|  |  |
| --- | --- |
| Handläggare  Avdelning |  |

# Utvecklarhandledning till VTL-motor

# Introduktion

VTL20Engine är en fristående motor för exekvering av kod skriven i språket VTL så som det definieras i version 2.0. Den är skriven i C# med .net standard 2.0 och använder sig av Antlr 4 runtime. I detta dokument beskrivs VTL-motorns implementation ur ett utvecklarperspektiv. Avsikten med texten är att utvecklare med hjälp av den lättare ska förstå implementation och kunna använda den som referens vid vidareutveckling av VTL-motorn.

# Översikt

Exekveringen av VTL-koden kan delas in i tre steg.

1. Koden parsas med Antlr och ett abstrakt sökträd byggs upp.
2. Det träd som Antlr byggt upp traverseras med hjälp av en implementation av visitormönstret. Varje nod i trädet besöks och en beräkningskedja av operatorobjekt byggs upp.
3. Beräkningen utförs genom att exekveringsflödet rullar igenom beräkningskedjan. Endast de noder som behövs för att ta fram det önskade resultatet exekveras.

# Parsning

Parsningen av VTL-koden utförs med hjälp av tredjepartskomponenten Antlr4. Med hjälp av språkdefinitionsfilerna vtl.g4 och vtlTokens.g4 som sammanställts av Eurostat (VTL SDMX task force) genererar Antlr4 kod för