Microsoft Dynamics® AX 2012

Using the Policy Framework in Microsoft Dynamics AX 2012

White Paper

This document focuses on the end-to-end experience of developers who are adding a new policy type or extending existing policy types (by adding new rule types) in Microsoft Dynamics AX 2012. Topics covered include creating and updating required Application Object Tree (AOT) objects and interfacing with the data model and the APIs provided. This paper references real-life examples that use these concepts and describes the business logic and implementation options available in the policy framework.

Date: April 2011

http://microsoft.com/dynamics/ax

Author: Rob Drollinger, Software Development Engineer

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

Table of Contents

Introduction

Microsoft Dynamics® AX 2012 introduces a policy framework that allows users to define policy rules in modules to help guide the flow of business. The framework provides a common pattern and rich feature set that users can leverage to enforce simple rules or to evaluate and act on several different rules in a complex organizational hierarchy. The actual rules used by businesses, and the actions that these rules allow, prevent, or alter are outside the scope of the framework, however this white paper does provide examples of the policy types and rule types that ship with Microsoft Dynamics AX 2012.

Note The policy framework is only used for defining and enforcing business rules; it is not related to the framework that creates extensible data security policies also being introduced in Microsoft Dynamics AX 2012.

There are three perspectives from which the policy framework can be viewed and used, depending on the role of the user:

- A developer introduces the policy types and rule types to be evaluated.
- An application developer writes the logic that retrieves date-effective rules for a given user. This is done in order to enforce these rules as the module requires (for example, in form logic to allow or prevent certain actions).

  Note One person often performs the role of developer and application developer.
- An administrator, in a manager role, defines specific rule data that will be enforced.

This white paper first introduces the policy framework core concepts that both developers and administrators need to understand. It then discusses more specific uses of the policy framework by each group, focusing primarily on the developer perspective. Finally, the document describes some of the advanced features of the policy framework, which can be used to provide businesses with even more flexibility.

Core concepts

This section describes the concepts that are common to both developers and administrators who use the policy framework.

Overview of policy framework use

Developers and administrators can use the policy framework to enhance their business processes by enforcing rules for Microsoft Dynamics AX 2012 users. Examples of rules include:

- Expense reports submitted by employees for items above a certain currency amount are disallowed or result in an appropriate warning.
- Employees in one organization are able to view a specific procurement catalog while others (for example, those in a different country or region with different suppliers) are required to view a different catalog.
- Employees with a particular title are allowed to approve lines only up to a certain amount of currency.
- Documents must be validated against certain criteria, and violations of those criteria must be flagged.

For each of these cases, a developer specifies the type of data that is stored in the policy and how various forms and classes behave with that data. An administrator then defines and manages who the policies will affect (and when) and specifies the exact data values for each rule.

Framework components

The policy framework consists of three main components that provide the functionality corresponding to the developer and administrator roles described earlier.

- A unified, extensible schema that provides the developer with a basic platform on which to build a policy solution.
- An API that provides the application developer with the means to retrieve policy rules. Policy rules take the application developer directly to the data needed to execute the appropriate business logic.
- A common entry point for both the policy administrator and the developer, who can use it to manage policies. Very little customization is required beyond entry of their business data.

These components provide the flexibility that developers and administrators must have to achieve their objectives without difficulty. They also provide the user with the same look and feel as the rest of the policy UI in Microsoft Dynamics AX 2012.

Policy types

A policy type is a collection of all the rule types that are applicable to a given module or feature area.

A user in a manager role can be granted permission to define policies of a given type through a menu item that opens a list page. The list page displays all the policies of a particular policy type.

Microsoft Dynamics AX 2012 ships with six policy types. These types are specified with the following **SysPolicyTypeEnum** enumeration values:

- **ApprovalPolicy**
- **AuditPolicy**
- **PurchasingPolicy**
- **TrvExpensePolicy**
- **TrvRequisitionPolicy**
- **VendInvoicesPolicy**

Each policy type is associated with a "hierarchy purpose" (specified in the HierarchyPurpose field in the SysPolicyType table) that defines the organizational hierarchies that can be assigned to these policies. For example, a developer can only assign organizations in the hierarchies that are designated for "Procurement internal control" to **PurchasingPolicy** policies.

Note The same organizational hierarchy can be assigned to multiple hierarchy purposes.

