



Rules



Contents

Chapter 1: Overview	1
Overview of Rules	2
Rules Workflow	2
About Family Rule Projects	2
Configure a Family to Use Family-Level Rules	3
Access the APM Rules Editor	5
Access a Family Rule Project	6
Chapter 2: Compiling Rules	8
About Compiling Rules	9
Compile the Rules for a Specific Family	9
Compile the Rules for a Specific Rules Library Project	9
Compile the Entire Database	10
Chapter 3: Rules Library	11
About Rules Library	12
Access the Rules Library	12
Add a Folder to the Rules Library	13
Delete a Folder from the Rules Library	13
Create a Rules Library Rule Project	14
Add a Reference to a Rules Library Project	14
Modify a Rules Library Rule Project	15
Delete a Rules Library Rule Project	15
Chapter 4: Installation	16
Install the APM Rules Editor	17
Chapter 5: Reference	22
About Family- and Field-Level Rules	23
About Rule Code Storage Options	23

About Rule Terminology and Concepts	24
About Rules Project References	25
About Family-Level Rules	26
About Field-Level Rules	28

Copyright Digital, part of GE Vernova

© 2024 GE Vernova and/or its affiliates.

GE Vernova, the GE Vernova logo, and Predix are either registered trademarks or trademarks of GE Vernova. All other trademarks are the property of their respective owners.

This document may contain Confidential/Proprietary information of GE Vernova and/or its affiliates. Distribution or reproduction is prohibited without permission.

THIS DOCUMENT AND ITS CONTENTS ARE PROVIDED "AS IS," WITH NO REPRESENTATION OR WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF DESIGN, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. ALL OTHER LIABILITY ARISING FROM RELIANCE UPON ANY INFORMATION CONTAINED HEREIN IS EXPRESSLY DISCLAIMED.

Access to and use of the software described in this document is conditioned on acceptance of the End User License Agreement and compliance with its terms.

Chapter 1

Overview

Topics:

- [Overview of Rules](#)
- [Rules Workflow](#)
- [About Family Rule Projects](#)
- [Configure a Family to Use Family-Level Rules](#)
- [Access the APM Rules Editor](#)
- [Access a Family Rule Project](#)

Overview of Rules

Rules consist of code that is written in Visual Basic.Net (VB.Net), a programming language that is compatible with the language in which APM is written. If you have sufficient knowledge of writing VB.Net code, you can write rule code to be executed when certain changes occur in the APM database. You can write [family-level](#) or [field-level](#) rules.

As an alternative to family-level rules, you can use family policies to configure certain actions to occur when a record changes in the APM database. Family policies are created in a user interface where knowledge of Visual Basic.Net (VB.Net) is not required.

Note: For a single family, you can write family-level rules or family policies, not both. You can, however, use the Baseline Rule node in a family policy to execute any existing APM baseline rules that correspond to the policy's family and trigger. You can also use field-level rules and family policies for the same family.

Rules Workflow

Note: This workflow assumes that the APM Rules Editor extension in Microsoft has already been installed.

This workflow provides the basic, high-level steps for using this module. The steps and links in this workflow do not necessarily reference every possible procedure.

1. If you want to configure a family-level rule for a family, [specify in Configuration Manager that family-level rules should be used for that family](#).
2. [Open the Rules Editor](#), and then access either a [family rule project](#) or the [Rules Library](#).
3. If you accessed the Rules Library, then you can [add a folder to the library](#).
4. [Create a new Rules Library rule project](#).
5. [Add a reference to the project](#).

About Family Rule Projects

A family rule project is an organizational unit that provides the coding infrastructure within which the rules for that family will be written and stored.

- For baseline families, a family rule project exists for any family for which family-level or field-level rules have been defined within that specific family. For baseline subfamilies that inherit all their behaviors from higher-level families, a family rule project will not exist.
- For custom families, the family rule project is created the first time that you access the family-level or field-level rules for the family.

More Details

Family rules can be viewed and managed in the APM Rules Editor. When you open the rule project for a family, the family rule project will be selected by default. Below the project name is a list of all the items that are currently included in the project.

By default, the content of the project will consist of references and files. Each family rule project contains a file for the family itself and a file for each field within the family. A file will be created for each field that is defined directly within that family and each field that is spread down from a family where the field can be customized at the sublevel.

As you add fields to a family, new files will be added to the family rule project. You can select any file to view and modify the code that is stored within it.

Within each file, a class is defined for the corresponding family or field. Each class serves as the organizational unit within which actual family-level and field-level customization code exists. The name of the file matches the name of the class defined within it, which in turn corresponds to the field ID or family ID (i.e., not the family or field caption).

Note: Changing the ID for a field when the field already has a VB.Net file will cause APM to create a new file based on the new ID. The old file will not be deleted but will be disconnected from the field.

You can open multiple rule projects at a time. When you do so, each rule project will appear as a root-level entry in the **Solution Explorer** pane.

When you access the family rule project for a baseline APM family, the corresponding baseline Rules Library project will also appear in the **Solution Explorer** pane. The baseline Rules Library project has the same name as the family rule project with `_Base` appended to it.

You cannot modify the baseline Rules Library project, but it is displayed so that you can easily view and debug the baseline rule code to understand the baseline family and field rules.

Configure a Family to Use Family-Level Rules

About This Task

If you want to configure a family-level rule for a family, you first must specify in **Configuration Manager** that family-level rules should be used for that family.

Important: For a single family, you can use family-level rules or family policies, not both.