parsning av VTL-koden. Denna kod bör genereras om när uppdaterade språkdefinitionsfiler släpps och det görs med scriptet generateCode.bat. Scriptet ligger tillsammans med all automatgenererad och externt skriven kod i katalogen \Parser\Antlr.

Följande kodblock visar hur parsningen i Antlr anropas:

var inputStream = new AntlrInputStream(inputVtlCode);

var lexer = new VtlLexer(inputStream);

var tokens = new CommonTokenStream(lexer);

var parser = new VtlParser(tokens) {BuildParseTree = true};

var context = parser.start();

Först skapas en teckenström. Därefter skapas en lexer som identifierar alla enskilda tecken i strömmen. Sedan grupperas tecknen som nyckelord och symboler, även kallade tokens. Sedan skapas en parser som bygger upp ett träd med relationer mellan alla tokens.

# Beräkningsobjekt

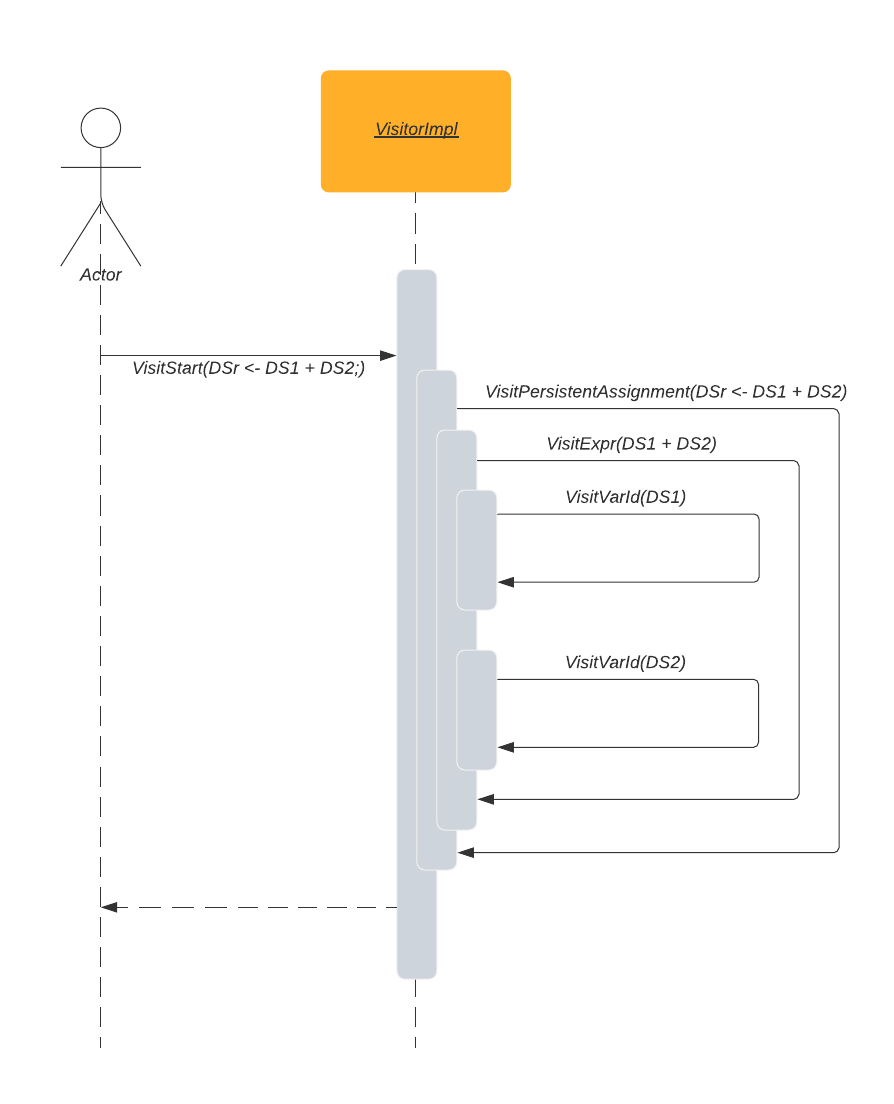
Det träd som Antlr byggt upp traverseras med hjälp av en implementation av visitormönstret. Varje nod i trädet besöks och en beräkningskedja av operandobjekt byggs upp. I praktiken har denna samma topologi som det träd som skickades in men det består av objekt av klasser som kan exekveras för att få ett beräkningsresultat. Nedan visas fortsättningen på koden som startartar flödet.