Rule types

Each policy type encompasses an entire feature area; its associated rule types categorize the rules that can be set up for specific features within that area. For example, *airline*, *hotel*, and *meals* are examples of expense policy rule types. These rule types correspond to specific categories of expenses on which different limits can be set. Each policy type can have any number of related rule types. A policy administrator can maintain rules of all the rule types that are associated with the policy type for which the administrator is responsible.

Unlike policy types, which are static and well known, rule types do not have to be predefined (although they generally are). For example, the **AuditPolicy** and **VendorInvoicesPolicy** policy types allow the user to define the rule types that are to be enforced. Rule types for these two policy types do not ship with Microsoft Dynamics AX 2012. All other policy types have a predefined set of rule types that do ship with Microsoft Dynamics AX 2012.

Each rule type is associated with a corresponding form, which contains data that is specific to that rule type, and which is used when creating or editing a rule. At a minimum, the form contains the effective dates of the rule. Each rule type will also generally have an underlying table in the data model. This will be covered in more detail in the SysPolicyRule table section later in this white paper.

Rules

Rules are at the core of any policy. The essential data that defines how policies will be enacted resides in rules. After a developer has defined policy types and rule types and has created a policy, the policy administrator can add rules to that policy. For example, suppose an administrator adds a catalog rule to a purchasing policy. The catalog rule is configured to specify the "Adventure Catalog" for internal procurement. After the purchasing policy is associated with a particular organization, all employees of that organization will see the Adventure Catalog when they browse the internal procurement site.

Rules are date-effective, which gives you a set of powerful options for manipulating policy events over time. You can define rules that take effect in the present and expire at some point in the future, or that never expire. You can also set up rules that will take effect at some specified point in the future. By defining date-effective rules, you prevent multiple rules from being active at any given time, when they might provide conflicting data.

After a rule has gone into effect, it can only expire or be retired; it can never be removed from the policy. This constraint provides enhanced auditing capability and historical policy retention. The policy framework can be queried for the rules that were in effect at a specific time, if needed.

As mentioned earlier, each rule is associated with a corresponding form. The developer implements the form, which displays and stores data in the data model. The administrator can then interact with this form to define and update the rule as needed.

Organization assignment

Before a policy can affect anyone, it must be associated with an organization. The administrator configures the association with the organization on the **SysPolicy** form after a policy has been created. In the business logic, the application developer queries the API for rules that are in effect for a specific organization.

All hierarchies with a specific hierarchy purpose can be sources from which to select organizations. An additional source from which to select organizations is the "flat list" of legal entities in the CompanyInfo table, which are designated in the policy framework as "Companies." (An example would be a legal entity with a *DataAreaId* of "DAT.") Although not a true hierarchy, this can be thought of as a default hierarchy type that is always available.

You can assign only organizations from a single hierarchy type to a given policy (this is the typical scenario). If organizations from multiple hierarchy types require the same policy, you must create multiple policies. For example, if you need to create an **AuditPolicy** policy for employees of organizations in a procurement hierarchy and for employees of companies in the "flat list" in Microsoft Dynamics AX 2012, you will have to create a separate policy for each group.

Rules evaluation

Multiple rules can affect users in an organization, and these rules might be returned by the policy API in a specific order. Evaluating rules will be described in detail in the Multiple rules evaluation section of this white paper.

Developing policies

This section follows the development cycle in the policy framework, from creating policy and rule types to calling the APIs that are used to retrieve date-effective rules.

Data model

Before you start developing policies, you should understand the underlying data model. There are only three core tables with which you need to be familiar:

- The SysPolicyRule table
- The SysPolicyType table
- The SysPolicyRuleType table

Note The Entity-Relationship (ER) diagram provided at the end this white paper shows the key relationships among all the tables in the data model. None of the related tables will be directly queried, but the diagram provides a good basis for understanding the entity relations involved in the policy framework.

