

Microsoft Dynamics® AX 2012

Developing Extensible Data Security Policies

White Paper

The extensible data security framework is a powerful new feature in Microsoft Dynamics AX 2012 that can enable developers and administrators to provide comprehensive data security using rich policies. This whitepaper describes the steps involved in developing policies that have minimal performance overheads.

Date: January, 2011

Author: Arindam Chatterjee, Principal Program Manager Lead,
Microsoft Dynamics AX

Send suggestions and comments about this document to adocs@microsoft.com. Please include the title with your feedback.



Table of Contents

Introduction.....	3
Data security policy concepts.....	3
Constrained table.....	3
Primary table	3
Policy query	3
Context	3
Developing an extensible data security policy.....	3
Designing and modeling the policy query	4
Creating the policy	5
Adding constrained tables and views	6
Setting the policy context	8
Developing efficient extensible data security policies.....	9
Using extensible data security constructs to minimize performance overhead	9
Debugging extensible data security policies	11
Summary	13

Introduction

The extensible data security framework is a new feature in Microsoft Dynamics® AX 2012 that enables developers and administrators to secure data in shared tables such that users have access to only the part of the table that is allowed by the enforced policy. This feature can be used in conjunction with role-based security (also supported in Microsoft Dynamics AX 2012) to provide more comprehensive security than was possible in the past.

Extensible data security is an evolution of the record-level security (RLS) that was available in earlier versions of Microsoft Dynamics AX. Extensible data security policies, when deployed, are enforced, regardless of whether data is being accessed through the Microsoft Dynamics AX rich client forms, Enterprise Portal webpages, SSRS reports, or .NET Services.

This white paper outlines the steps that developers have to take to create new extensible data security policies. It highlights strategies for minimizing the performance impact of these policies as well as policy debugging techniques.

Data security policy concepts

When developing a data security policy, you need to become familiar with several concepts, such as constrained tables, primary tables, policy queries, and context. This section outlines these concepts. Subsequent sections will use these concepts to illustrate how they work together to provide a rich policy framework.

Constrained table

A *constrained table* is the table or tables in a given security policy from which data is filtered or secured, based on the associated policy query. For example, in a policy that secures all sales orders based on the customer group, the SalesOrder table would be the constrained table. Constrained tables are always explicitly related to the primary table in the policy.

Primary table

A *primary table* is used to secure the content of the related constrained table. For example, in a policy that secures all sales orders based on the customer group, the Customer table would be the primary table.

Policy query

A *policy query* is used to secure the constrained tables specified in a given extensible data security policy. This query will return data from a primary table that is then used to secure the contents of the constrained table.

Context

A policy context is a piece of information that controls the circumstances under which a given policy is considered to be applicable. If this context is not set, then the policy, even if enabled, is not enforced.

Contexts can be of two types: role contexts, and application contexts. A *role context* enables policy application based on the role or roles to which the user has been assigned. An *application context* enables policy application based on information set by the application.

Developing an extensible data security policy

Developing an extensible data security policy involves the following steps:

1. Modeling the query on the primary table.

2. Creating the policy.
3. Adding the constrained tables and views.
4. Setting the context.

However, before you start developing a policy, you must understand the underlying requirements. You should identify the set of constrained tables and analyze the relationships that these tables have with the primary table. You should also analyze the data access patterns of the constrained tables, the table sizes, and existing indexes on both the primary and constrained tables. All of these aspects influence the impact that a policy will have on the performance of the application.

Designing and modeling the policy query

The policy query and the table or tables being constrained are the two most important aspects of an extensible data security policy. The policy query will be added to the WHERE clause (or ON clause) on all SELECT, UPDATE, DELETE and INSERT operations involving the specified constrained tables. Unless carefully designed and tested, policy queries can have a significant performance impact. Therefore, you should follow certain simple but important guidelines when developing an extensible data security policy ([see Developing efficient extensible data security policies](#) later in this article for details).

As an example, we will walk through the development of a policy that will ultimately constrain records in several tables, such as AssetTable and BankChequeTable, so that users who are vendors can only retrieve their own records.

