RELATIONS,

RULES

AND

RULEBOOKS

Contents

[Introduction 4](#_Toc189972261)

[Association 5](#_Toc189972262)

[Datatype Properties 6](#_Toc189972263)

[location 6](#_Toc189972264)

[evalCtl 6](#_Toc189972265)

[execCtl 6](#_Toc189972266)

[execAssoc 6](#_Toc189972267)

[assoc 6](#_Toc189972268)

[eval () 7](#_Toc189972269)

[check () 7](#_Toc189972270)

[exec Main() 7](#_Toc189972271)

[exec () 7](#_Toc189972272)

[append(obj, [afterObj]) 7](#_Toc189972273)

[hasAssoc(obj) 7](#_Toc189972274)

[length() 7](#_Toc189972275)

[prepend(obj, [beforeObj]) 7](#_Toc189972276)

[removeAssoc(obj) 8](#_Toc189972277)

[Association Classes 9](#_Toc189972278)

[Syntax 9](#_Toc189972279)

[ADV3 Rulebook Library Extension 11](#_Toc189972280)

[Predefined Rules 11](#_Toc189972281)

[GplayerCharRule 11](#_Toc189972282)

[GactorRule 11](#_Toc189972283)

[GplayerCharLocRule 11](#_Toc189972284)

[GactorLocRule 11](#_Toc189972285)

[GactionRule 11](#_Toc189972286)

[GdobjRule 11](#_Toc189972287)

[GiobjRule 11](#_Toc189972288)

[GctRule 12](#_Toc189972289)

[Predefined Rulebooks 13](#_Toc189972290)

[InsteadOfActionRulebook 13](#_Toc189972291)

[BeforeActionRulebook 13](#_Toc189972292)

[CheckActionRulebook 13](#_Toc189972293)

[InsteadOfExecActionRulebook 13](#_Toc189972294)

[ExecActionRulebook 13](#_Toc189972295)

[AfterActionRulebook 13](#_Toc189972296)

[The Action doActionOnce() Hook 14](#_Toc189972297)

[Examples of ADV3 Rules and Rulebooks 15](#_Toc189972298)

# Introduction

The Tads 3 Rulebooks library extension is meant to help an author facilitate complex relationships in a rule-driven manner.

# Association

The basic datatype is the Association. This class is subclassed into the following:

* *Relations* – these classes evaluate all of their associations then invoke their exec() method.
* *Rules* – these classes must have all of their associations evaluate either true or nil before they will invoke their exec() method.
* *Rulebooks* – these classes must have one of their associations evaluate either true or nil before they will invoke their exec() method.

In addition, each of these classes is subclassed based on what sort of evaluation return the association will send after its exec() method has been invoked:

* *true* – regardless of the return from exec() the class will evaluate true.
* *nil* – regardless of the return from exec() the class will evaluate nil.
* *value* – the class will evaluate with the value returned from its exec() method.

## Datatype Properties

The datatypes encapsulate the following properties:

### location

If non-nil this points to the object that considers this object its association.

### evalCtl

Indicates how processing of associations is to be handled by the object. The valid values are:

* EvalCtlDoAll – all associations are to be processed and exec() is to be invoked.
* EvalCtlAllNil – In order for exec() to be invoked all of the associations must evaluate nil. If any association does not do so then evaluation of associations stops and exec() is not invoked.
* EvalCtlAllTrue – In order for exec() to be invoked all of the associations must evaluate true. If any association does not do so then evaluation of associations stops and exec() is not invoked.
* EvalCtlExistNil – In order for exec() to be invoked one of the associations must evaluate nil. If none of the associations does so then exec() is not invoked.
* EvalCtlExistTrue – In order for exec() to be invoked one of the associations must evaluate true. If none of the associations does so then exec() is not invoked.

### execCtl

Indicates how evaluation of this object is to be handled once exec() is to be invoked. The valid values are:

* ExecCtlRetVall – return to the caller the return value of the exec() invocation.
* ExecCtlRetNil – return nil to the caller.
* ExecCtlRetTrue – return true to the caller.

### execAssoc

Indicates the association to be evaluated when exec() is invoked. If Association’s default exec() method has been overridden and not inherited then this property will be ignored. This is equivalent to coding the following:

exec() { return (execAssoc.eval(); }

### assoc

This property is a list of the datatype’s associations. Depending on the datatype’s class this list may represent relationships, prerequisites, or members. The result of evaluation of these associations determines whether the datatype’s exec() method is invoked.

### eval ()

This method is meant to be called when an author desires to evaluate a datatype. Evaluation means that the datatype’s associations will be processed (evaluated), and if indicated, the datatype’s exec() method invoked.

### check ()

This method is executed when the datatype is evaluated, via its eval() method. This method processes the evaluation of the datatype’s associations and returns true or nil to indicate whether the datatype’s exec() method should be invoked.