SysPolicyRule table

The SysPolicyRule table is the main table with which you will interact. To demonstrate the design options that this table allows, we will start with the output and work backwards. When you query the policy API (using the **SysPolicies::getPolicyRuleIds** method) for a list of rules for a given policy type, rule type, and organization, the API returns an array of RecIds to the SysPolicyRule table, which holds a list of specific rule instances. This strategy allows you to follow one of two general patterns (listed below) for storing and accessing your rule data. Choose the pattern that best suits your requirements:

- **One set of data for each rule**

  This is the more common pattern. There is just one set of data for each rule, which is stored in one row in a table. To implement this pattern, create a table that is a subtype of the SysPolicyRule table and add all the fields required for your feature. A RecId that you receive from the policy API will now directly correspond to a date-effective row in your table (which is managed by the policy framework). You can directly query your table without needing to know that it extends the SysPolicyRule table. Examples of this pattern can be seen in the CatCatalogPolicyRule and TrvPolicyRule tables. For details, see the ER diagram.

- **Many entities for each rule**

  Use this pattern when you need to store several rows of information for each rule (that is, the rules are displayed as a grid on the rule form). In this case, create a table with the fields you need that is not a subtype of SysPolicyRule table. Instead, add a foreign key to the SysPolicyRule table to establish a many-to-one relationship. By doing this, you can reference many rows of data in your table with one SysPolicyRule RecId. Examples of this pattern can be seen in the ProcCategoryPolicyParameter and ProcCategoryAccessPolicyParameter tables. For details, see the ER diagram.

SysPolicyType and SysPolicyRuleType tables

The SysPolicyType and SysPolicyRuleType tables store the core type information that defines your policy and its rules. This information is static; therefore, after they have been configured, the rows in these tables should not need to change, and no user interface (UI) is provided for doing so.

Creating a policy type

To create a new policy type, complete the following steps:

1. Add a new value for the policy type to the **SysPolicyTypeEnum** base enumeration.
2. Create a row for the new policy type in the SysPolicyType table.

You can add a row to the SysPolicyType table in one of two ways:

- Update the **SysPolicySetup** class. Because this class is an implementation of the **SysSetup** class, its **loadData** method (which fills the policy and rule type tables) is called on installation

or upgrade of Microsoft Dynamics AX 2012, on activation of the configuration keys, and on full synchronizations of all the tables. You can also write a job to manually invoke the **SysPolicySetup.loadData** method.

- Write a simple job to insert the row.

The only two required fields in the SysPolicyType table are PolicyType and HierarchyPurpose. The HierarchyPurpose field was discussed earlier in the Policy types section. The PolicyType field must match the value that you add to the **SysPolicyTypeEnum** enumeration. The remaining fields are optional settings that will be covered in detail in the Policy type options section of this white paper.

Entry point for the policy type

To create a basic working path for your new policy type, you need to add a menu item that points to the **SysPolicyListPage** form. You must set three properties on this menu item to filter to the new policy type.

Property	Value
EnumTypeParameter	The literal string value, **SysPolicyTypeEnum**.
EnumParameter	The name of the new policy type (for example, **TrvRequisitionPolicy**) that has been added to the **SysPolicyTypeEnum** enumeration.
Parameters	An integer that represents the enumeration value of the new policy type (for example, 3).

Adding a new menu item also requires that you add the menu item to a menu and to a privilege. Knowledge of how to do this is assumed and is not discussed in this white paper.

Creating rule types

The process for creating new rule types is similar to the process for creating a new policy type. You need to perform the same two basic steps: add rows in the table, and add new enumerations (if the rule types are known).

This white paper assumes that your rule types are static and known at design time. If your requirements stipulate that the rule types must be dynamic (that is, defined by the user) and added at runtime, see the **SysPolicySourceDocumentRuleType** form in the AOT for an example of this pattern.

To add rule types to the SysPolicyRuleType table, edit the **SysPolicySetup** class as you did for the policy type. There are several instances of this procedure in the class code that you can follow as examples.

The required fields are as follows:

- **Name:** Should match the name of the enumeration that was added.
- **PolicyType:** A foreign key to the SysPolicyType table.
- **RuleFormName:** The AOT form name of the form that will be launched when configuring a rule of this type. For more information, see the Creating rule forms section later in this white paper.
- **IncludeParentRules:** A flag that enables the **IsInherited** flag on rule instances. This flag is usually set to "Yes" to allow rules set at a parent organization level to affect all organizations below it in the hierarchy.
- **IsPrecedenceSupported:** A flag that tells the framework whether the hierarchy precedence for a policy type can be overridden at a rule-type level.