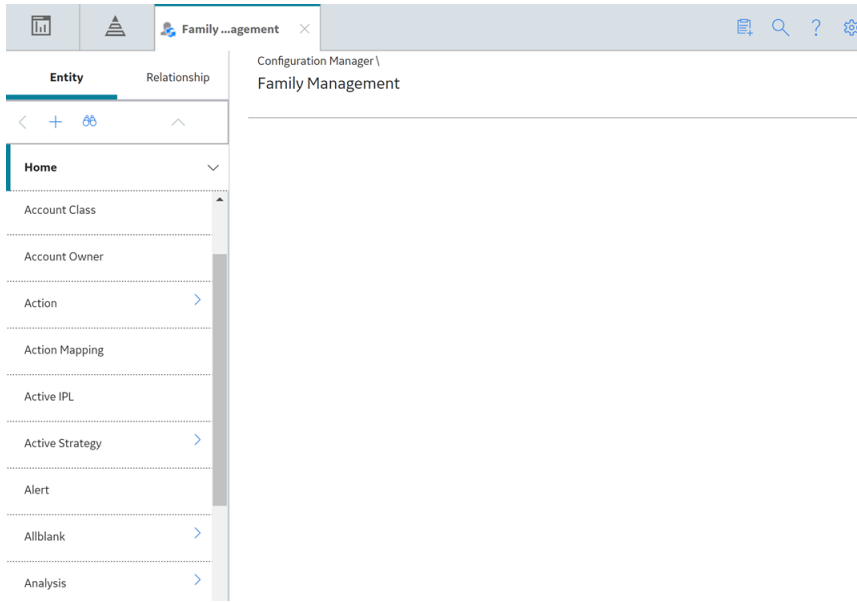
Procedure

1. Access the **Configuration Manager** page, and then select **Data Configuration**.

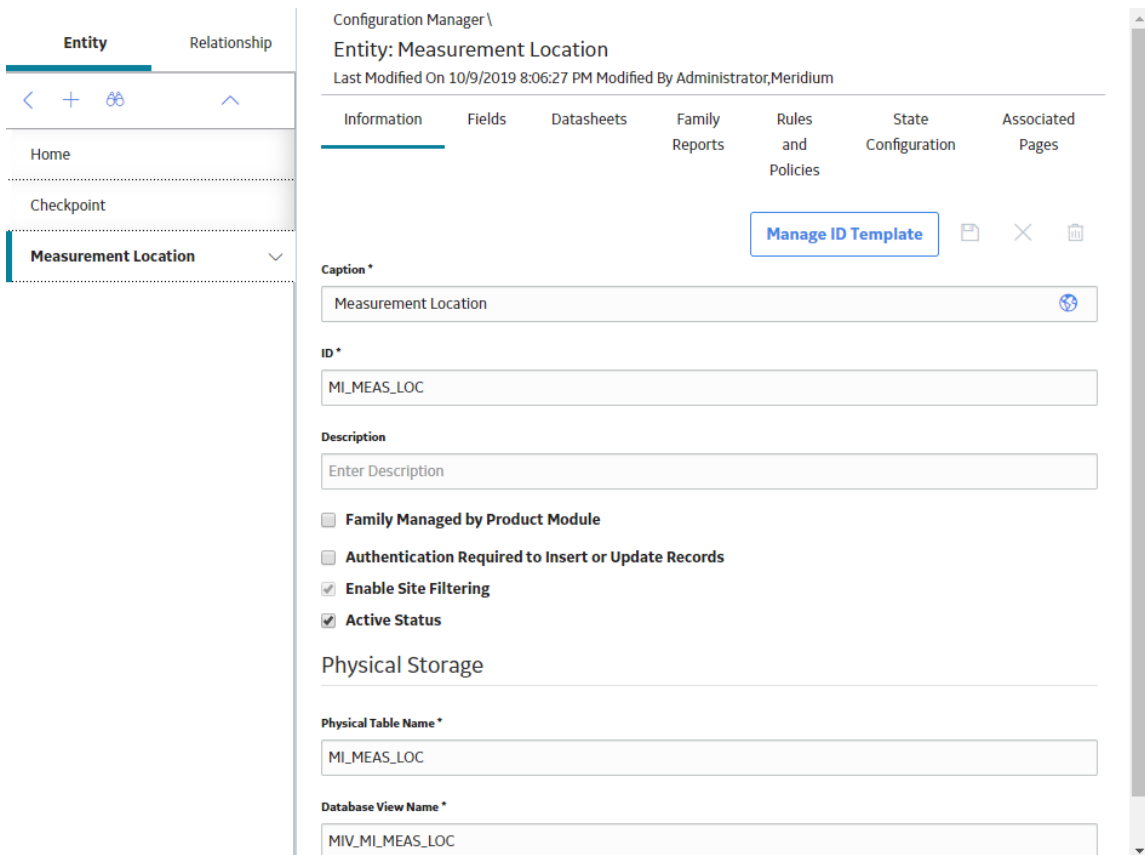
The **Data Configuration** page appears.

2. Select **Family Management**.

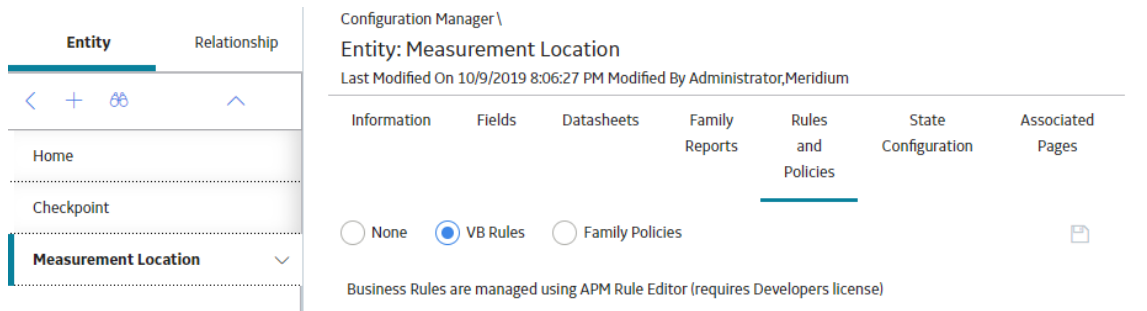
The **Family Management** page appears.




3. Select the family for which you want to use family-level rules.
The workspace for the selected family appears.



4. In the workspace, select **Rules and Policies** tab.
The **Rules and Policies** section appears.



5. Select the **VB Rules** radio button, and then select .

Important: If **Family Policies** was previously selected, VB Rules will be used instead of any configured family policies.

Next Steps

- [Access the APM Rules Editor.](#)

Access the APM Rules Editor

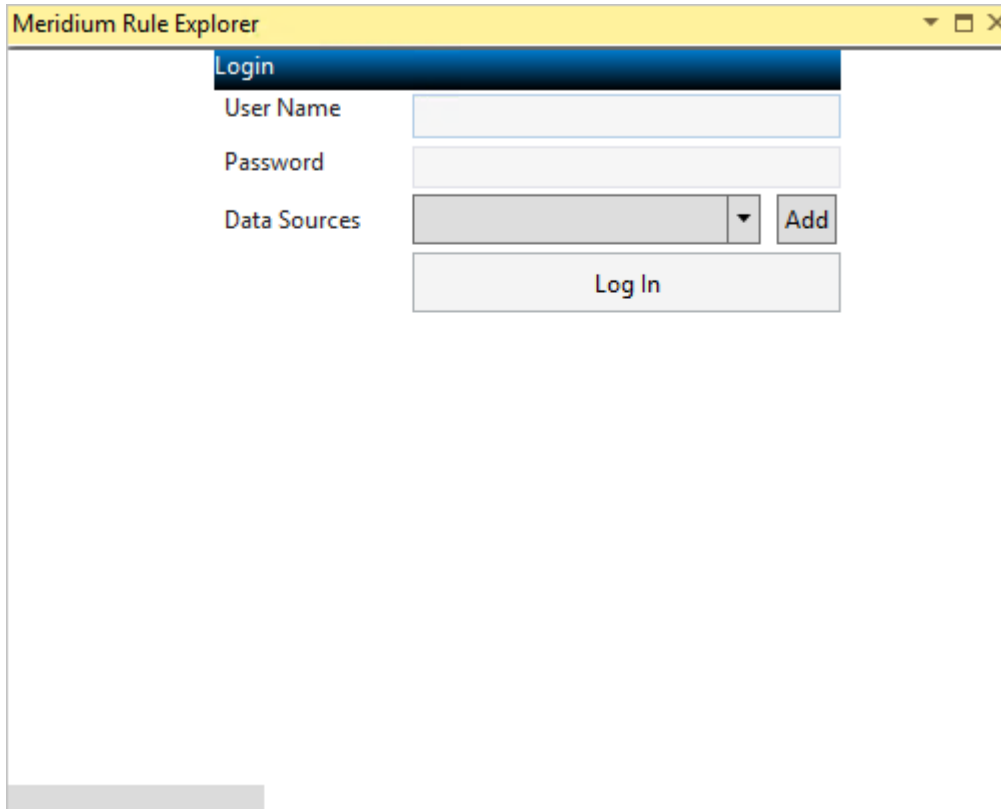
Before You Begin

If you want to configure a family-level rule for a family, [specify in Configuration Manager that family-level rules should be used for that family.](#)

Procedure

1. On the APM Server, access Microsoft Visual Studio.
2. On the **Tools** menu, select APM Rules Editor.

The APM Rule Explorer pane appears, prompting you to log in.



3. In the **User Name** box, enter a user name that can log in to the data source whose rules you want to manage.
4. In the **Password** box, enter the password that is associated with the specified Security User.
5. In the **Data Sources** list, select the data source whose rules you want to manage.
6. Select **Log In**.

The APM Rule Explorer pane displays the list of entity families.

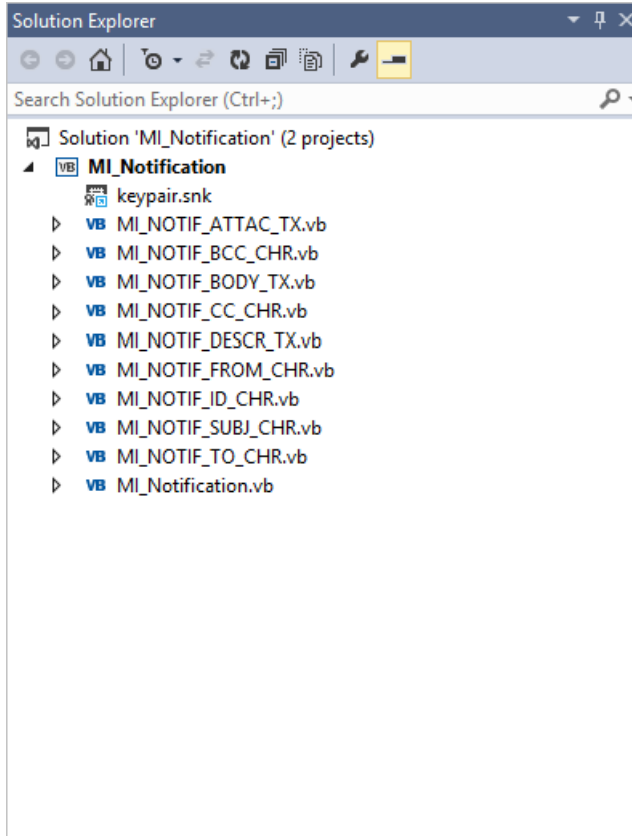
Access a Family Rule Project

Procedure

1. [Access the APM Rules Editor](#).
2. Select **Families** or **Relationships**.
3. Select the family whose rule project you want to open.

