Description

This section contains example OCL rules. They are copied from internal material originally used by XMLdation's team while learning OCL. The rules here tend to be the most complicated and longest to write, which is why they were put into a single document in the first place, so don't worry if the rule are hard to understand or seem complicated.

The purpose is to give the reader an idea how certain types of rules are written in OCL. Edits to rules are most likely needed when copying them to other situations.

List of rules

--Control (GrpHdr) sum check. Please note that using types inside code ignores context otherwise.
--This is why this check can only be used in Group Lvl, as it takes every Amt in the file

```
self.CtrlSum->size() = 1 implies
self.CtrlSum.oclAsType(decimal) =
(
  CreditTransferTransactionInformation10.allInstances()->Amt.InstdAmt->sum().oclAsType(decimal) +
  CreditTransferTransactionInformation10.allInstances()->Amt.EqvtAmt.Amt->sum().oclAsType(decimal)
)
```

--Sumcheck for 2nd lvl

```
self.CtrlSum->size() = 1 implies self.CtrlSum.oclAsType(decimal) =
self.CdtTrfTxInf.Amt.InstdAmt->sum().oclAsType(decimal) +
self.CdtTrfTxInf.Amt.EqvtAmt.Amt->sum().oclAsType(decimal)
```

--Element is mandatory

```
self.ChrgBr->size() = 1
```

--Element length

```
self.Cdtr.Nm->size() = 1 implies  self.Cdtr.Nm.size() <= 70
```

--Either ‘Structured’ or ‘Unstructured’ may be present.

```
not( self.Strd->size()= 1 and self.Ustrd->size()= 1 )
```

--Or

```
self.Strd->size()= 1 implies self.Ustrd->size()= 0
```

--Or

```
self.Strd->size()= 1 and self.Ustrd->size()= 1 implies false
```

-- Amount between certain values

```
self.Amt.InstdAmt > 0.00 and self.Amt.InstdAmt < 1000000000.00
```

-- Element value if given

```
self.CdtrRefInf.Tp->size() = 1 implies self.CdtrRefInf.Tp.CdOrPrtry.Cd = "SCOR"
```

-- Regular expression example
```
self.CdtrSchmeId.Id.PrvtId.Othr->size() >= 1 implies
  self.CdtrSchmeId.Id.PrvtId.Othr->forAll(q | q.Id.matches('[A-Z][A-Z][0-9][0-9][A-Za-z0-9]+'))
```

-- Checking regular expressions inside a repetitive element (Othr is [1..n])...
```
self.UltmtDbtr.PstlAdr.Ctry = "IT" implies
self.UltmtDbtr.Id.OrgId.Othr->forAll(o |
o.Id.matches('^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$') implies
o.Issr = "ADE"
)
```

-- ...and an appropriate query to pinpoint the error message to a specific place inside the repetitive element.
```
self.UltmtDbtr.Id.OrgId.Othr->select(o |
o.Id.matches('^[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$') and
o.Issr <> "ADE"
).Issr
```

-- Basic latin characters
```
self.matches("[a-zA-Z0-9/\-?:().,'+ =!\x22 %&\x2A <>;@#${}]*\r\n\x5B \x5D \x5C _\x5E \x60 |\x7E ")
```

--'Escaping any character
```
self.MsgId.matches("\u94F6") implies false
```

--More information about characters:
--http://www.fileformat.info/info/unic...94F6/index.htm (lists C/C++/Java source codes)
--http://codepoints.net/U+94F6 (general information about a character)
--http://hexed.it/ (shows the encoding of inputted file)
--http://webdesign.about.com/od/locali...odes-ascii.htm

-- Limiting fractional part of number (Causes validation to fail if "let s :" doesn't find anything
-- was .oclAsType(decimal).toString(). Was updated some time ago so casting to integer not needed anymore. if it exists somewhere, will cause building to fail
```
self.Amt.InstdAmt->size() = 1 implies (
  let s : string = self.Amt.InstdAmt.toString()
  in
  s.contains('.') implies
  (s.indexOf('.') = s.size() - 3 or
   s.indexOf('.') = s.size() - 2)
)
```

-- also note:
```
s.indexOf('.') > (s.size()-4)
```
-- if currency amt can be ended with .

-- All the Ids (if given) must be same

```
DirectDebitTransactionInformation9.allInstances()->DrctDbtTx.CdtrSchmeId.Id.PrvtId.Othr.Id->asSet()->size() <= 1
```

-- Same written differently

```
self.PmtInf->collect(a | a. PmtTpInf.SeqTp)->asBag()->size() > 1 implies (
self.PmtInf->collect(a | a. PmtTpInf.SeqTp)->asBag()->size() <>
self.PmtInf->collect(a | a. PmtTpInf.SeqTp)->asSet()->size()
)
```

--No line feeds

```
self.Ustrd->size() > 0 implies not(self.Ustrd->exists(q | q.matches("(.*[\r|\n|\r\n].*)+")))
    --Query
    self.Ustrd->select(q | q.matches("(.*[\r|\n|\r\n].*)+"))
```

--All Ids are unique

```
self->allInstances()->collect(a | a. PmtInfId)->asBag()->size() =
self->allInstances()->collect(a | a. PmtInfId)->asSet()->size()
```

--This would give error message to the beginning of every context.