### **exec** Main()

This method determines what the object will return when it’s exec() method is to be invoked. Based on execCtl, the object may return nil, true, or the return value from exec().

### **exec** ()

This method is invoked only when the datatype’s check() method returns true.

Additionally, the datatypes have the following methods for use in verifying, adding, and removing associations.

### append(obj, [afterObj])

Appends the value obj to the datatype’s associations. If afterObj is supplied then obj is appended to the datatype’s associations after afterObj. If afterObj is not found in the datatype’s associations then obj is appended to the end of datatype’s associations.

### hasAssoc(obj)

Returns true if obj is one of the datatype’s associations. Otherwise nil is returned.

### length()

Returns the number of associations for this datatype.

### prepend(obj, [beforeObj])

Prepends the value obj to the datatype’s associations. If beforeObj is supplied then obj is prepended to the datatype’s associations before beforeObj. If beforeObj is not found in the datatype’s associations then obj is prepended to the end of datatype’s associations.

### removeAssoc(obj)

Removes obj from this datatype’s associations. If obj is not one of the datatype’s associations the method does nothing.

## Association Classes

The following table summarizes the characteristics of Association subclasses:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | **Do** **All** | **All** **True** | **All** **Nil** | **Exist****True** | **Exist****Nil** | **Return****Value** | **Return****True** | **Return****Nil** |
| **Relation** | **X** |  |  |  |  |  | **X** |  |
| **RelationDoAllRetVal** | **X** |  |  |  |  | **X** |  |  |
| **RelationDoAllRetTrue** | **X** |  |  |  |  |  | **X** |  |
| **RelationDoAllRetNil** | **X** |  |  |  |  |  |  | **X** |
| **Rule** |  | **X** |  |  |  | **X** |  |  |
| **RuleAllTrueRetVal** |  | **X** |  |  |  | **X** |  |  |
| **RuleAllTrueRetTrue** |  | **X** |  |  |  |  | **X** |  |
| **RuleAllTrueRetNil** |  | **X** |  |  |  |  |  | **X** |
| **RuleAllNilRetVal** |  |  | **X** |  |  | **X** |  |  |
| **RuleAllNilRetTrue** |  |  | **X** |  |  |  | **X** |  |
| **RuleAllNilRetNil** |  |  | **X** |  |  |  |  | **X** |
| **Rulebook** |  |  |  | **X** |  |  | **X** |  |
| **RulebookExistTrueRetVal** |  |  |  | **X** |  | **X** |  |  |
| **RulebookExistTrueRetTrue** |  |  |  | **X** |  |  | **X** |  |
| **RulebookExistTrueRetNil** |  |  |  | **X** |  |  |  | **X** |
| **RulebookExistNilRetVal** |  |  |  |  | **X** | **X** |  |  |
| **RulebookExistNilRetTrue** |  |  |  |  | **X** |  | **X** |  |
| **RulebookExistNilRetNil** |  |  |  |  | **X** |  |  | **X** |

## Syntax

Defining any of the Association subclasses is:

**<assocName1:> AssocSubclass**

 **<location = assocName2>**

 **<execAssoc = assocName3>**

 **<exec() { // do something }>**

**;**

Each object can have any number of associations. These are identified as members of the object’s assoc list. They can be defined by setting their location property to point to the object, or by use of the Tads 3 “+” notation.

An example Rulebook might look something like this:

**Rulebook;**

**+ Rule;** // Notice that this rule has associations.

 **++ Rule;**

 **++ Rule;**

**+ Rule;**

**+ Rule;**

The rulebook will evaluate each rule until one returns true. At that point evaluation of its associations stop and the rulebook will return true to its caller.

Below is an example rule defining an exec() method that returns true. Since this is a rule the return value of its exec() will be returned to its caller.

**Rule**

**{**

**exec()**

**{**

 **// do something**

 **return true;**

**}**

**}**

Below is an example of how to define a rule with “prerequisites” that will return the evaluation of a rulebook. MyRule will first evaluate each of its associations, and if each returns true then it will return the value of the evaluation of myRulebook:

**MyRule: RuleAllTrueRetVal
{**

 **execAssoc = myRulebook**

**}**

**+ Rule;**

**+ Rule;**

**MyRulebook;**

**+ Rule;**

**+ Rule;**

**+ Rule;**

# ADV3 Rulebook Library Extension

The arl.tl will implement a simple override of the ADV3 library’s Action process, allowing an author to incorporate rule handling into the process.

## Predefined Rules

While there are many ways that rules and rulebooks can be coded, the conceptual design behind the Rulebooks library extension has been to make rules as atomic as possible. Rather than coding multiple complex conditions in a single method, the library makes use of “associations”, which are in turn rules or rulebooks. These associations must all be true in order for the rule to execute.

The ADV3 Rulebooks library extension provides a rules.t module that contains some predefined rules that are useful in dealing with the “globals” of the TADS ADV3 library.

