**C# implementation of Publication Package**

Contents

[Change History 3](#_Toc510796951)

[C# Package structure 4](#_Toc510796952)

[C# Code implementation 5](#_Toc510796953)

[1.1 Service/Form VR implementation 5](#_Toc510796954)

[1.2 Cross Form VR Implementation 5](#_Toc510796955)

# Change History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Revision Date | Author | Summary Of Change |
| 1.0 | 03/04/2018 | ATO | Tax Time 2018 code publication package implementation Overview. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Document explains high level C# code implementation for the validation rules and cross form rules.

# C# Package structure

C# publication package has the following folder structure.

… **<ServiceName>2018**

…… Properties

…… References

…… Taxonomy

…… Validation

………..CrossForm

…………….ChildReference

…… Ato.CD.Inbound.<serviceName>.csproj

…… Ato.CD.Inbound.<serviceName>.sln

…… ReadMe.txt

**Example: NITR service TRT**

… **TRT2018**

…… Properties

…… References

…… Taxonomy

…… Validation

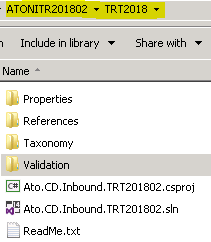
………..CrossForm

…………….ChildReference

…… Ato.CD.Inbound.TRT201802.csproj

…… Ato.CD.Inbound.TRT201802.sln

…… ReadMe.txt



**Properties**: Visual studio property file

**References**: all the class files required for solution build

**Taxonomy**: Taxonomy files (for parent service, we will have parent and its schedules taxonomy files. For schedule service, we will have only schedule service taxonomy)

**Validation**: ESR generated C# Validation code files

**CrossForm**: C# CrossForm rule files for the service. Schedule/Child service will not have CrossForm folder.

**ChildReference**: schedules service consumer and entity class files for service.

**Ato.CD.Inbound.TRT201802.csproj**: .net project file

**Ato.CD.Inbound.TRT201802.sln**: .net solution file

**ReadMe.txt**: Read me file for the solution compile and build.

# C# Code implementation

## Service/Form VR implementation

Package contains following C# code files used to implement the Service/form validation rules.

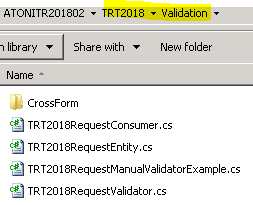
<**Service**>RequestConsumer.cs

<**Service**>RequestEntity.cs

<**Service**>RequestManualValidatorExample.cs (optional if there are no manual rules)

<**Service**>RequestValidator.cs

**Example**: TRT 2018 service



## Cross Form VR Implementation

Package contains following C# code files used to implement the Cross Form validation rules.

**CrossForm**

…..ChildReference (Folder)

….. CrossFormExtensions.cs

….. CrossFormLocator.cs

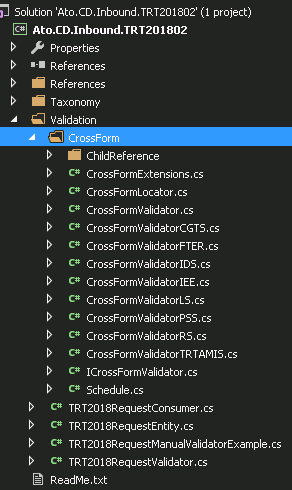
….. CrossFormValidator.cs

….. CrossFormValidator<childform>.cs

….. ICrossFormValidator.cs

….. Schedule.cs

**Example**: TRT CrossForm C# Validation



**Schedule.cs:** The Schedule class contains the names of all schedules associated with this service (TRT).

**CrossFormExtensions.cs:** Certain CrossForm rules require access to a schedule’s contexts. The CrossFormExtensions class acts as facilitator for building up a List of contexts. Each context added to the list of contexts is a struct. Structs are value types introducing less overhead than classes. The CrossFormExtensions class exposes a single Extension Method that can be consumed in a fluent manner.

**ChildReference (Folder)**: Contains consumer /entity and context classes associated with Cross Form validation.

**CrossFormLocator.cs**: The CrossFormLocator class is a Service Locator implementation. It allows for dynamic type resolution and polymorphic invocation of CrossForm validation by calling the ValidateCrossFormRules method on ICrossFormValidator implementers.

**Example**: TRT has CGTS, LS, PSS, FTER, IEE, RS, TRTAMIS and IDS as schedules



**CrossFormValidator<childform>.cs:** ContainsCrossForm validation rules associated with each specific schedule (CGTS, LS, PSS, FTER, IEE, RS, TRTAMIS, IDS). Each CrossFormValidator<childform> class is required to implement the *ICrossFormValidator* interface**.**

**ICrossFormValidator.cs**: The *ICrossFormValidator* Interface acts as a contract, which is required to be implemented by all schedule CrossForm Validators (CrossFormValidator<childform> classes). Having this interface in place allows for dynamic type resolution and polymorphic invocation of CrossForm validation rules. It further facilities implementation of the Service Locator.

**CrossFormValidator.cs:** This class is not an*ICrossFormValidator* implementerand nor is it part of the Service Locator. CrossForm VRs that require access across multiple schedules, or where COUNT(SCHEDULE) = 0, must be implemented in the CrossFormValidator class. Example of such rules are shown below

1. Example of VR requiring access across multiple schedules

VR.ATO.TRT.432044

IF (COUNT(SCHEDULE = "PSS") > 0) AND ([TRT49] <> SUM([PSS18]))

RETURN VALIDATION MESSAGE

ENDIF

1. Example of VR checking (COUNT(SCHEDULE) = 0)

VR.ATO.TRT.432042

IF [TRT133] > 10000 AND [TRT44] <> TRUE AND (COUNT(SCHEDULE = "CGTS") = 0)

    RETURN VALIDATION MESSAGE

ENDIF

**Example of the TRT parent form implementing Cross Form C# validation**

static List<ProcessMessageDocument> ValidateAssociatedDocuments(BusinessDocument crossFormDocument)

{

// contains error messages returned by each CrossForm validator

var returnMessages = new List<ProcessMessageDocument>();

// Note: that CrossForm C# Validation requires that the ConsumedReport property

// have been set for the parent and all its attached schedules

TRT2018 consumedReport = (TRT2018)crossFormDocument.ConsumedReport;

// Check if any schedules are attached to the parent form

if (crossFormDocument.ChildDocuments != null)

{

// foreach schedule…

     foreach (BusinessDocument childDocument in crossFormDocument.ChildDocuments)

     {

// locate the CrossForm Validator class for this schedule.

// (uses the Service Locator to dynamically load each schedule validator)

         ICrossFormValidator iCrossFormvalidator = CrossFormLocator.LocateCrossFormValidatorFor(childDocument, consumedReport);

// polymorphically execute the CrossForm Validator for this schedule and append return messages, if any

         returnMessages.AddRange(iCrossFormvalidator.ValidateCrossFormRules());

}

}

// The CrossFormValidator class contains CrossForm VRs that require access

// across multiple schedules, or where COUNT(SCHEDULE) = 0

// instantiate an instance of the CrossFormValidator class

CrossFormValidator crossFormValidator = new CrossFormValidator(consumedReport, crossFormDocument.ChildDocuments);

// execute the VRs located in the CrossFormValidator class and append return messages, if any

returnMessages.AddRange(crossFormvalidator.ValidateCrossFormRules());

// return List of error messages

return returnMessages;

}