Note: Only green folders contain rule projects.

The **Solution Explorer** pane appears, displaying the family rule project.



Chapter 2

Compiling Rules

Topics:

- [About Compiling Rules](#)
- [Compile the Rules for a Specific Family](#)
- [Compile the Rules for a Specific Rules Library Project](#)
- [Compile the Entire Database](#)

About Compiling Rules

After you make changes to entity or relationship family rules, you must compile the family for the changes to be applied. Likewise, after you make changes to a Rules Library rule project, you must compile the project for the changes to be applied.

Note: To [compile an entire database](#), you must compile all entity family rules, all relationship family rules, and all projects in the Rules Library.


Compile the Rules for a Specific Family

About This Task

After you make changes to entity or relationship family rules, you must compile the family for the changes to be applied. The following instructions provide details on compiling the rules for a single family. Families cannot be compiled until the family rule project has been created.

Note: To [compile an entire database](#), you must compile all entity family rules, all relationship family rules, and all projects in the Rules Library. You can use the following instructions to compile rules for entity and relationship families.

Procedure

1. [Access the Rules Library](#).
2. Select **Families**.
3. Select the family whose rules you want to compile.
4. On the toolbar, select .

The family rules are compiled, and the result appears in the **Output** pane.

Note: If it is not already selected, in the **Output** pane, select to show output from the APM Server Rules.


Compile the Rules for a Specific Rules Library Project

About This Task

After you make changes to a Rules Library rule project, you must compile the project for the changes to be applied. The following instructions provide details on compiling a single Rules Library rule project.

Note: To [compile an entire database](#), you must compile all entity family rules, all relationship family rules, and all projects in the Rules Library.


Procedure

1. [Access the Rules Library](#).
2. Select **Rules Library**.
3. In the tree, select the Rules Library project that you want to compile.
4. On the toolbar, select .

The Rules Library rule project is compiled, and the result appears in the **Output** pane.

Compile the Entire Database

Procedure

1. [Access the Rules Library](#).
2. Select **Families**.
3. On the toolbar, select  .

The compilation process begins. Rules Library project are compiled first, followed by entity families and then relationship families. The progress of the compilation is shown in the **Output** pane.

Chapter 3

Rules Library

Topics:

- [About Rules Library](#)
- [Access the Rules Library](#)
- [Add a Folder to the Rules Library](#)
- [Delete a Folder from the Rules Library](#)
- [Create a Rules Library Rule Project](#)
- [Add a Reference to a Rules Library Project](#)
- [Modify a Rules Library Rule Project](#)
- [Delete a Rules Library Rule Project](#)

About Rules Library

The APM Rules Library serves as a repository of rule projects that can be referenced from family rule projects. By storing rule code in the Rules Library and referencing it from family rule projects, you can reuse rule code repeatedly. This results in more efficient and organized rule authoring than using family-level and field-level customization alone.

The Rules Library contains two types of projects:

- **APM Rule Projects:** Rule projects that are distributed as part of the APM baseline product and are referenced within the APM baseline families. These projects are stored in the **APM** folder in the Rules Library. The **APM** folder contains the following subfolders that organize the APM rule projects according to the APM module in which they are used.
 - **Module-specific subfolders:** Folders that store rule projects, organized according to the module in which each project is used (e.g., Calibration, Inspection, Metrics, etc.). These folders contain baseline rules, but they are not associated with a specific baseline family in the same way as the rule projects in the **Root Entity Families** folder. In other words, the Calibration rule project is not necessarily associated specifically with the Calibration family. It simply stores rule code that is used within Calibration Management.
 - **Root Entity Families:** Folders that store rule projects that are associated with specific, baseline entity families. The name of the rule project corresponds to the name of the family for which it stores rule code.
 - **Relationship Families:** Folders that store rule projects that are associated with specific, baseline relationship families. The name of the rule project corresponds to the name of the family for which it stores rule code.

You can view the rule projects that are delivered with APM and reference them in any family you like, and you can copy any of the rule code and use it as the basis for creating your own rule projects. You cannot, however, modify or delete these projects, and you cannot add new projects to the **APM** folder.

- **Client Rule Projects:** Rule projects that are created by customers to support their own unique implementations of APM. These projects are stored in the **Client** folder in the Rules Library.

Access the Rules Library

Procedure

1. [Access the APM Rules Editor.](#)
2. Select **Library**.

The Rules Library appears.

Next Steps


- [Add a Folder to the Rules Library.](#)

Add a Folder to the Rules Library

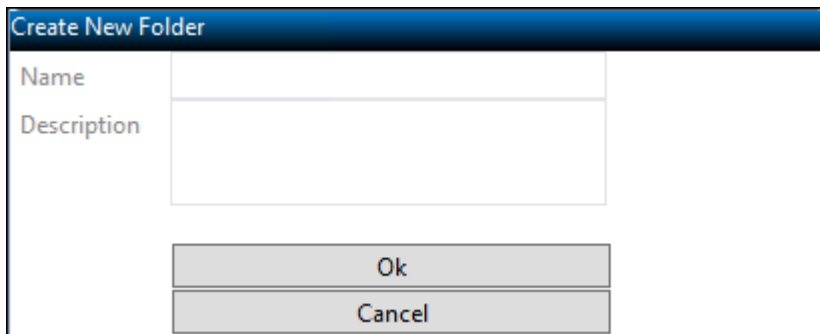
About This Task

In the Rules Library, each rule project must be stored within a folder. You can add folders only under the **Client** folder.

Procedure

1. [Access the Rules Library.](#)
2. Select the **Client** folder or a subfolder of the **Client** folder to which you want to add a folder.
3. Select .

The **Create New Folder** section appears.



Create New Folder	
Name	<input type="text"/>
Description	<input type="text"/>
<input type="button" value="Ok"/>	
<input type="button" value="Cancel"/>	

4. Enter a name and description for the folder.
 5. Select **OK**.
- The new folder is added in the selected location.

Next Steps

- [Create a Rules Library Rule Project.](#)

Delete a Folder from the Rules Library

About This Task

You can delete folders only under the **Client** folder.

Procedure

1. [Access the Rules Library.](#)
2. Select the **Client** folder or a subfolder of the **Client** folder from which you want to delete a folder.
3. Select .


A window appears, asking you to confirm that you want to remove the folder.

4. Select **Yes**.

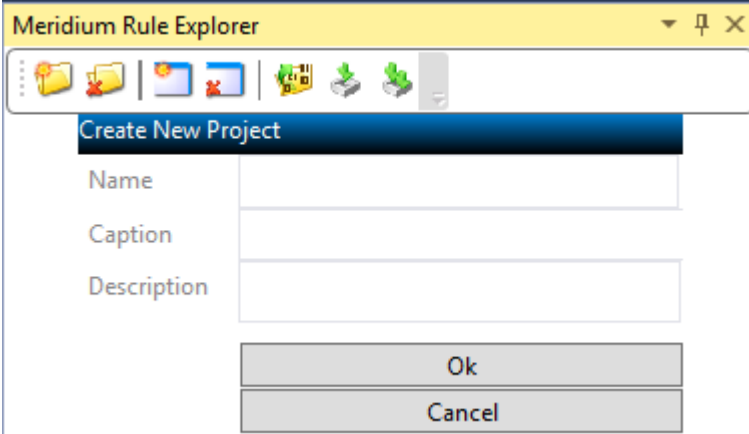
The selected folder is deleted from the Rules Library.

Create a Rules Library Rule Project

Procedure

1. [Access the Rules Library.](#)
2. Select the **Client** folder or a subfolder of the **Client** folder to which you want to add a rule project.
3. On the toolbar, select .

The **Create New Project** section appears.



4. Enter a name, caption, and description for the project.
5. Select **OK**.

The rule project is created.


Next Steps

- [Add a reference to the Rules Library project](#) from the family project. This will make the rules that are stored within the Rules Library project available for use within the family rule project.
- Within the family rule project, call the parts of the referenced Rules Library project that you want to use. How and where you make the calls will depend on how the Rules Library project is organized and where you want to invoke certain functionality.