var heap = inputOperands.ToList();

var visitor = new VtlVisitorImpl(heap);

visitor.VisitStart(context);

Visitor.VisitStart besöker startnoden med hela parserträdet som context. Metoden VisitStart kollar vilka noder som finns på nästa nivå i context-trädet och besöker sedan dess visitor-metoder med motsvarande delträd som context. Trädet traverseras genom anrop till nästlade VTL-kommandon. Slutligen kommer trädets löv att initiera anrop till visitor-metoder som hämtar konstanta värden eller värden som ligger lagrade i variabler på heapen. Exemplet i figuren nedan visar principiellt hur uttrycket DSr <- DS1 + DS2; bearbetas.



## Operand

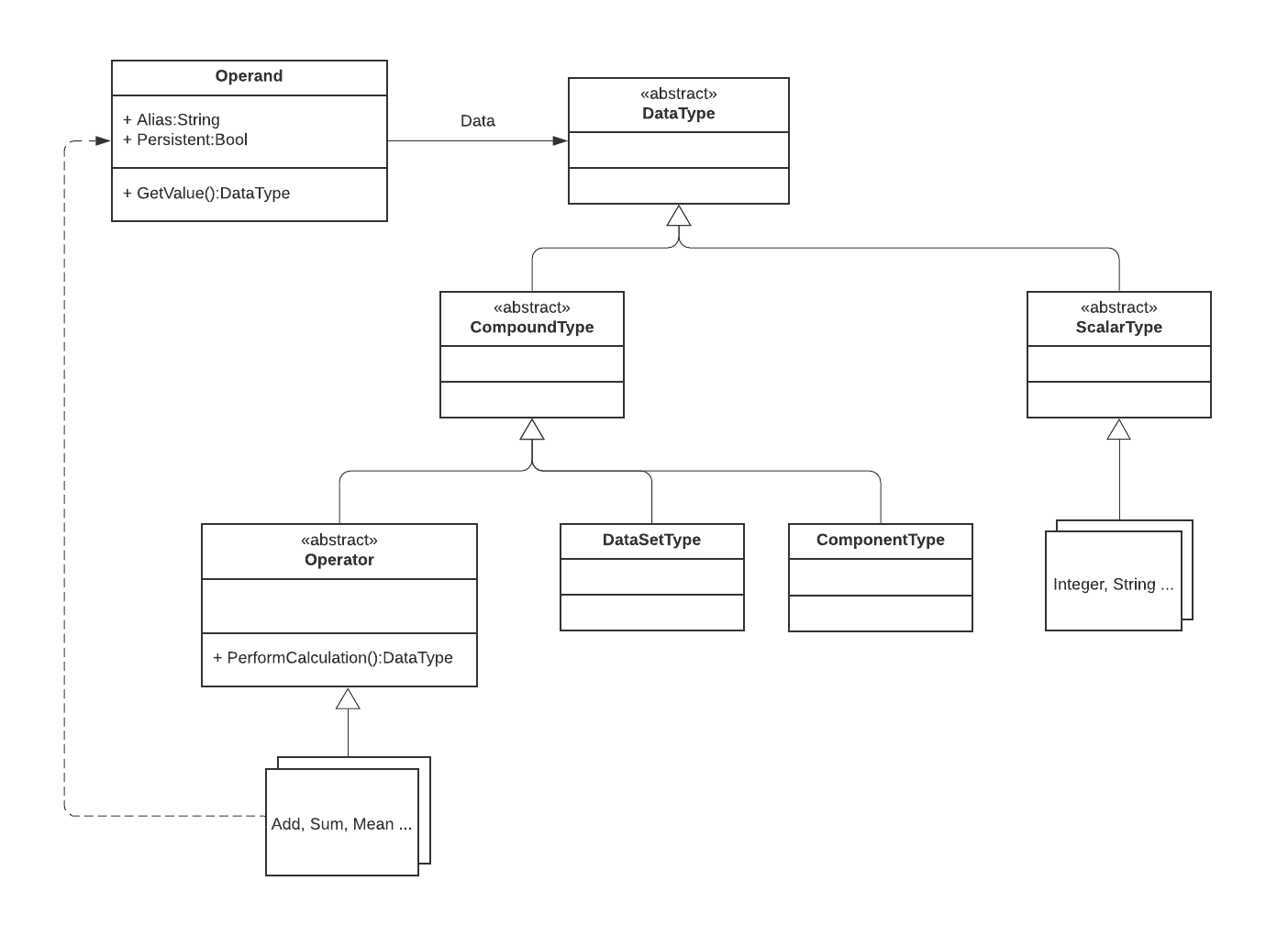
En operand är ett objekt som kan skickas som argument till en VTL-funktion, en operator. Den kapslar in ett dataobjekt och håller även lite metadata som behövs för beräkningskörningen. Operander har tre viktiga egenskaper: Alias är operandens namn och används för att identifiera den. Persistent anger om operanden ska betraktas som ett resultat från VTL-körningen. Om den inte är markerad som persistent är den inte tillgänglig utanför VLT-motorn. Slutligen har Operand egenskapen Data av datatypen DataType. Den kan gestalta sig som en Operator eller som ett fast värde, antingen av enkel typ (heltal, sträng o.s.v.) eller av sammansatt typ (dataset, komponent o.s.v.).

## Operator

En operator utför en specifik funktion, ofta en beräkning. Den returnerar alltid ett dataobjekt (DataType) som resultat men beroende på funktion tar den varierande antal operander och styrparameter som argument.