To design and develop the policy query for this policy:

1. Identify the primary table. In this case, the primary table is VendTable.
2. Create a modeled query under the AOT **Queries** node:
 - Use VendTable as the first data source.
 - Add other data sources as required by the Vendor data model.

Figure 1 shows the query.

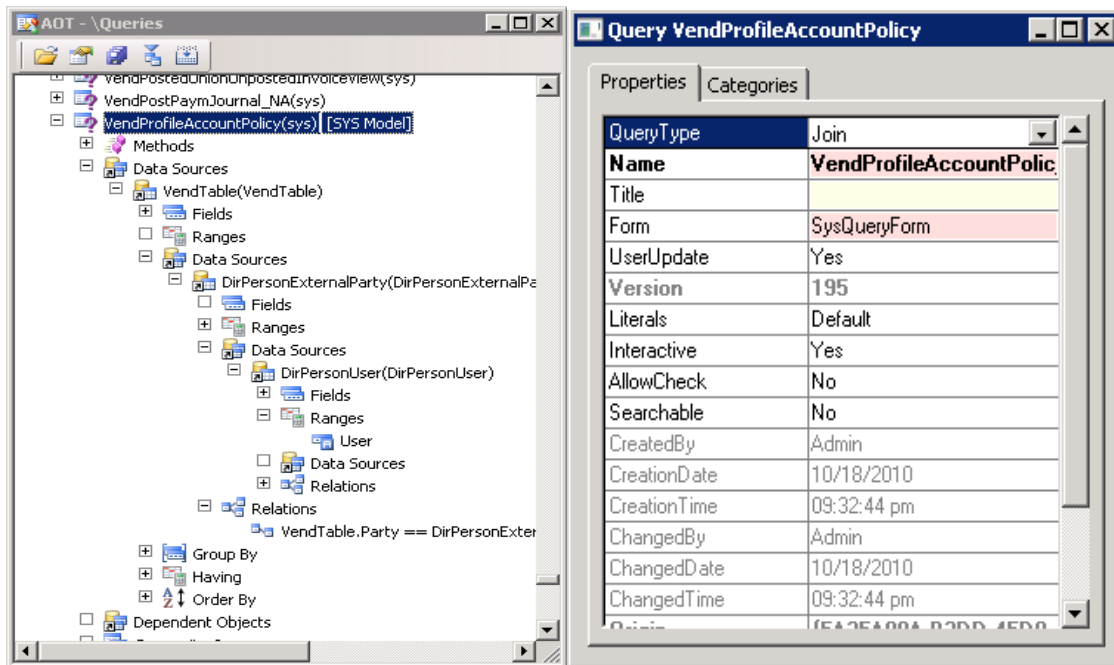


Figure 1: Policy query on primary table

Creating the policy

Now that the policy query has been created, the next step is to create the policy itself under the **AOT Security > Policies** node.

1. Right-click the **AOT Security > Policies** node, and select **New Security Policy**.
2. Set the **PrimaryTable** property on the policy to **VendTable**.
3. Set the **Query** property on the policy to **VendProfileAccountPolicy**.
4. Set the **PolicyGroup** property to **Vendor Self Service**.
When set, this property can be used by administrators and developers to quickly identify groups of related policies. There is no run-time usage of this property.
5. If you want the primary table to be secured using this policy, set the **ConstrainedTable** property to **Yes**.
6. Set the **Enabled** property to **Yes** or **No**. This property can be used to control whether the policy will be enforced by the extensible data security runtime.
7. Set the **ContextType** property to one of the following:
 - **ContextString** – Set the property to this value if a global context is to be used to determine whether the policy should be applied. When required, this context string needs to be set by the application using the `XDS::SetContext` API.
 - **RoleName** – Set the property to this value if the policy should be applied only if a user in a specific role accesses the constrained tables.
 - **RoleProperty** – Set the property to this value if the policy is to be applied only if the user is a member of any one of a set of roles that have the **ContextString** property set to the same value (see [Setting the policy context](#) later in this article for details for more details).