Creating rule forms

To allow an administrator to fill your rule table with data, you must create a form that displays and saves that data. The name of the form or menu item pointing to that form is stored in the RuleFormName field of the SysPolicyRuleType table. The only data that you are required to show on this form are the effective date (which defaults to the current date and time) and the expiration date (which defaults to "Never"). These dates are stored in the ValidFrom and ValidTo fields in the SysPolicyRule table. The basic flow of this form is that, when it is opened, it shows either the values of the rule that is currently open or the default values to use when creating a new rule. When the user closes the form, the appropriate records should be saved to the SysPolicyRule table and to any other tables that you are using to store your rule data.

Several of these forms ship with Microsoft Dynamics AX 2012. Because forms are feature-specific, we cannot provide more precise guidance about how to design them. We recommend that you look at the forms listed in the RuleFormName column of the SysPolicyRuleType table for examples. The CatCatalogPolicyRule form, shown in Figure 1 is the simplest example because it contains just one piece of data for each rule. The following illustration of this form is provided for reference.

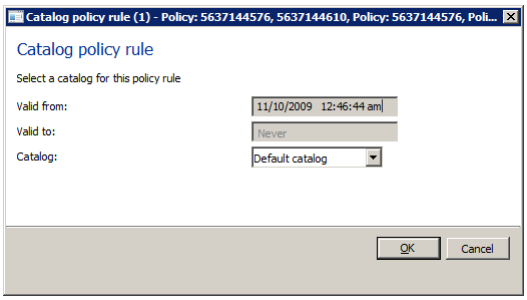


Figure 1: The CatCatalogPolicyRule form

Retrieving date-effective rules

When rules are defined and are in effect in the system, your application logic can query for these rules and take appropriate action. The **SysPolicies::getPolicyRuleIds** method is provided to facilitate this process. At a minimum, this method must be passed the policy type and the rule type that you set up earlier, and the organization that you would like to check for rules. The method will return references to the SysPolicyRule table for the rules that are active in the specified organization. From there, querying for the needed values in the table that you added (which can be either a subtype of, or contain a foreign key to, the SysPolicyRule table) will depend on your business logic. See the XML documentation for a complete, up-to-date description of the **getPolicyRuleIds** method.

After the developer has finished setting up the policies, a policy administrator can configure policies as required by their business needs.

Administering policies

This section briefly describes some of the general administrative flow common to all policy types. It is important to be familiar with this flow for full development and testing in the policy framework. A full discussion of the complete administrative experience is outside the scope of this white paper.

Managing policies

The main entry point for managing each policy type is the **SysPolicyListPage** form in the AOT, which is shown in Figure 2. In the application, there are entry points to this form for each policy type. For

example, purchasing policies use the navigation path **Procurement and Sourcing** > **Setup** > **Policies** > **Purchasing policies**.