In general to use one of these rules, simply define an anonymous instance of the desired rule and assign an appropriate value to its value property.

These are as follows:

### GplayerCharRule

This rule returns true when the player character equals or is of kind value; otherwise it returns nil.

### GactorRule

This rule returns true when the actor equals or is of kind value; otherwise it returns nil.

### GplayerCharLocRule

This rule returns true when the player character location equals or is of kind value; otherwise it returns nil.

### GactorLocRule

This rule returns true when the actor location equals or is of kind value; otherwise it returns nil.

### GactionRule

This rule returns true when the action equals or is of kind value; otherwise it returns nil.

### GdobjRule

This rule returns true when the direct object equals or is of kind value; otherwise it returns nil.

### GiobjRule

This rule returns true when the indirect object equals or is of kind value; otherwise it returns nil.

### GctRule

This rule returns true when the game clock time equals value or is between start and end inclusive; otherwise it returns nil.

## Predefined Rulebooks

The Rulebooks library extension provides a rulebooks.t module that contains some predefined rulebooks and associated rules that are useful in dealing with the “action” phase of the TADS ADV3 library.

In general to use one of these rulebooks, simply include rulebooks.t in your compilation.

These are as follows:

### InsteadOfActionRulebook

This rulebook automatically stores any InsteadOfActionRule rules.

### BeforeActionRulebook

This rulebook automatically stores any BeforeActionRule rules.

### CheckActionRulebook

This rulebook automatically stores any CheckActionRule rules.

### InsteadOfExecActionRulebook

This rulebook automatically stores any InsteadOfExecActionRule rules.

### ExecActionRulebook

This rulebook automatically stores any ExecActionRule rules.

### AfterActionRulebook

This rulebook automatically stores any AfterActionRule rules.

## The Action doActionOnce() Hook

By including the rb\_mods.t file in your compilation the ADV3 Action class’ doActionOnce() method will automatically provide “hooks” into the evaluation of the predefined rulebooks of rulebooks.t.

The behavior of these hooks is consistent across all stages of the Action doActionOnce() phase: the appropriate rulebook is given first shot at handling the phase. If the rulebook evaluates as true then it is assumed that the phase has been handled by the rulebook; otherwise the library’s default handling takes over.

## Examples of ADV3 Rules and Rulebooks

The following examples are part of the sample game source included with the Rulebooks library extension. We explain how some of this code works.

Suppose we wish to do something during a specific period of game clock time. We must first determine at which stage of the Action execution phase we want our rule to have effect. In the example below we choose an InsteadOfAction rule, which means the whole action phase will be bypassed if the InsteadOfActionRulebook returns true. If the rulebook returns nil then library’s default checkRemapping() mechanism takes over.

Next we determine what we want the rule to do by defining an exec() method. In the example below this method will simply display the game clock time.

**InsteadOfActionRule**

**{**

 **exec()**

 **{**

 **"<<msg>>";**

 **return true;**

 **}**

 **msg = 'GCT: ' + Schedulable.gameClockTime + '. '**

**}**

**+ GctRule**

**{**

 **start = 3**

 **end = 5**

**}**

The GctRule definition is our InsteadOfActionRule rule’s association (or prerequisite). We’ve defined this rule using the TADS 3 “+” notation to indicate its associate with the InsteadOfActionRule. We have only to define the game clock start and end values for this rule.

When an action’s doActionOnce() method is called the first stage will be to evaluate the InsteadOfActionRulebook. Each association will be evaluated individually until one of them returns true or the list has been exhausted. When our InsteadOfActionRule is evaluated its associations will be evaluated. Since GctRule has no associations its exec() method will be executed. If the game clock time is between 3 and 5 inclusive it will return true to its caller. Since our InsteadOfActionRule has only one association this will satisfy its prerequisites and it will execute the message displaying the game clock time.

Our next example is a little more complex. In this case we want to prevent our actor from taking the loaf of bread when he is in the kitchen. In this situation we arbitrarily chose the Actor Action stage, using an ActorActionRule.

**ActorActionRule**

**{**

 **exec()**

 **{**

 **"<<msg>>";**

 **return true;**

 **}**

 **msg = '{You/he} reach{es} for the loaf, but think better of it. '**

**}**

The rule only defines the message to be displayed and indicates to the caller that it has handled the process. The associations for this rule include 2 rules and a rulebook:

**+ GactionRule**

**{**

 **value = TakeAction**

**}**

**+ GactorLocRule**

**{**

 **value = kitchen**

**}**

**+ Rulebook**

**;**

For the purpose of this example (and unlike the sample game source) we’ve defined the rulebook anonymously. The rulebook also has two associated rules (indicated by the “++” level notation):

++ GdobjRule

{

 value = Chair

}

++ GdobjRule

{

 value = Food

}

So in this particular example the actor would be prevented from taking either a Chair class object or a Food class object while in the kitchen.