The example policy created can be seen in Figure 2. Note that the **ContextType** property is currently set to the value **ContextString**, but the **ContextString** property is empty. This combination implies that when enabled, this policy will always be applicable.



Figure 2: Policy properties

Adding constrained tables and views

Now that the policy query is in place and the policy has been created, the next step is to add the constrained tables and views that contain the data that will be secured by this policy.

To add constrained tables or views:

1. Right-click the **Constrained Tables** node.
2. Click **New > Add table** to add a constrained table, for example, the AssetBook table.
When adding a constrained table, you must also choose the relationship to be used to join the primary table with this constrained table.
3. Click **New > Add View** to add a constrained View to this policy.

Repeat these steps for every constrained table or view that needs to be secured through this policy.

Figure 3 shows what such a policy would look like in the AOT.

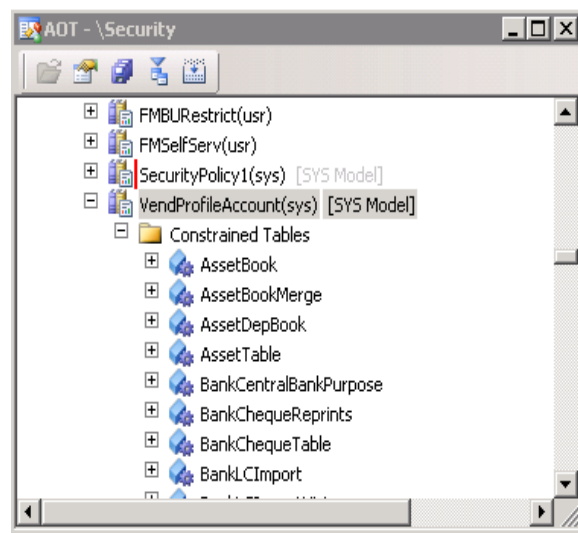


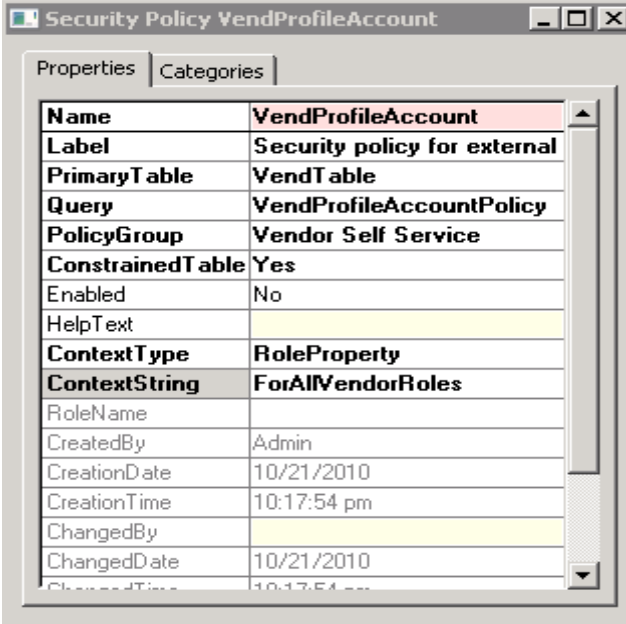
Figure 3: Final VendProfileAccount policy

At this point, the policy is ready for deployment. When the **Enabled** property is set to **Yes**, the policy will become fully operational for all users attempting to access the tables (or views) in the **Constrained Tables** node. Please note that security policies are not applied for system administrators (i.e. users in the SysAdmin role).

Setting the policy context

Based on the original requirements, the policy needs to be modified to apply only to users assigned to the vendor role. The following changes will have to be made to achieve that:

1. Change the **ContextType** property on the policy node to **RoleProperty**.
2. Set the **ContextString** property on the policy node to **ForAllVendorRoles** (see Figure 4).