Note: You can create references to APM rule projects and to Client rule projects.

Add a Reference to a Rules Library Project

Procedure

1. [Access the APM Rules Editor.](#)
2. Open the family rule project to which you want to add a reference.
3. In the **Solution Explorer** pane, select .

The **Properties** pane for the selected family rule project appears.

4. Select **References**, and then select **Add**.

The **Reference Manager** window appears.

5. Navigate to the reference(s) you want to add, and then select the appropriate check boxes.
6. Select **OK**.

The new references appear in the **Properties** pane, in the **References** section.

Modify a Rules Library Rule Project

About This Task

You can modify Rules Library rule projects only under the **Client** folder or one of its subfolders.

Procedure

1. [Access the Rules Library](#).
2. In the **Client** folder structure, select the rule project that you want to modify.

The content of the rule project is displayed in the **Solution Explorer** pane.

3. Modify the code as needed.
4. Build the project.
5. Save the rule code.
6. Close the **Solution Explorer** pane.


Delete a Rules Library Rule Project

About This Task

You can delete Rules Library rule projects only from the **Client** folder or one of its subfolders.

Procedure

1. [Access the Rules Library](#).
2. In the **Client** folder structure, select the rule project that you want to delete.

3. On the toolbar, select .

A window appears, asking you to delete the project.

4. Select **Yes**.

The selected project is deleted from the Rules Library.

Chapter 4

Installation

Topics:

- [Install the APM Rules Editor](#)

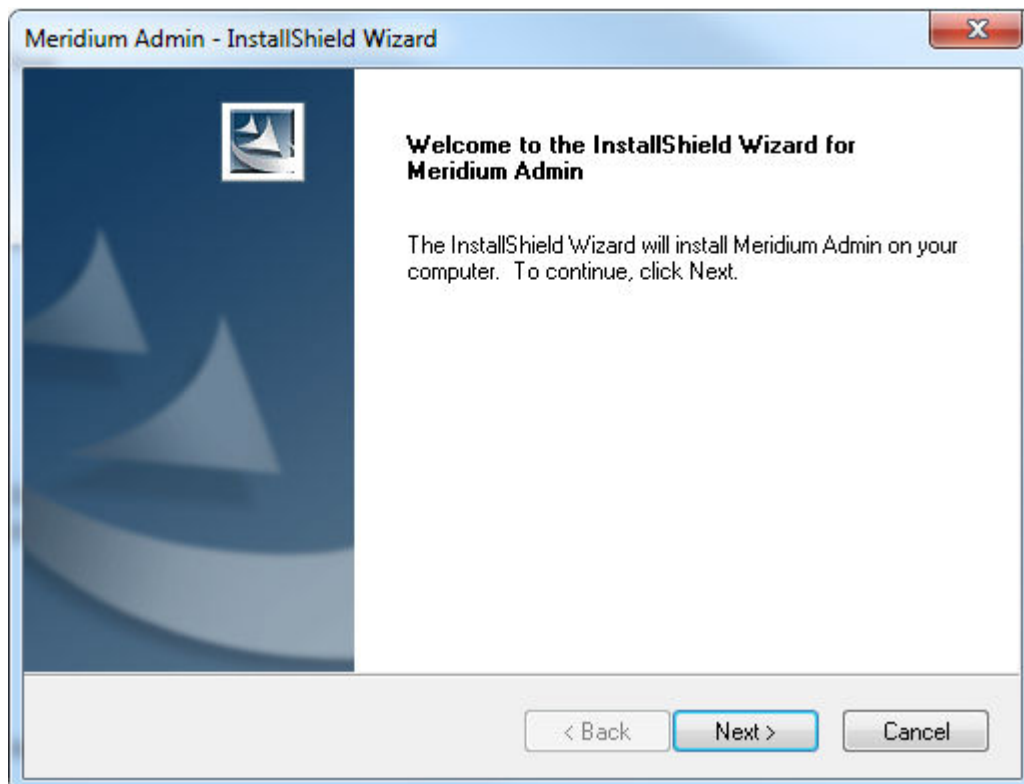
Install the APM Rules Editor

Before You Begin

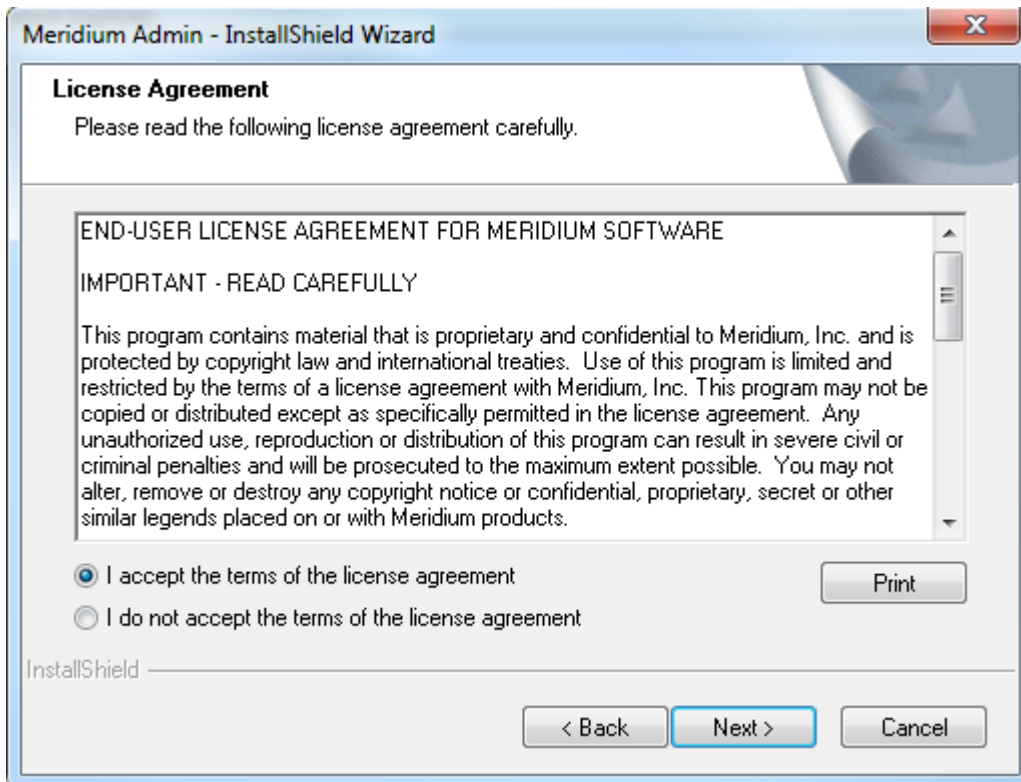
- Microsoft Visual Studio 2017 or 2019 Professional must be installed on every workstation where you want to work with rules in the APM system.
- MSXML must also be installed on these workstations.
- You must be logged in as the administrator for the system.

