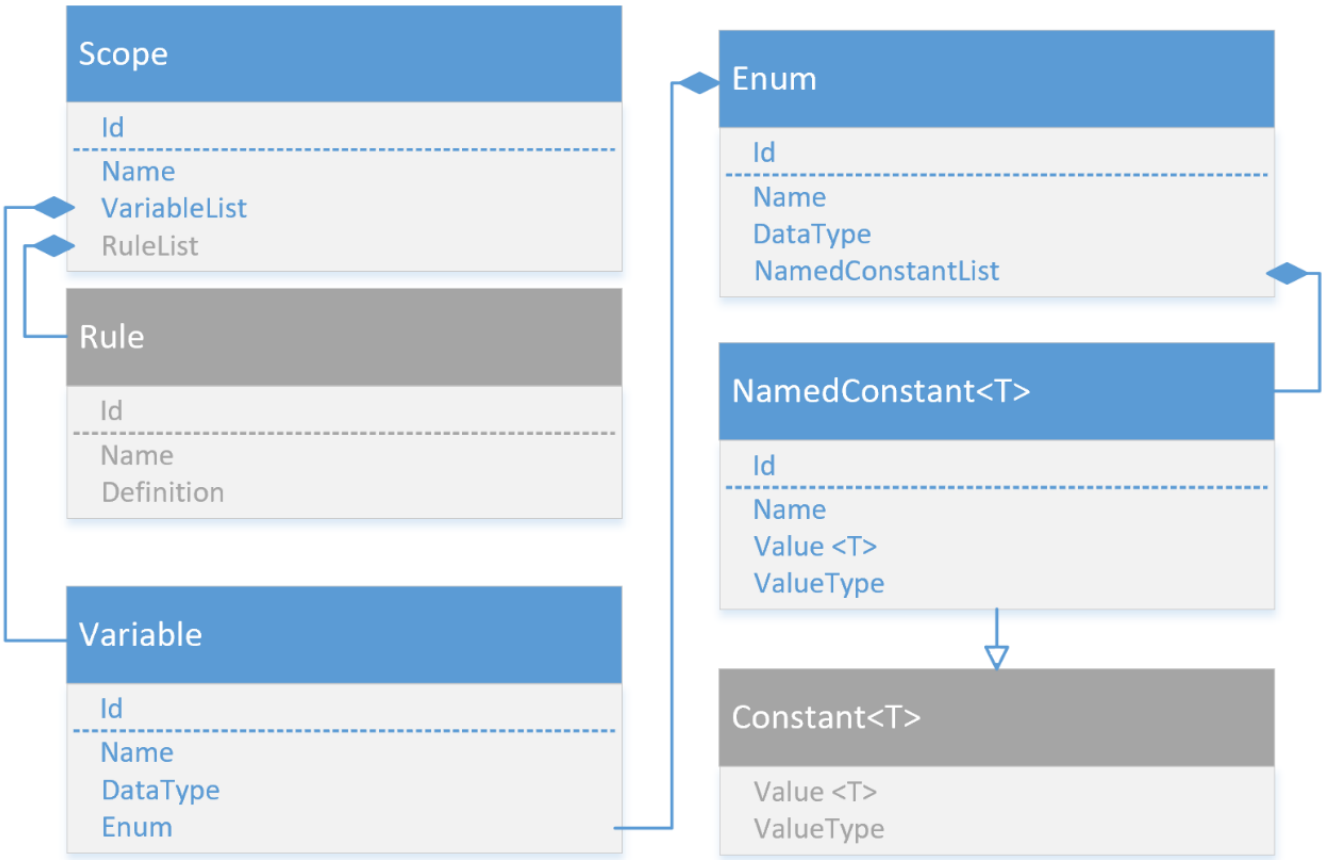
Scope Object Model

Scope Object Model Design

The Scope object contains a list of Variables. A Variable can be one of three data types (integer, date, string) or can have Enum with a list of values that can be assigned to the variable.

Figure 1 illustrates the structure of the Scope Object Model.



Important

Effectivity Services does not support Rules. It means that RuleList should remain empty in the Scope object and Rule objects should not be created and added to the RuleList.

©2026 Aras Corporation All Copyrights Reserved

Variable classes store variable definitions. Variable datatypes are integer, date, string or null. The Variable type will be null if the Variable uses Enum with a list of Named constants. If Enum is null, the Variable is handled by type.

Enum is the container for the list of Named constants. A Named constant is intended to store the value definition. It contains the property name, value, and valuetype.

Customizing Scope Builder

The Scope Builder method is a server method that is responsible for constructing a Scope Object. The purpose of the builder method is to construct a Scope Object using custom business data and business logic.

The Scope Builder uses a predefined method template. The template enables you to implement a Scope builder method properly.

Implementing the Scope Builder Method

The Scope Builder method uses the CSharp:Aras.Server.Core.Configurator template to define a class that inherits from the base abstract class. Here is the template for the Scope builder method:

Base abstract class:

```
public abstract class ScopeBuilderBase
{
    public Item ScopeItem { get; set; }
    public IServerConnection IomConnection { get; private set; }
    public void Init(Item scopeItem, IServerConnection iomConnection) {}
    public abstract Scope BuildScope();
    public abstract string[] GetGuidsItemDependsOn();
    public abstract List<string> GetItemTypeNamesItemDependsOn();
    public abstract ArrayList GetCustomKey();
}
```

The Scope Builder method is designed to provide a way for constructing a Scope object. The Scope Builder method also provides the built-in possibility to cache the Scope object.