Security Policy VendProfileAccount	
Properties	
Name	VendProfileAccount
Label	Security policy for external
PrimaryTable	VendTable
Query	VendProfileAccountPolicy
PolicyGroup	Vendor Self Service
ConstrainedTable	Yes
Enabled	No
HelpText	
ContextType	RoleProperty
ContextString	ForAllVendorRoles
RoleName	
CreatedBy	Admin
CreationDate	10/21/2010
CreationTime	10:17:54 pm
ChangedBy	
ChangedDate	10/21/2010
ChangedTime	10:17:54 pm

Figure 4: Policy with role context

Now, to associate this policy with all the vendor roles, the **ForAllVendorRoles** context must be applied to the appropriate roles.

1. Navigate to the AOT node corresponding to each role that needs to be associated with this policy, for example, the **VendVendor** role.

- Set the **ContextString** property on the **VendVendor** role node to **ForAllVendorRoles** (see Figure 5).

Note The **ContextString** property is not localized. Therefore, only culture-neutral strings should be used for this property.

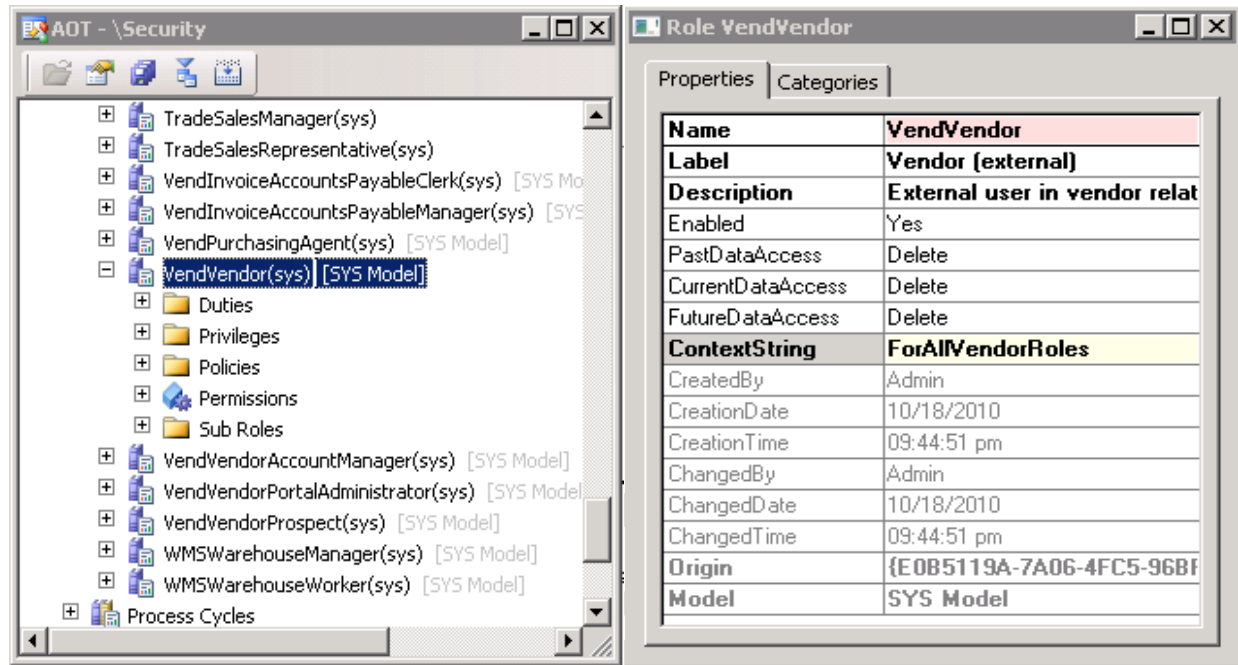


Figure 5: Setting context on the vendor roles

Developing efficient extensible data security policies

As mentioned before, applying extensible data security policies on any constrained table will affect the run-time performance of queries on that table. When developing policies, you should keep the following principles in mind:

- Follow standard best practices of developing efficient queries. For example, create indexes on join conditions.
- Reduce the number of joins in the query. Complex and normalized data models can lead to queries with a very large number of joins. Consider changing the data model or adopting patterns such as extensible data security constructs to reduce the number of joins at run time.