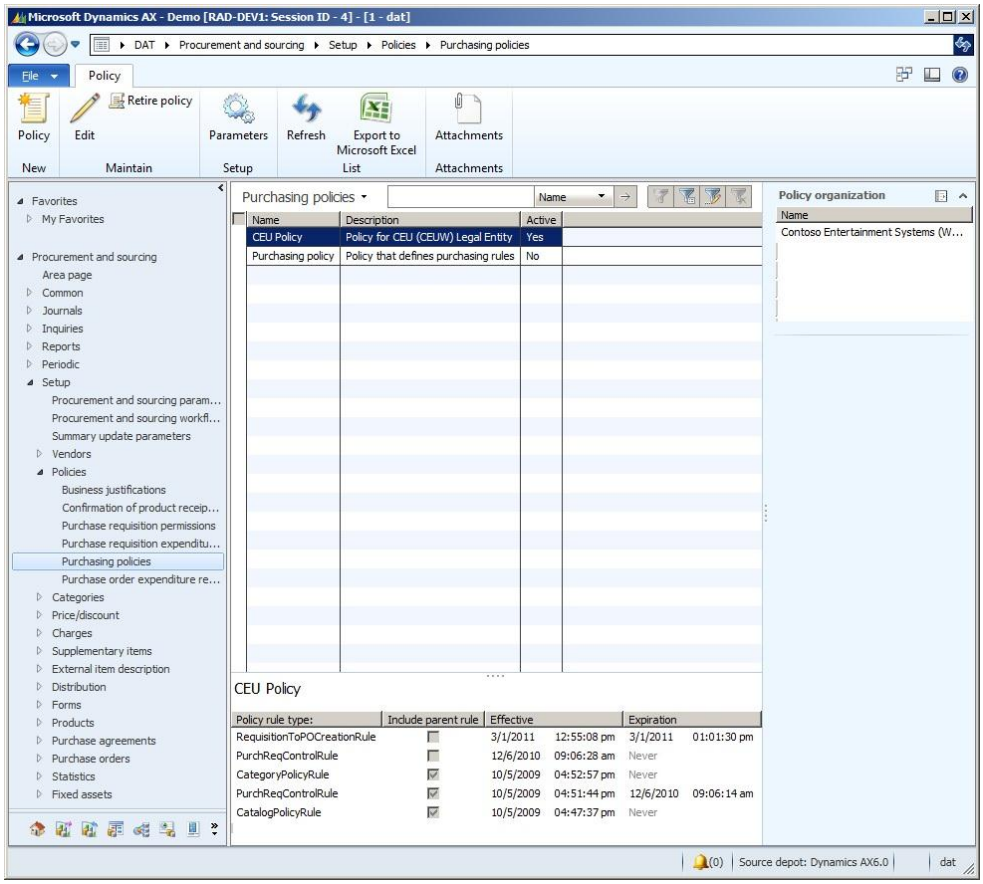


Figure 2: The SysPolicyListPage form

This list page displays all the policies of a single type. This type is determined by the menu item used to access the page. From this form, an administrator can create new policies, edit existing policies, retire (deactivate) policies, and quickly see information about the rules and organizations for each policy.

Note After a policy has been retired, it cannot be reactivated.

Optionally, the administrator can configure settings that are applicable to all policies of this type by using the additional parameters form. See the Policy type options section in this white paper for more details.

Configuring a policy

On the **Policy** list page, either click **Policy** in the **New** group or choose a policy from the list and then click **Edit** in the **Maintain** group. This opens the **SysPolicy** form, as shown in Figure 3.

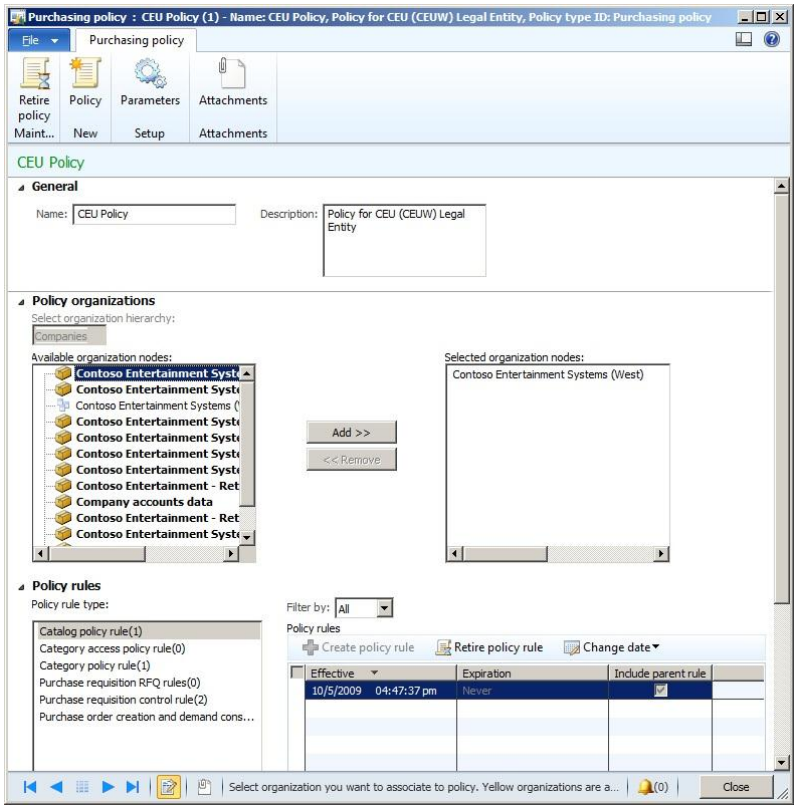


Figure 3: The SysPolicy form

This form contains three FastTabs:

- The **General** FastTab supplies a name and a description for the policy. These values can be changed at any time after the policy has been created.
- The **Policy organizations** FastTab shows the organizations that this policy will affect. As soon as an organization is assigned to the policy, any rules that exist are immediately effective in that organization.
- The **Policy rules** FastTab displays all policy rules associated with the policy (categorized by rule type) and provides three actions for managing the rules. Only one rule of each type can be in effect at any given time, but rules can be set up ahead of time to take effect on a future date. Double-clicking a rule row in the grid will open the rule form to allow the rule to be configured, which was discussed earlier in the Creating rule forms section.