Procedure

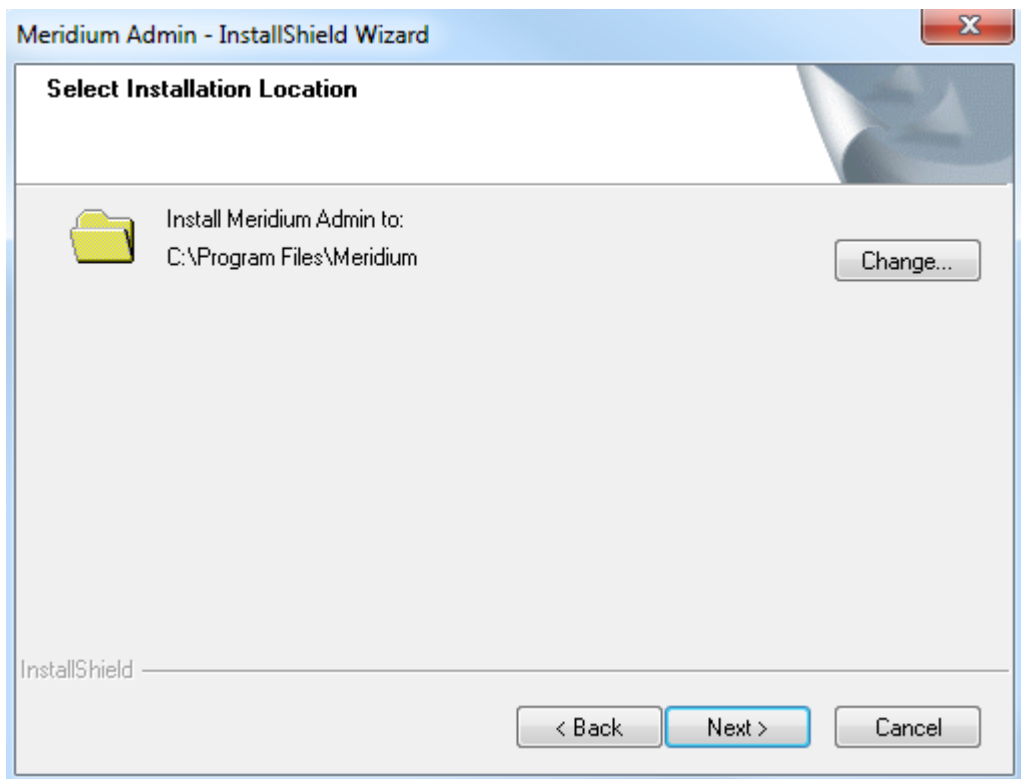
1. On the machine that will serve as the APM rules editor, access the APM distribution package, and then navigate to the folder \\General Release\Meridium APM Setup\Setup\Admin.
2. Open the file Setup.exe.
The **Meridium Admin - InstallShield Wizard** screen appears.



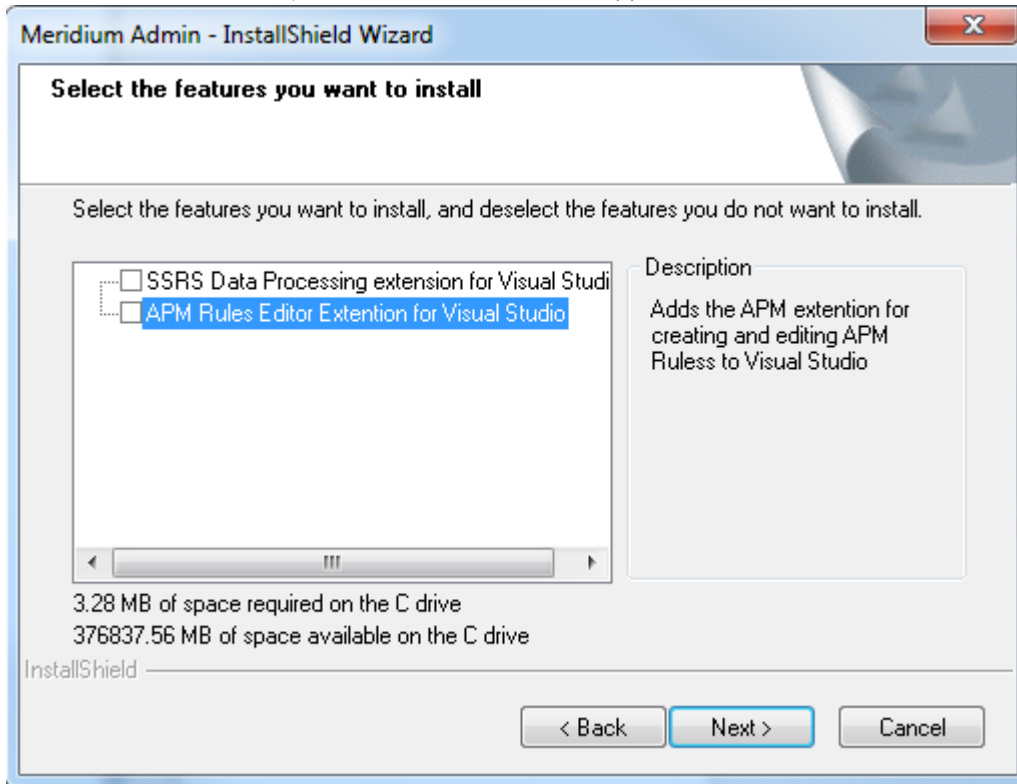
3. Select **Next**.
The **License Agreement** screen appears.



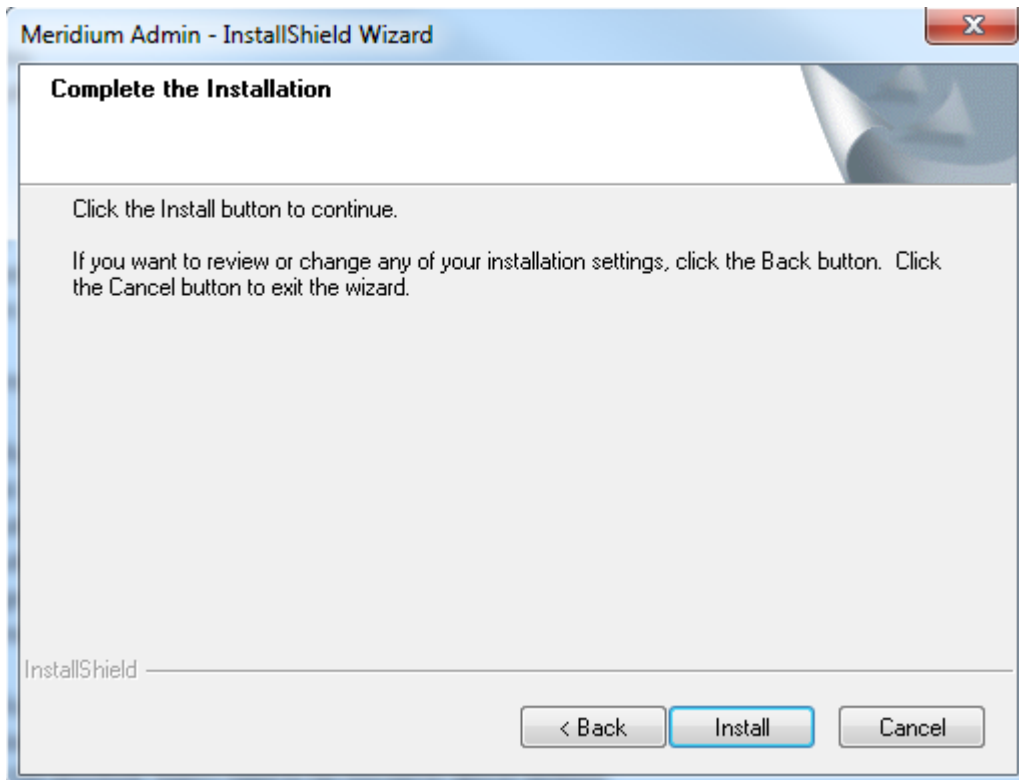
4. Read the License Agreement and, if you agree, select the **I accept the terms of the license agreement** option. Then, select **Next** button. The **Select Installation Location** screen appears.