Using extensible data security constructs to minimize performance overhead

Rich policy requirements and complex underlying data models can often lead to policy queries that are not only intricate but that also involve joins across a large number of tables. Not surprisingly, extensible data security policies that include such queries can cause significant performance problems.

However, in many cases, a significant portion of the policy query retrieves static data such as legal entities for the logged-on user and departments to which they belong.

The extensible data security framework provides a way by which this static portion of the policy data can be retrieved just once for a given client session and then reused in subsequent policy applications.

Extensible data security constructs are temporary tables that are populated once for every client session. They exist in the AOT under the **Data Dictionary > Tables** node (see Figure 6).

Use the **XDS** table method to create a temporary table as an extensible data security construct. This method is available for developers to write X++ logic to populate the temporary table. Invoke the method the first time that a policy query with the construct as a data source is used. After the temporary table is populated, subsequent policy queries will use the temporary table.

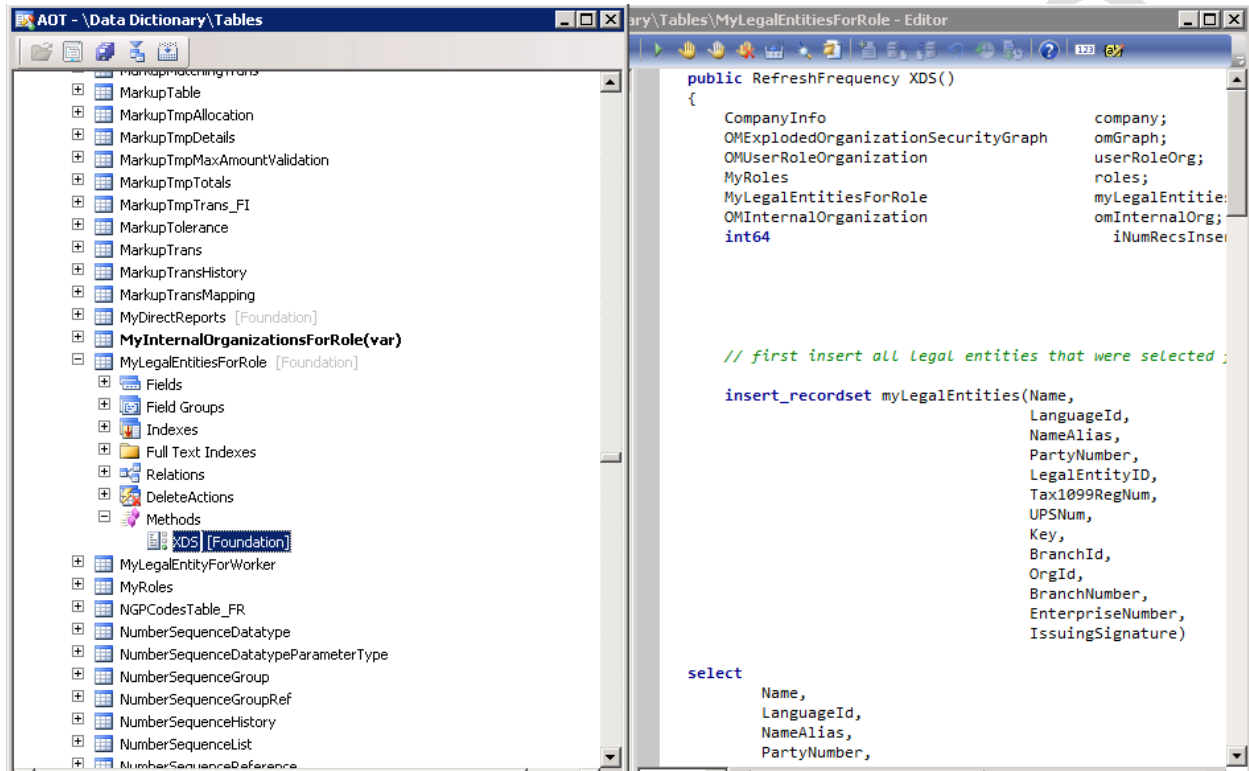


Figure 6: MyLegalEntitiesForRole construct

Figure 7 shows an example of a policy query that leverages the **MyLegalEntitiesForRole** construct.