Advanced features

This section describes some of the more advanced features that the policy framework provides. You may want to take advantage of some of these options, depending on your business requirements.

Policy type options

Optional fields in the SysPolicyType table can be enabled if they are required for your particular scenario. These fields are described in the following table.

Field	Purpose
AdditionalParamFormName	Specifies a form name or menu item that is opened when the **Additional Parameters** button is clicked. Provide a value in this field if your policies need additional configuration at the policy type level. The button will be dynamically added to the policy forms if this field is enabled. The **AuditPolicy** policy type makes use of this functionality.
IsDropDialog	When enabled, the **Additional Parameters** button will open the form specified by the AdditionalParamFormName field as a drop dialog. When the flag is disabled (the default value), the button will open the form in the normal pop-up style. When the AdditionalParamFormName field is not specified, this flag has no effect.
IsPolicyReadOnly	Enables or disables the creation or modification of all policies of the specified type. This flag should generally be disabled. Enable only if you want to prevent all policies from being changed.
IsReassignOrganizationSupported	When enabled, this flag allows the policy administrator to assign the same organization to multiple policies of this type. This flag therefore allows multiple rules of the same type to be in effect at the same time (through different policies). When the flag is disabled (the default value), the UI in the **SysPolicy** form prevents the administrator from assigning an organization to a policy if it is already assigned to another policy of that type.

Evaluation of multiple rules

When you query the policy framework, the policy API may find multiple rules for a given organization or set of organizations. Only your business requirements can determine how to interpret these results. The following two scenarios present typical evaluation strategies.

Scenario 1

Each of several rules defines some type of spending limit. Your business requirements state that the user should be allowed to spend up to the largest of these limits. In this case, you must examine each rule to determine which one specifies the largest limit. Similar scenarios might require finding the minimum rule or aggregating rules together in a different manner.

Scenario 2

Multiple rules exist for specifying which catalog a user should see. Because the functionality does not exist for the end user to switch catalogs, the developer is only interested in determining which rule has the highest precedence. In this case, the method **SysPolicies::getPolicyRuleId** can be used, which will always return only one rule.

Factors affecting rules

The number of rules and the specific order in which they are returned can be influenced by several factors:

- Whether multiple hierarchies are involved (for example, the organization exists in multiple hierarchies).

- Whether a user is querying for rules in more than one organization at the same time. (For example, if User A in Org1 performs an action on behalf of User B in Org2, the rules for either organization might apply.)

  In these two cases, the order in which the rules are returned depends on how the order of precedence among these hierarchies has been configured in the **SysPolicyParameter** form. Rules for organizational hierarchies that have a higher order of precedence will be returned first.

- Whether the **SysPolicyRuleType.IncludeParentRules** and **SysPolicyRule.IsInherited** flags are enabled.

  Enabling the **IncludeParentRules** flag on the rule type allows the use of the **IsInherited** flag on individual rules. (If **IncludeParentRules** is set to "No," the **IsInherited** flag must always be set to "No.") If the **IsInherited** flag is checked for a rule, the rule will be inherited by all descendent organizations down the tree.

  Setting the **IncludeParentRules** flag allows you to set rules for many organizations by explicitly assigning the policy to only a root organization.

  Not setting the **IncludeParentRules** flag allows you to explicitly set a rule for only a specific organization. The policy framework will start by looking for rules at the specified organization and continue traversing up the hierarchy to the root, adding the rules it finds until it finds a rule where **IsInherited** is set to "No."

