynamics AX, in Microsoft Dynamics AX 2012 you can now debug with reduced permissions. Here are the steps:

1. Start the application, and add the role that you want to test with your administrator level user ID.

   System administration > **Setup** > **Security** > **Assign users to roles**

2. Open one Microsoft Dynamics AX developer workspace in administrator mode.
3. Close all application workspace windows.

   Now you are left with only one Microsoft Dynamics AX window, a developer workspace.

4. Set break points in the code that you want to debug.
5. Run the following AOT job to turn off administrator permissions:

```
static void Job2(Args _args)
{
    SecurityUtil::sysAdminMode(false);
}
```

6. Open an application workspace by using the button or menu from the development workspace (or by pressing Ctrl+W).
7. Debugger will start when a break point is encountered.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989
Worldwide +1-701-281-6500
www.microsoft.com/dynamics