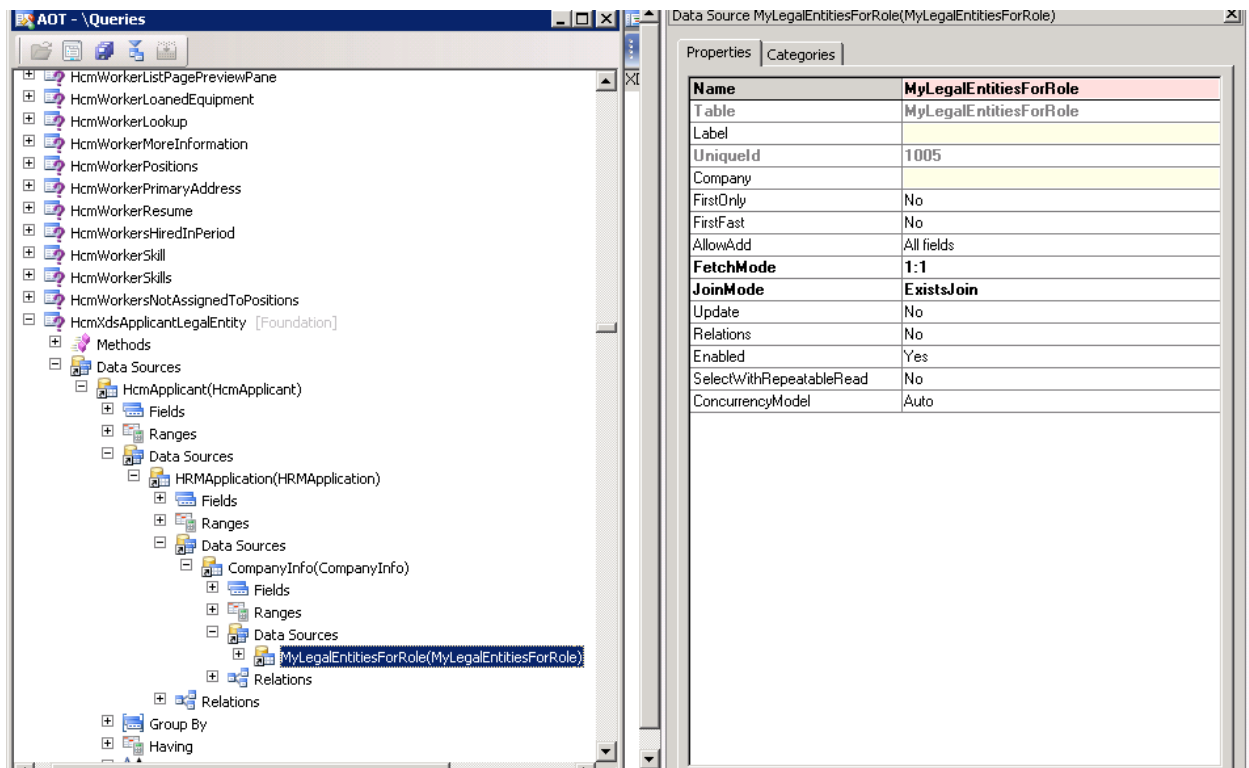


Figure 7: Policy query using MyLegalEntitiesForRoles

The **HcmXdsApplicantLegalEntity** query involves joins across four data sources. The fourth data source is the **MyLegalEntitiesforRole** construct, which encapsulates several joins. If an extensible data security construct were not used, this query would have involved joins across four more tables on every policy application—a significant performance overhead.

In this scenario, leveraging extensible data security constructs converted a policy query with seven or more joins into a policy query with four joins—a significant performance gain.