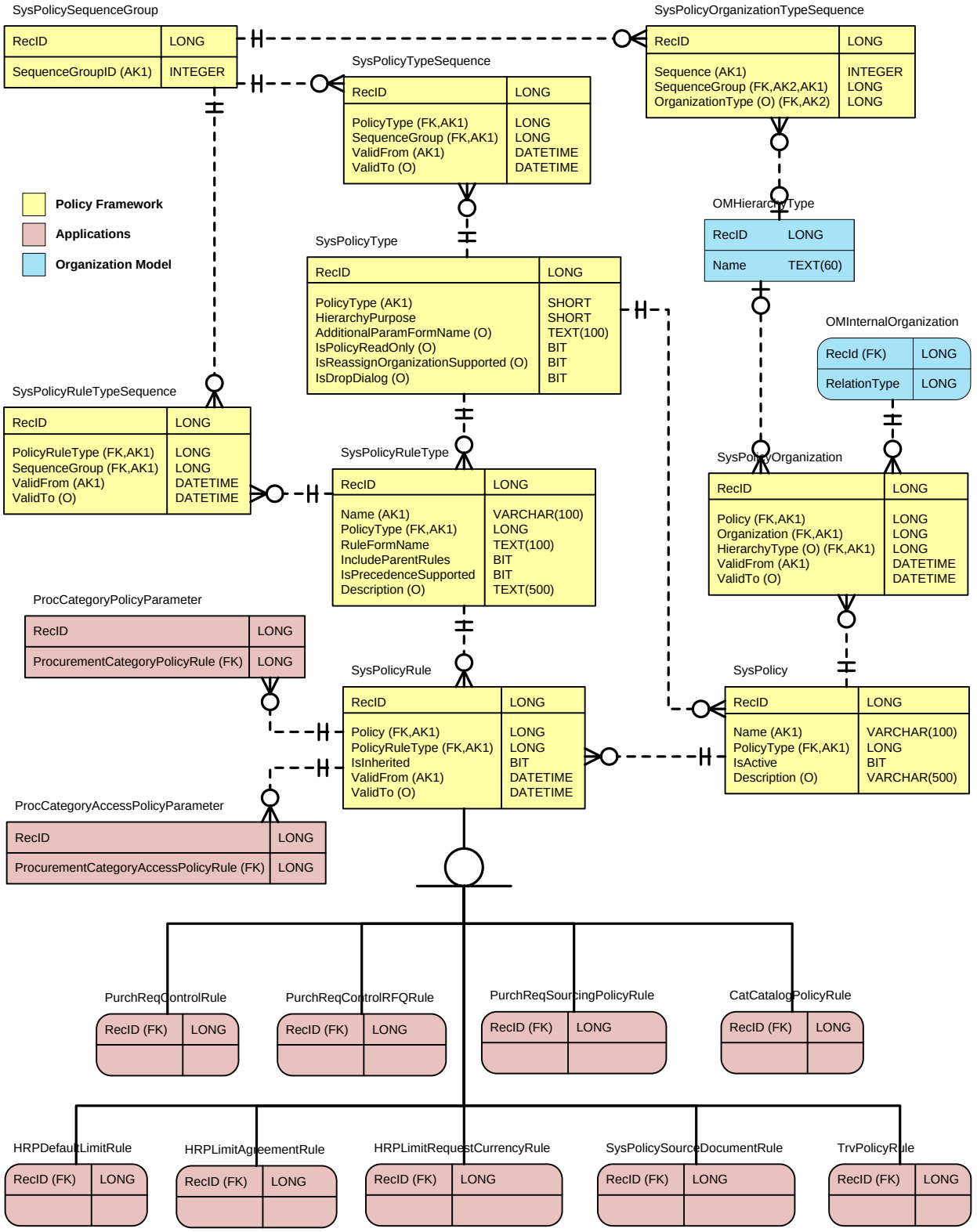
- Whether the **SysPolicyType.IsReassignOrganizationSupported** flag is enabled and multiple rules exist for the same organization. See the Policy type options section for details.

The policy framework does not provide a guaranteed order in which rules for the same organization will be returned. Therefore, it is the responsibility of the application code to take the correct composite view of all the rules that are found.

Summary

The new policy framework in Microsoft Dynamics AX 2012 provides rich possibilities for managing and enforcing an organization's business processes. This white paper has described the end-to-end development process for leveraging this powerful framework and has provided some insight into its integral features. This document should help developers, system designers, and policy administrators understand how the policy framework can best be adapted to their business needs.

Appendix: Entity-Relationship diagram



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