Scope Object Model

Configurator Services has an internal Scope object model that is customizable. Configurator Services processing logic is built around this object model.

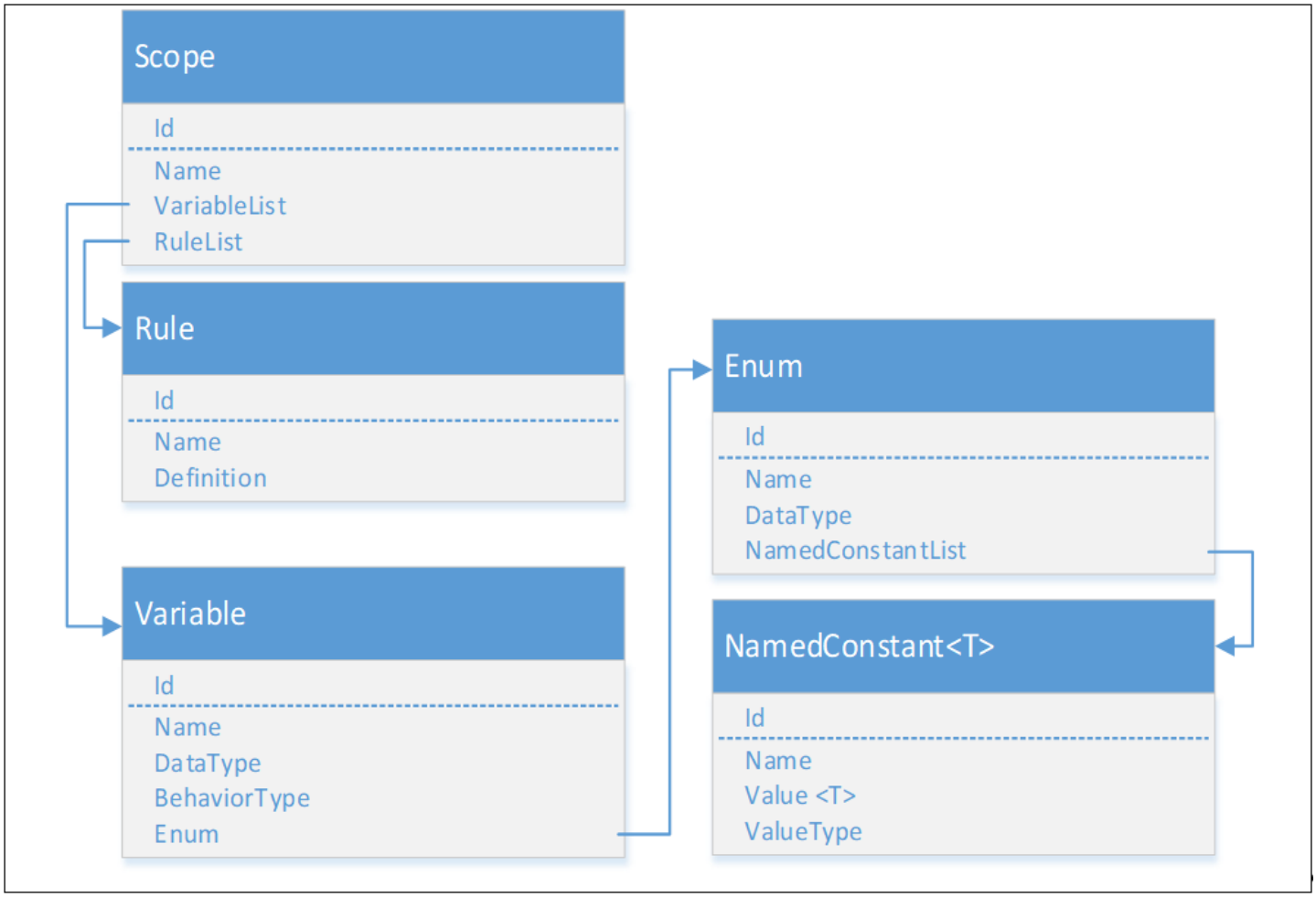
This section describes the following:

- Scope Object Model
- Configurator API Method Workflow
- Building the Scope Object Model

Scope Object Model Design

The Scope object contains Variables with a list of values that can be assigned to the variable and rules that define relationships between assigned variables.