5. Select **Next** to accept the default location.
The **Select the features you want to install** screen appears.

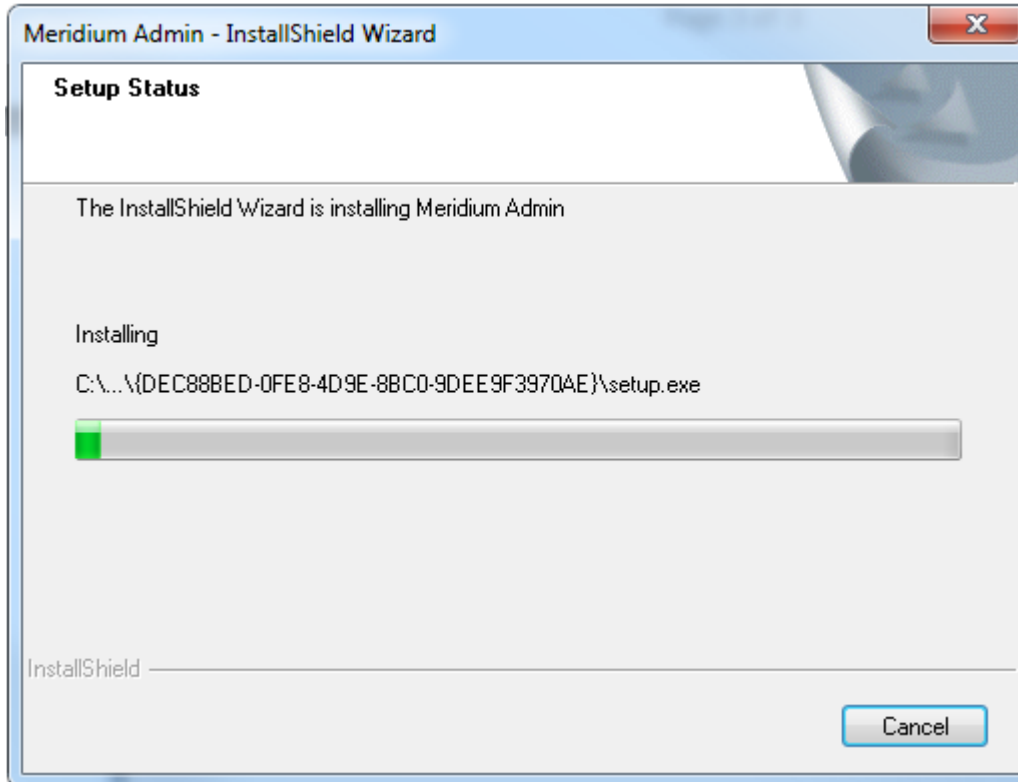


6. Select the **APM Rules Editor Extension for Visual Studio** option.
APM performs a check to make sure that your machine contains the required prerequisites for the features that you want to install. If one or more prerequisites are missing or there is not enough space on the machine, a dialog box will appear, explaining which prerequisites are missing or asking to free up space. If this occurs, close the installer, install the missing prerequisite or free up some space, and then run the installer again.
7. Select **Next**.
The **Complete the Installation** screen appears.



8. Select **Install**.

The **Setup Status** screen appears, which displays a progress bar that shows the progress of the installation process. After the progress bar reaches the end, a message appears, indicating that Meridium Admin is installed successfully. Optionally, you can select to launch the APM System Administration tool when the installer window closes.



9. Clear the **Launch APM System Administration now** box, and then select **Finish**.
10. If you have Microsoft Visual Studio 2019 Professional installed, go to the `C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\Common7\IDE` folder.
11. Access the `devenve.exe.config` file in an application that can be used to modify text files (for example, Notepad++).
12. In the file, locate the following text:

```
<assemblyIdentity name="System.Runtime.CompilerServices.Unsafe"
publicKeyToken="b03f5f7f11d50a3a" culture="neutral"/>
```

13. Below the line, within the `bindingRedirect` tag, ensure that the value of the `oldVersion` parameter is `0.0.0.0-4.0.6.0` and the value of the `newVersion` parameter is `4.0.6.0`.
14. Save and close the file.

Results

- The APM rules editor is installed.

Chapter 5

Reference

Topics:

- [About Family- and Field-Level Rules](#)
- [About Rule Code Storage Options](#)
- [About Rule Terminology and Concepts](#)
- [About Rules Project References](#)
- [About Family-Level Rules](#)
- [About Field-Level Rules](#)

About Family- and Field-Level Rules

In APM, [family-level](#) and [field-level](#) rules consist of code that determines how records in the APM database will behave under specific conditions. Rule code is written in Visual Basic.Net (VB.Net), a programming language that is compatible with the language in which APM is written. In this way, you can specify that when certain actions are taken in APM, certain rules should be executed.

Important: Modifying rules without the proper knowledge and expertise could cause your system to work improperly.

The purpose of writing business rules for a family is to control how records in that family will behave when you work with those records in APM. The rule code itself can be stored in two locations:

- Within the family rule project itself.
- Within the Rules Library.

Rules can range from very simple to highly complex. If you have sufficient knowledge of writing VB.Net code, you can use Microsoft Visual Studio to customize your system to suit the specific needs of your organization.

Writing complex rules requires knowledge of VB.Net that exceeds the scope of this documentation. The purpose of this documentation is to explain the basic structure of family rule projects and to describe the tools that are available to help you write rules. In addition, we provide some basic information about rule code itself to help you understand and navigate your existing rules and some basic examples of custom rule code.

Note: Family policies allow you to make changes similarly to family-level rules, but family policies are created in a user interface where knowledge of Visual Basic.Net (VB.Net) is not required.

About Rule Code Storage Options

Custom family-level and field-level rules can be stored in two places:

- Within the family rule project itself.
- Within the Rules Library.

Storing family rules within the family rule project itself means that the actual rule code is stored in the family and field classes within a family rule project. This form of rule code storage is acceptable but can be cumbersome as it can result in a large amount of rule code that must be maintained on a family-by-family basis. In addition, rule code that is stored within family rule projects applies only to one family. To apply the same rule code to another family, you would have to copy the rule code and paste it into another family rule project.

The Rules Library, on the other hand, stores rule code in projects that can be referenced from family rule projects. In this way, you can store the actual VB.Net code in one central location and then apply it to multiple families. This method of rule code storage offers many advantages, including limiting the amount of code that must be maintained and increasing the ease and efficiency with which that code can be applied to other families.

The Rules Library, however, also imposes some limitations. To take advantage of the exact same rule code across multiple families, you would need to have multiple families that should behave exactly the same way. It is more likely, however, that you will have multiple families that should behave in similar ways.

For this reason, you will probably want to use a combination of the two rule code storage methods. In the Rules Library, you can store code that will serve as the foundation on which family-level and field-level

rules are built. After referencing that code from a family rule project, you can extend the rule through family-level and field-level customization.

About Rule Terminology and Concepts

To modify and create rules, you must understand the following terminology and concepts:

- Rule projects
- Classes
- Functions
- Inheritance

Rule Projects

A rule project is a container for rule code. APM uses two types of rule projects:

- **Family Rule Project:** Stores all the rules for a given family.
- **Rules Library Project:** Stores all the rules that exist for that project.

Rule projects contain references and files, also called code items. The files contain rules that are defined for that project. The file structure for family projects is determined by the content of the family. The file structure of a Rules Library project is determined by the project owner and can be customized as necessary to meet the requirements of the project.

Classes

A class is a VB.Net element that serves as a container for storing objects. Classes are defined within the files that exist in a rule project.

- In family rule projects, APM automatically defines classes within the files that exist for the family itself and the fields within that family.
- In baseline Rules Library projects, classes serve as containers for the code within those projects.
- In custom Rules Library projects, you can define your own classes as needed. When you create a new project, one file will be created within that project, and within that file, a default class will be defined.

Any VB.Net class can be inherited from another class so that the rules defined in one class can be reused as often as needed.

The following code excerpt shows an example of the default class Class1 that is created when you create a new Rules Library project:

```
Option Strict On
Option Explicit On
Imports GE Digital APM.Core.DataManager
Imports GE Digital APM.Core.DataManager.Customization
Imports GE Digital APM.Core.Internals
Imports GE Digital APM.Core.Metadata
Imports GE Digital APM.Core.Security.ApplicationUser
Imports GE Digital APM.Core.Uom
Imports System
Imports System.Xml
Public Class Class1
Private Sub New()
MyBase.New
'
'TODO: Add constructor logic here
'
```

```
End Sub
End Class
```

In this example, the lines `Public Class Class1` and `End Class` define the class; the code in between these two lines of text represents all the objects that belong to the class.

Functions

A function is a block of rule code that defines a Function procedure. Functions can be defined to invoke specific behaviors. The standard field-level rules that are available in APM are defined through functions. In the following example, the `IsRequired` function is between the two lines `Public Class Class1` and `End Class`.

```
Option Strict On
Option Explicit On
Imports GE Digital APM.Core.DataManager
Imports GE Digital APM.Core.DataManager.Customization
Imports GE Digital APM.Core.Internals
Imports GE Digital APM.Core.Metadata
Imports GE Digital APM.Core.Security.ApplicationUser
Imports GE Digital APM.Core.Uom
Imports System
Imports System.Xml
Public Class Class1
Public Overrides Function IsRequired() As Boolean
Return True
End Function
End Class
```

Inheritance

Inheritance allows one class to use the behaviors defined in another class.

- The class that is inherited is considered the base class and serves as a foundation for the functionality of the class that inherits it.
- Any class that inherits from another class is considered a derived class.

All the functions and behaviors defined in the base class are automatically applied to the derived class. Within the derived class, code can be written to extend or override specific functions defined in the base class. Inheritance allows you to create new classes based on existing classes and is an important component of the Rules Library and baseline rule storage.