Important

©2026 Aras Corporation All Copyrights Reserved

Implementing a Scope Builder method requires the override of all abstract methods - BuildScope, GetGuidsItemDependsOn, GetItemTypeNamesItemDependsOn, GetCustomKey.

The BuildScope method is responsible for constructing the Scope object instance.

The GetGuidsItemDependsOn, GetItemTypeNamesItemDependsOn, and GetCustomKey methods are used for Scope object caching and cache invalidation (see Caching section for more details).

Important

It is recommended to implement methods operating system agnostic for use with Linux and Windows. The primary OS incompatibility considerations for method implementation are path case-sensitivity, path separators, OS-specific line endings and OS-specific code. For more information about cross-platform development, please refer to section 2.3 Cross-platform development in Aras Innovator 40 - Programmer's Guide.

Builder Method Usage

The Effectivity Scope ItemType (effs_scope) is associated with the property builder_method. This property is a link to the Method Item. Effectivity Services automatically runs the linked builder method to get the Scope Object for the specified Effectivity Scope Item.

Caching

The Scope builder approach provides a built-in ability to cache a constructed Scope object. You can override the CustomKey function to store and retrieve the Scope object from the cache. This function returns the same ArrayList for each request for the same Scope object. To store different caches for different Scope objects, you can override the GetCustomKey function to return the different contents of the ArrayList:

```
public override ArrayList GetCustomKey()
{
return new ArrayList {
ScopeItem.getID(),
};
}
```

The Scope object can be invalidated automatically. Invalidation is triggered if at least one of the Items that the Scope object depends on changes. To do this, the system maintains a list of Item IDs and list

©2026 Aras Corporation All Copyrights Reserved

of ItemTypes associated with these Items. Any time an item associated with a maintained ItemType and ID is changed, the system invalidates the found item.

The following collections must be created to make caching and invalidation possible:

- **List of ItemType Names**. This list contains the name of each Item Type that is used to build the Scope.

```
public override List<string> GetItemTypeNamesItemDependsOn()
{
return new List<string> {
"ItemType1",
"ItemType2",
"ItemType3",
"ItemType4"
};
}
Each Item Type
```

  in this list must be a poly source item for the ScopeCacheDependency Poly Item ItemType.
- **List of Item IDs**. This list contains the ID of each Item that is used to build the Scope.

```
public override string[] GetGuidsItemDependsOn()
{
return new string[] {
"item_id_Z4",
"item_id_Z5",
"item_id_Z6",
"item_id_Z7"
};
}
```

Sample Builder Method

The effs_scope ItemType has a property named builder_method. This Item property is the link to a custom server method. The Builder method is a server method that is responsible for constructing a

©2026 Aras Corporation All Copyrights Reserved

Scope object from custom business data.

This sample shows how to implement a Builder method that creates a static Scope object.

```
//MethodTemplateName=CSharp:Aras.Server.Core.Configurator;
public override Scope BuildScope()
{
Scope builtScope = new Scope { Id = "item_id_scope", Name = "Model/SN Scope" };
Variable int_variable = new Variable(DataType.Int) { Id = "item_id_serialNumber", Name = "Serial Number"
builtScope.VariableList.Add(int_variable);

Aras.Server.Core.Configurator.Enum vEnum = new Aras.Server.Core.Configurator.Enum(DataType.String) { Id =
vEnum.AddNamedConstant("item_id_modelX", "Model X", "Model X");
vEnum.AddNamedConstant("item_id_modelY", "Model Y", "Model Y");
vEnum.AddNamedConstant("item_id_modelZ", "Model Z", "Model Z");
builtScope.VariableList.Add(new Variable(DataType.String) { Id = "item_id_model", Name = "Model", Enum =

return builtScope;
}

public override ArrayList GetCustomKey()
{
return new ArrayList { "static_scope" };
}

public override string[] GetGuidsItemDependsOn()
{
return new string[0];
}

public override List<string> GetItemTypeNamesItemDependsOn()
{
return new List<string> { };
}
```

In this sample we have a static Scope object, so we can store it in the cache using a hardcoded ID as a key. The method result is cached with key "static_scope". The Scope has two Variables:

- [Serial Number] – data type of integer
- [Model] - data type of string, with list of values:
  - [Model X]
  - [Model Y]
  - [Model Z]

Important

IDs in this sample code are hardcoded to show the meaning of the builder method. An implementation of a builder method retrieves business data and builds the Scope object using actual ids.GetCustomKey, GetGuidsItemDependsOn, and

©2026 Aras Corporation All Copyrights Reserved

GetItemTypeNamesItemDependsOn return either static or empty values for this sample static Scope Object. A Scope Object using real business data would return a feasible result.

It is recommended to implement methods operating system agnostic for use with Linux and Windows. The primary OS incompatibility considerations for method implementation are path case-sensitivity, path separators, OS-specific line endings and OS-specific code. For more information about cross-platform development, please refer to section **Cross-platform development** in **Aras Innovator 40 - Programmer's Guide**.

©2026 Aras Corporation All Copyrights Reserved