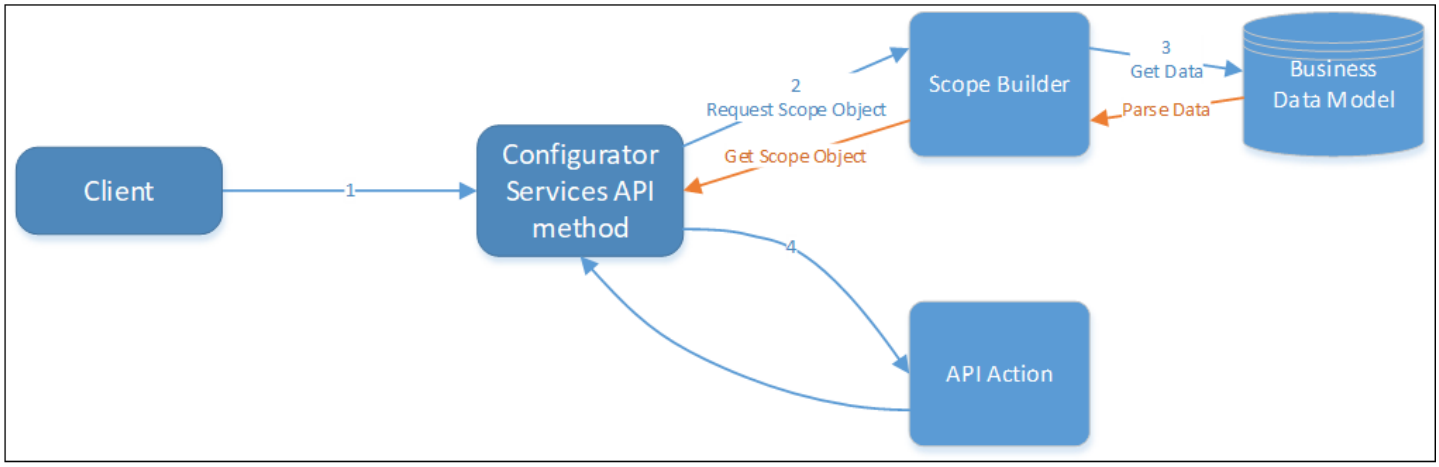
The following diagram illustrates the structure of the Scope Object Model:



- Scope is a class that contains two lists: a list of Variables and a list of Rules.
- Variable class stores the variable definition. BehaviorType defines the restriction of the number of values in a variable for solving the scope. The BehaviorType can be exactly-one, at-most-one or at-least-one. If BehaviorType is not specified, the default restriction is exactly-one.
- Named constant is intended to store the value definition. It contains the property name, value, and valuetype.
- Enum is the container for the list of Named constants.
- Rule is a container for an expression. The Rule has both a name and definition where the definition type is ExpressionBase.
- ExpressionBase is an object representation of a Boolean expression.

Configurator API Method Workflow

The following workflow diagram illustrates the process of collecting data from its source and using this data to perform an API Action.



1. The Client calls the Configurator Services API method.
2. The Configurator Services API method requests the scope builder method to build the Scope object.
3. The scope builder method collects and parses data from the business data source.
4. The Configurator Services API method performs an API action in the context of the built Scope object.

Customizing Scope Builder

Scope Builder is a server method that uses a predefined method template. The template enables customers to implement a Scope builder method. All the Configurator Services API methods that perform actions in the context of Scope should be parameterized with the Scope object. The following is an example of the method template:

```
<Item type="Method" action="%action name%" %specific API attributes%>
  <targetScope>
    <Item type="Method" action="%builder method name%" %builder method attributes%>
      %builder method properties%
    </Item>
  </targetScope>
  %specific API properties%
</Item>
```

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. The following is the template for the Scope builder method:

```
namespace $(pkgname)
{
    public class $(clsname): ScopeBuilderBase
    {
        $(MethodCode)
    }
}
```

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

Anyone who implements a scope builder method will be forced to override all abstract methods.

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 - Programmer's Guide*.

Input Item Format

The targetScope Property enables you to specify the Scope builder method for Configurator Services API methods using AML syntax.

The syntax is as follows:

```
<Item type="Method" action="%action name%" %specific API attributes%>
  <targetScope>
    <Item type="Method" action="%builder method name%" %builder method attributes%>
      %builder method properties%
    </Item>
  </targetScope>
  %specific API properties%
</Item>
```

Where:

%action name%	Name of the action associated with the Configurator API method.
%specific API attribute%	List of API attributes specifically for a particular method. For example, “fetch” is an API attribute for the cfg_GetValidCombinations and cfg_GetConflicts methods.
%builder method name%	Name of the server method that builds the Scope object model based on the business logic being used.
%builder method attributes%	Attributes that are needed for the builder method to process a business Item.
%builder method properties%	Properties that are needed for the builder method to process the business item.
%specific API properties%	List of the API properties specific to a particular method. For example, the “condition” property is specific to the cfg_GetValidCombinations method.

%specific API attributes% and ***%specific API properties%*** define the method signature that is specified in the action attribute.

%builder method attributes% and ***%builder method properties%*** should contain all the information that is necessary for building the Scope object.

> **Important**
>
> Samples are provided for each method in 5 API Usage.

Scope Resolver

Scope Resolver is an internal module that is used as part of the Scope Builder process. It is used to parse request AML, call for the Scope builder method, and cache the Scope builder method result. The following description is for information only, to make internal processing clearer:

```
public class ScopeResolverModule
{
    public Scope BuildScope(Item item)
    {
        ScopeBuilderBase builder = GetBuilder(item);
        builder.Init(item, this.serverConnection);
        return this.cache.CacheDriverNotNull(
            CachableScopeContainer.GetKey(builder.GetCustomKey()),
            list => CachableScopeContainer.GetInstance(builder, this.cacheItemType)).Scope;
    }

    private ScopeBuilderBase GetBuilder(Item scopeItem) { }
}
```

Caching

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

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

The Scope object can be invalidated automatically. You need to create the following collections to make this possible:

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

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

Each Item Type in this list should be a polysource item for ScopeCacheDependency.

- List of Item IDs. This list contains the ID of each Item that was used to build the Scope.

```
public override string[] GetGuidsItemDependsOn()
{
    return new string[] {
        "item_id_color",
        "item_id_red",
        "item_id_wheelsize",
        "item_id_17inch"
    };
}
```