Inheritance is achieved through an `Inherits` statement in the derived class. For instance, in the following example, the class `MI_RCA_ANALY_COST_NBR` (the derived class) inherits the class `EntityFieldCustomization` (the base class).

```
Public Class MI_RCA_ANALY_COST_NBR
Inherits EntityFieldCustomization
End Class
```

About Rules Project References

Creating Rules Library projects is only the first step in implementing family and field customizations using the Rules Library. After a Rules Library project has been created, it must be called from the family or field class of the family or field by which it will be used. You must complete two main steps to call a Rules Library project from a family project:

1. First, you must add a reference to the Rules Library project from the family project. This will make the rules that are stored within the Rules Library project available for use within the family rule project.
2. Second, within the family rule project, you must make calls to the parts of the referenced Rules Library project that you want to use. How and where you make the calls will depend on how the Rules Library project is organized and where you want to invoke certain functionality.

After the Rules Library project has been called within the family rule project, it can be extended via the family rule project in order to customize the rule for that specific family.

Note: You can create references to APM rule projects and to Client rule projects.

About Family-Level Rules

Whereas field-level rules control the behavior of specific fields within a record, family-level rules reside within the code item that represents an entity family and control actions performed against the entire record.

Note: As an alternative to writing basic family-level rules, you can use family policies to configure the actions to be performed against a record. Family policies are created in a user interface where knowledge of Visual Basic.Net (VB.Net) is not required. For a single family, you can write family-level rules or family policies, not both. You can, however, use the Baseline Rule node in a family policy to execute any existing APM baseline rules that correspond to the policy's family and trigger.

Family-level rules provide flexibility in determining how records will behave. Some of the most common uses of family-level rules include:

- Managing record links.
- Calculating values.
- Sending email notifications.

You can create family-level rules by developing custom code.

APM supports the following family-level rule types.

Rule Type	Stores logic that is executed...
BeforeInsert	Before a record is created.
AfterInsert	After a record is created.
BeforeUpdate	After changes have been made to a record, before those changes are saved to the database.
AfterUpdate	After changes to a record have been saved.
BeforeDelete	Before a record is deleted.
AfterDelete	After a record has been deleted.

A change to a record in the APM database will trigger the appropriate rule regardless of the current user's permissions. However, if a user does not have permissions for an action a rule is taking, the rule will not execute and the transaction will be rolled back and no changes will be made. Similarly, the transaction will be rolled back if an error occurs during the rule's execution.

Example: AfterInsert

One use of an AfterInsert rule is to create a link between two families after a new record is created. Consider an example where, when you create a Functional Location record, you want to link it to the Site Reference record that represents the site that

contains the functional location. You can accomplish this by creating a family-level rule for the Functional Location family.

The following code shows an example of an AfterInsert rule that you could write to create a link between the Functional Location record and the Site Reference record after the Functional Location record is created. You would insert this rule into the family-level code item for the Functional Location family.

```
Public Overrides Sub
AfterInsert ()
    ManageSiteReference ()
End Sub

Private Sub Manage Site Reference ()
    Dim session As EntitySession =
CurrentEntity.Session

    If session Is Nothing Then
        session = New
EntitySession (ApplicationUser)
    End If

    Dim currentSite As SiteReference
= SiteReference.LoadExisting (session, CurrentEntity)
    Dim newSite As SiteReference =
SiteReference.RetrieveSiteReference (session,
CurrentEntity)

    If currentSite Is Nothing
AndAlso newSite Is Nothing Then
        Return
    End If

    If currentSite Is Nothing And
Not newSite Is Nothing Then

        newSite.AddAssetToSite (CurrentEntity)
    ElseIf Not currentSite Is
Nothing And newSite Is Nothing Then
        If Not
CurrentEntity.Fields ("MI_FNCLOC00_SAP_SYSTEM_C").Value
Is DBNull.Value Then

            currentSite.RemoveAssetFromSite (CurrentEntity)
        End If
    ElseIf currentSite.Key <>
newSite.Key Then

        currentSite.RemoveAssetFromSite (CurrentEntity)

        newSite.AddAssetToSite (CurrentEntity)
    End If
End Sub
```



About Field-Level Rules

Field-level rules define how a field will behave under certain circumstances. The field-level rules for a given field are stored within the family rule project for the family to which the field belongs.

Each family rule project contains a code item for each field that exists within the family. The rules for a given field are stored in the file that corresponds to that field.

Note: Code items will not exist for fields that have been spread down from a higher-level family and are configured at the subfamily level to inherit rules from the source family. In this case, the code item that exists in the family rule project of the source family will be used for defining and executing rules at the subfamily level. If the subfamily is configured not to inherit rules from the source family, a code item will exist within the family rule project of the subfamily and will be used for defining and executing rules at the subfamily level.

The following types of rules can be defined for each family field.

Rule Type	Description	Associated Function
Required	Determines whether or not a value must be entered into a field before a record can be saved in the family.	IsRequired
Validation	Lets you define criteria that will be used to validate values that are entered into a field.	Validate
Valid Values	Lets you define a list of values that will be available for selection in the field. You will be able to select any value from those defined in the list of Valid Values.	GetPickList
Default Value	Defines the default value that will be provided for a field. When you create a new record, the default value will be provided automatically. You can accept the default value or specify a different value.	GetDefaultInitialValue
Disabled	Determines when, if ever, the field will be disabled, or locked from modification.	IsDisabled
Format	Determines the formatting that will be applied to values entered into a field.	FormatValue
Formula	Calculates the value in the field using a formula that has been specified through rules.	GetCalculatedValue

You can develop custom field-level rules manually by accessing the code item for the appropriate field and inserting the custom code. The code for each rule type must be defined within the appropriate function, which serves as a container for the code for that rule.

Example 1: Required Rule

Instead of making a field always required, you may prefer to make it required only if another field in the same record also contains a value. For example, assume that the Recommendation family contains the logical field Completed (MI_REC_COMPL_FLG), which is meant to be flagged by a user after a recommendation has been completed. The Recommendation family also contains the Completed Date field, which is meant to contain the date the recommendation was completed. Therefore, you want to create a rule so that when the Completed field is set to True, the Completed Date field is required.

To enforce this condition, you would create a Required rule on the Completed Date field that looks like this:

```
Public Overrides Function
IsRequired() As Boolean
    If
Object.Equals(CurrentEntity.Fields("MI_REC_COMPL_FLG").Value, True) Then
        Return True
    Else
        Return False
    End If
End Function
```

The portion of the code shown in red (MI_REC_COMPL_FLG) identifies the Field ID of the field that you want to use to enforce the Required rule condition, and the portion on the If line shown in blue (True) specifies the value that the field must contain.

Example 2: Validation Rule

Consider an example where the Recommendation family contains the field Create Work Request? (MI_REC_CREATE_SAP_NOTIF_FLG). Suppose you want to create a rule that validates the presence of an active external interface (e.g., Oracle) before APM triggers the creation of a work request. Should the rule return no active external interface, then no work request will be created.

You could enforce the need for a work request management system by creating the following Validation rule for the Create Work Request? field:

```
Public Overrides Function
Validate(ByRef newValue As Object) As GE Digital
APM.Core.DataManager.Customization.FieldValidationStatus
    If
InterfaceUtility.InterfacesActive(Me.ApplicationUser) Or
_
InterfaceUtility.OracleInterfacesActive(Me.ApplicationUser) Or
_
InterfaceUtility.MaximoInterfacesActive(Me.ApplicationUser)
```

```

r) Then
    Return
    FieldValidationStatus.Success
    End If

    Return
    FieldValidationStatus.Failure(InterfaceUtility.EAM_INTERF
ACES_NOT_ACTIVATED)

    End Function