Debugging extensible data security policies

One of the common issues reported when a new extensible data security policy is deployed is that an unexpected number of rows are being returned from a given constrained table. For example, more sales orders are being returned than would be expected if the sales order table were being constrained by a given customer group.

The first thing to do in such scenarios is to review the SQL query being generated. In more recent builds of Microsoft Dynamics AX 2012, the extensible data security framework provides an easy debugging method. The X++ *select* has been extended with the *generateonly* command that instructs the underlying data access framework to generate the SQL query without actually executing it. The generated query can be retrieved using simple method calls.

The following job runs a select query on the SalesTable with a *generateonly* command. It then calls the *getSQLStatement()* on the SalesTable and dumps the output using the info API.

```
static void VerifySalesQuery(Args _args)
{
    SalesTable      salesTable;
    XDSServices     xdsServices = new XDSServices();

    xdsServices.setXDSContext(1, '');

    //Only generate SQL statement for custGroup table
    select generateonly forceLiterals CustAccount, DeliveryDate from salesTable;

    //Print SQL statement to infolog
    info(salesTable.getSQLStatement());

    xdsServices.setXDSContext(2, '');
}
```

The extensible data security policy development framework further eases this process of doing some advanced debugging by storing the query in a human-readable form. This query and others on a given constrained table in a policy can be retrieved by using the following Transact-SQL query on the database in the development environment (AXBDEV in this example):

```
SELECT [PRIMARYTABLEAOTNAME], [QUERYOBJECTAOTNAME],
       [CONSTRAINEDTABLE], [MODELEDQUERYDEBUGINFO],
       [CONTEXTTYPE], [CONTEXTSTRING],
       [ISENABLED], [ISMODELED]
FROM [AXBDEV].[dbo].[ModelSecPolRuntimeEx]
```

Figure 8 shows the query results.

	QUERYOBJECTAOTNAME	CONSTRAINEDTABLE	MODELEDQUERYDEBUGINFO
1	VendProfileAccountPolicy	AssetBook	SELECT * FROM AssetBook(AssetBook_1) EXISTS JOIN 'x' FROM VendTable(VendTabl...
2	VendProfileAccountPolicy	AssetBookMerge	SELECT * FROM AssetBookMerge(AssetBookMerge_1) EXISTS JOIN 'x' FROM VendTa...
3	VendProfileAccountPolicy	AssetDepBook	SELECT * FROM AssetDepBook(AssetDepBook_1) EXISTS JOIN 'x' FROM VendTable(V...
4	VendProfileAccountPolicy	Asset Table	SELECT * FROM Asset Table(Asset Table_1) EXISTS JOIN 'x' FROM VendTable(VendTa...
5	VendProfileAccountPolicy	BankCentralBankPurpose	SELECT * FROM BankCentralBankPurpose(BankCentralBankPurpose_1) EXISTS JOIN '...
6	VendProfileAccountPolicy	BankChequeReprints	SELECT * FROM BankChequeReprints(BankChequeReprints_1) EXISTS JOIN 'x' FROM ...
7	VendProfileAccountPolicy	BankChequeTable	SELECT * FROM BankChequeTable(BankChequeTable_1) EXISTS JOIN 'x' FROM Vend...

Figure 8: Query output

As you can see in Figure 8, the query that will be appended to the WHERE clause of any query to the AssetBook table is available for debugging. Other metadata, such as LayerId, is also available if needed.

The first debugging technique is recommended for most scenarios.

Summary

The extensible data security framework is a powerful new feature in Microsoft Dynamics AX 2012 that has been made available to developers and customers to address rich data security policy scenarios. This white paper introduced the concepts related to extensible data security and provided a step-by-step guide to developing an extensible data security policy. As with any powerful feature, developers must be aware of the issues that can crop up. This whitepaper covered some tips and tricks and framework-provided constructs that help alleviate performance issues and ease any debugging that might be required.

Pre-released

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

release

This document supports a preliminary release of a software product that may be changed substantially prior to final commercial release. This document is provided for informational purposes only and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, the Microsoft Dynamics Logo, and Microsoft Dynamics are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Microsoft