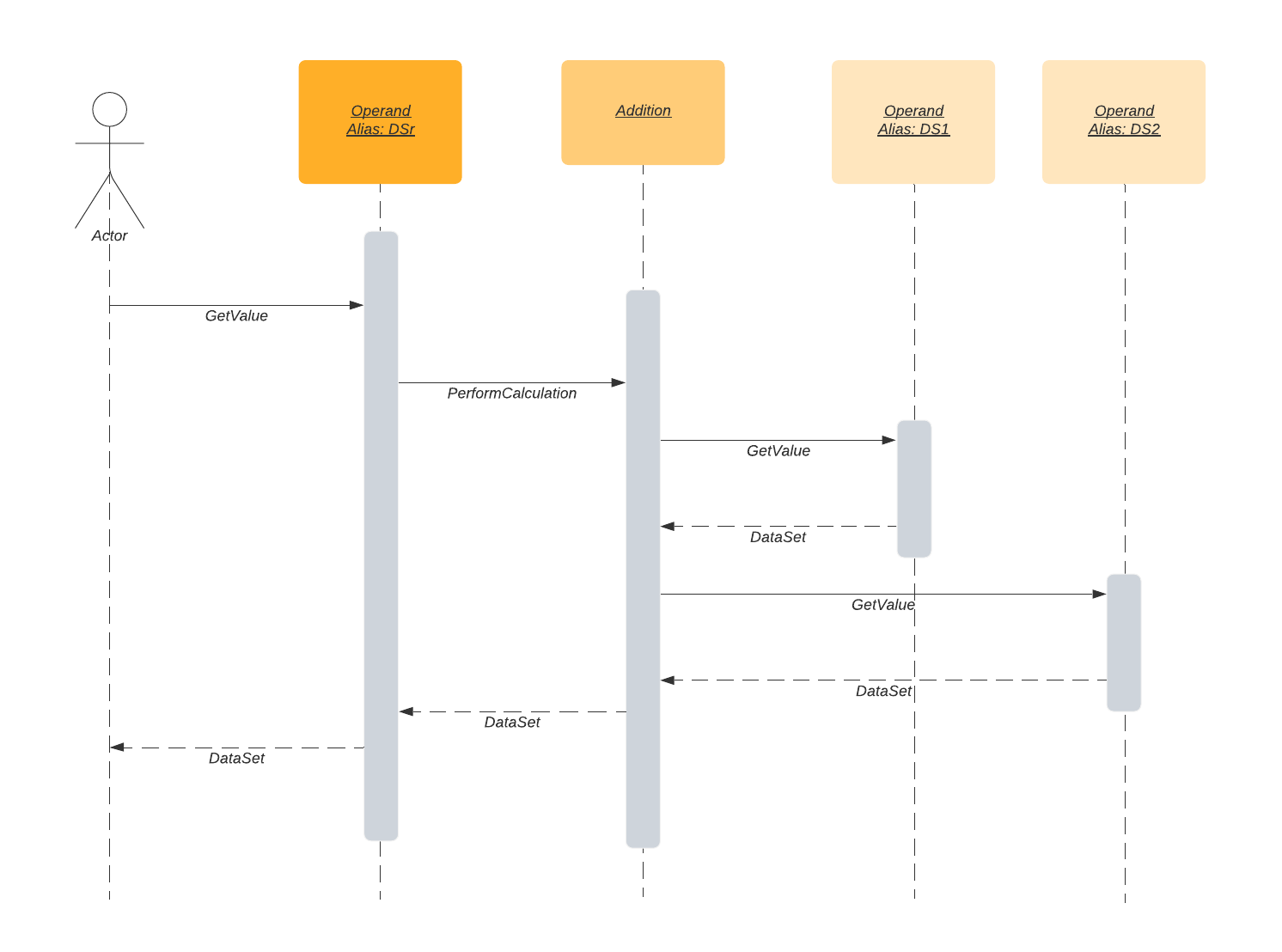
## DataType

DataType är basklass för all datahantering i VTL-motorn. Det möjliggör en väldigt flexibel hantering av nästlade anrop. Argument till en operator kan lika gärna vara andra operatorer som enkla datatyper eller dataset. Denna uppbyggnad är hämtad direkt från VTL dokumentationen, se VTL user manual sid 49. Se även figuren nedan.



# Exekvering

Vid anrop till motorn skickas förutom VTL-koden även en uppsättning namngivna in-parametrar med. De kan vara av skalär typ (t.ex. tal, textsträngar) eller sammansatt typ (t.ex. dataset, komponenter). Dessa kapslas in i operandobjekt, kompletteras med alias och läggs i en lista kallad heap. Från denna lista nås inparametrar och delresultat under beräkningens exekvering.

Beräkningen initieras genom att resultatet för en av beräkningskedjans noder efterfrågas. Alla noder utgörs av operander. Den efterfrågade noden begär sina parametrar som vid nästlade beräkningar också är operander. Denna begäran görs med metoden GetValue(). Om datat i operanden är en operator så anropas operatorns PerformCalculation-metod och är det en databärande typ så returneras helt enkelt datat. I figuren nedan visas ett sammanfattat händelseföljdsdiagram för exekveringen av VTL-koden DSr <- DS1 + DS2;

## Operatorer

VTL definierar många operatorer och för att hålla ordning på dem är de sorterade i olika kategorier. Varje operator är implementerad i en klass med undantag i vissa komplexa fall där hjälpklasser är nödvändiga. I utvecklingsprojektets katalogstruktur är dessa klasser grupperade på samma sätt som operatorerna är indelade i kapitel i VTL dokumentet ”Library of Operators”. Många operatorer har likartade beteenden så för att underlätta underhåll och vidareutveckling samt för att undvika duplicerad kod används arv frikostigt i implementationen av operatorerna.