```

Consider another example for a validation rule, where in the **Asset Criticality Analysis** datasheet, the **Analysis Definition Family** section contains the field **Analysis ID**. Suppose you want to create a rule that validates if the **Analysis ID** exists in the system database.

```

Public Overrides Function Validate(ByRef newValue As Object) As Meridium.Core.DataManager.Customization.FieldValidationStatus
    Dim oCmd As Command = New Command(Me.CurrentEntity.ApplicationUser)
    oCmd.CommandText = "SELECT COUNT(*) FROM [MI_ASCRTANL] analysis WHERE analysis.[MI_ASCRTANL_IS_DELET_L] = 'N' AND analysis.oCmd.Parameters.Add(New CommandParameter(DataType.Character, newValue.ToString()))

    Dim oRowSet As Rowset = oCmd.Execute(RowsetMode.RawData)

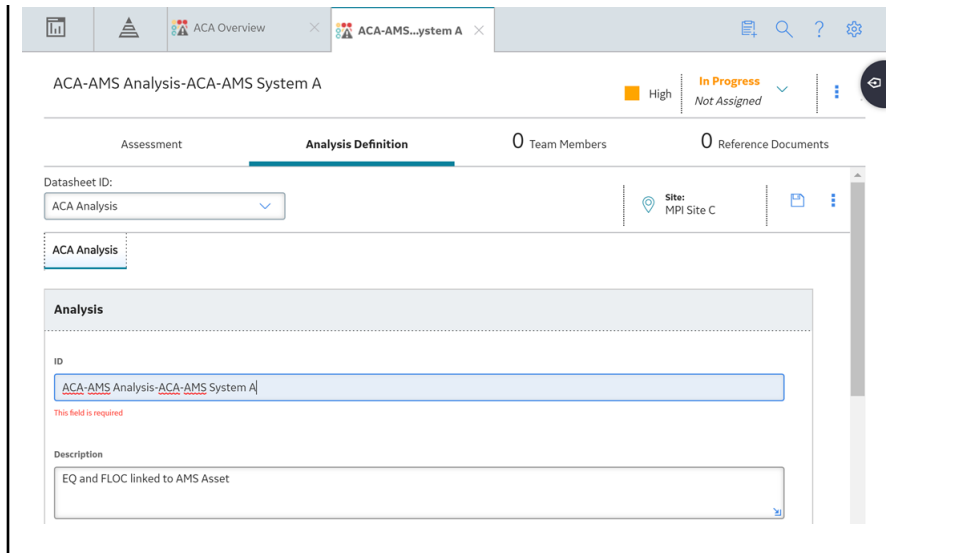
    If Convert.ToInt32(oRowSet.Rows(0)(0)) > 0 Then
        Return Meridium.Core.DataManager.Customization.FieldValidationStatus.Failure("The Analysis ID already exists in the system. Please choose a different ID.<a href=#ACA >Click</a>")
    Else
        Return Meridium.Core.DataManager.Customization.FieldValidationStatus.Success("Analysis ID length can be better.")
    End If
End Function

```

If the field validation status is a failure, then we can have the rule take the message text as input and return a failure message. We can embed a hyperlink in the validation failure error message, causing the hyperlink to be shown along with the error message under the field.

The screenshot shows a web application interface for 'ACA-AMS Analysis-ACA-AMS System A'. The page is in the 'Analysis Definition' tab. The 'Analysis' section has a 'Text input' field for 'ID' and a 'Description' field. The 'ID' field contains the text 'EQ and FLOC linked to AMS Asset'. A red error message is displayed below the 'ID' field: 'The System ID already exists in the analysis. Please choose a different ID.' The 'Description' field contains the text 'EQ and FLOC linked to AMS Asset'.

If the field validation status is a successful, then we can have the rule take the message text as input and return a success message. The message would be displayed under the field.



Example 3: Valid Values Rule

Within a list of available values, a blank value can be useful for clearing a previously selected value. When you select the blank value, it will clear whatever value had previously been selected in the list.

For example, the following code excerpt show a static Valid Values rule that will construct a list containing the values A, B, and C.

```

        Public Overrides Function
GetPickList() As DynamicPickList
        Dim pickList As DynamicPickList
        'Use MyBase.CreatePickList(True)
if you want a restricted PickList
        'Use
MyBase.CreatePickList(False) if you want an unrestricted
PickList
        pickList =
MyBase.CreatePickList(True)
        PopulatePickList(pickList)
        Return pickList

        End Function

        Public Overrides Sub
PopulatePickList(ByVal pickList As DynamicPickList)
        pickList.AddRange("A|B|
C".Split("|"c))

        End Sub

```

You can add a null value to the list by inserting the code `pickList.Add(DBNull.Value)` above the line `pickList.AddRange("A|B|C".Split("|"c))`. The resulting code would look like this:

```
Public Overrides Function
GetPickList() As DynamicPickList
    Dim pickList As DynamicPickList
    'Use MyBase.CreatePickList(True)
if you want a restricted PickList
    'Use
MyBase.CreatePickList(False) if you want an unrestricted
PickList
        pickList =
MyBase.CreatePickList(True)
        PopulatePickList(pickList)
        Return pickList

    End Function

Public Overrides Sub
PopulatePickList(ByVal pickList As DynamicPickList)
    pickList.Add(DBNull.Value)
    pickList.AddRange("A|B|
C".Split("|"c))

    End Sub
```

Example 4: Default Value Rule

Consider an example where the Recommendation family contains the field Final Approver Name (MI_REC_FINAL_APPROVE_NAME_C). Suppose you want to create a Default Value rule on the Final Approver Name field to set it by default to the name of the user who is logged in to APM at the time the record is approved. The resulting code would look like this:

```
Public Overrides Function
GetDefaultInitialValue() As Object
    Return
RecommendationUtilities.RecommendationDefaultValues.Final
ApproverName(Current Entity)

    End Function
```

Example 5: Disabled Rule

It may be appropriate to disable some fields conditionally, based on the value in another field.

Consider an example where the Equipment family contains the fields Plant Section (MI_EQUIP000_PLANT_SECTION_C) and Person Responsible for Plant Section (MI_EQUIP000_PLANT_SECT_DESC_C). The Person Responsible for Plant Section field does not need to be enabled until the Plant Section field contains a value.

To enforce this logic, you could create the following rule on the Person Responsible for Plant Section field so that it is disabled when the Plant Section field is empty. The resulting code would look like this:

```
Public Overrides Function
IsDisabled() As Boolean
    If
    Convert.IsDBNull(CurrentEntity.Fields("MI_EQUIP000_PLANT_
SECTION_C").Value) Or
Object.Equals(CurrentEntity.Fields("MI_EQUIP000_PLANT_SEC
TION_C").Value, "") Then
        Return True
    Else
        Return False
    End If
End Function
```

Example 6: Format Rule

You may want to format character fields so that the value typed in the field appears in all capital letters. Consider an example where the Recommendation family contains the field Completion Comments (MI_REC_CLOSE_COMME_TX), and you want to view values in that field in all capital letters. You could create the following Format rule on the Completion Comments field to do so:

```
Public Overrides Function
FormatValue(ByVal value As Object) As String
    If Not
    Convert.IsDBNull(CurrentEntity.Fields("MI_REC_CLOSE_COMME
_TX").Value) Then
        Return UCase(CStr(value))
    Else
        Return ""
    End If
End Function
```

Note: Like all Format rules, the rule in Example 6 affects only the displayed value in places where Format rules are supported. The value will be stored in the database exactly as it is entered.

Example 7: Formula Rule

For some fields, rather than requiring users to specify a value, you may want to populate the field with a value that is calculated from values entered in other fields. Consider an example where the Work History family contains three cost fields:

- **Maintenance Cost (MI_EVWKHIST_MAINT_CST_N):** The cost to date of maintenance on the asset.
- **Production Cost (MI_EVWKHIST_PRDN_CST_N):** The cost to date to keep the asset in production.
- **Total Cost (MI_EVWKHIST_TOTL_CST_N):** The total cost of the work, including the maintenance and production costs.

In this case, you may want to require users to enter values in the Maintenance Cost and Production Cost fields and then calculate the value for the Total Cost field by adding together the values in the Maintenance Cost and Production Cost fields. To implement this functionality, you could create the following Formula rule for the Total Cost field:

```
Public Overrides Function
GetCalculatedValue() As Object
    'Total Cost = Maintenance Cost +
    Production Cost
    Dim maintCost As Double = 0
    Dim prodCost As Double = 0

    If Not
Convert.IsDBNull(CurrentEntity.Fields("MI_EVWKHIST_MAINT_
CST_N").Value) Then
        maintCost =
Convert.ToDouble(Current
Entity.Fields("MI_EVWKHIST_MAINT_CST_N").Value)
    End If

    If Not
Convert.IsDBNull(CurrentEntity.Fields("MI_EVWKHIST_PRDN_C
ST_N").Value) Then
        prodCost =
Convert.ToDouble(CurrentEntity.Fields("MI_EVWKHIST_PRDN_C
ST_N").Value)
    End If

    Return maintCost + prodCost

End Function
```

Note: For this example to work properly, the Total Cost field must be set up as a formula field.