--All the substrings of Ids up until a fixed string are unique

```
AttachmentDetailsType.allInstances().AttachmentIdentifier->collect(a | a.substring(1,
a.indexOf('::attachment')))->asBag()->size() =
AttachmentDetailsType.allInstances().AttachmentIdentifier->collect(a | a.substring(1,
a.indexOf('::attachment')))->asSet()->size()
```

--Unique id's used

```
self.allInstances()->collect(q | q.PmtId.EndToEndId)->asBag()->size() =
self.allInstances()->collect(q | q.PmtId.EndToEndId)->asSet()->size()
```

--Checks that InstrId is not larger than the total occurrence amount of InstrId. Checks that the value one exists in InstrId

--I.E. Values of InstrId are from 1 to size(), in whatever order

```
self.CdtTrfTxInf.PmtId.InstrId->forAll(q |
q.oclAsType(Integer) <= self.CdtTrfTxInf.PmtId.InstrId->size()and
q.isNumeric() ) and
self.CdtTrfTxInf.PmtId.InstrId->exists(q |
q.oclAsType(Integer) = 1
)
```

--Values of InstrId are from 1 to size(). In that order. Also no duplicates allowed.

```
(self.CdtTrfTxInf.PmtId.InstrId->asBag()->size() = self.CdtTrfTxInf.PmtId.InstrId->asSet()->size()) and
self.CdtTrfTxInf.PmtId.InstrId->asSequence()->forAll(q |
q.oclAsType(Integer) = self.CdtTrfTxInf.PmtId.InstrId->asSequence()->indexOf(q)
```

)

--All ids are unique, with escaping of certain values
self.PmtInf.CdtTrfTxInf->collect(q | q.PmtId.EndToEndId)->asBag()->select(e |e <> "NOTPROVIDED")->size()
=
self.PmtInf.CdtTrfTxInf->collect(q | q.PmtId.EndToEndId)->asSet()->select(e |e <> "NOTPROVIDED")->size()

--Number of transactions
self.GrpHdr.NbOfTxs.oclAsType(integer) =
PaymentInstructionInformation3.allInstances()->CdtTrfTxInf->size().oclAsType(integer)

--May not contain å, ä, ö
self.MsgId.matches("[a-zA-Z0-9?\-/().,+{}:]+")

--EPC characters (http://wiki.xmldation.com/Support/EPC/Latin_Character_Set)
self.matches("[a-zA-Z0-9/\-?:().,\x27 +]*")

--May not contain only space, line feed, carriage return, tab (has to contain information)
self.Mndt.MndtReqId.matches('^[\n\r\t]*$') implies false

--Only cyrillic and latin chars and size (http://www.regular-expressions.info/unicode.html)
self.RmtInf.Ustrd->forAll(q | q.size() <= 25 and
q.matches("[\p{InBasic_Latin}\p{InCyrillic}]*"))

--Schema location
schemaLocation = 'urn:iso:std:iso:20022:tech:xsd:pain.001.001.03 pain.001.001.03.xsd'

--IBAN and BIC match example (only in countries where they can be checked)
(self.CdtrAcct.Id.IBAN.matches('NL.*') and self.CdtrAgt.FinInstnId.BIC->size() = 1) implies
CdtrAcct.Id.IBAN.substring(4,8) = CdtrAgt.FinInstnId.BIC.substring(0,4)

(self.DbtrAcct.Id.IBAN.matches('NL.*') and self.DbtrAgt.FinInstnId.BIC->size() = 1) implies
DbtrAcct.Id.IBAN.substring(4,8) = DbtrAgt.FinInstnId.BIC.substring(0,4)

-- Countries in IBAN and BIC have to match
CdtrAcct.Id.IBAN.substring(0,2) = CdtrAgt.FinInstnId.BIC.substring(4,6)

--InstdAmt matches with values in RmtInf
self.RmtInf.Strd->size() >= 2
implies
(
self.Amt.InstdAmt->sum().oclAsType(decimal) = (

```
   self.RmtInf.Strd->collect( q | q.RfrdDocAmt.RmtdAmt )->asBag()->sum().oclAsType(decimal) -
   self.RmtInf.Strd->collect( q | q.RfrdDocAmt.CdtNoteAmt )->asBag()->sum().oclAsType(decimal)
  )
)
```

--Limiting multiple instances of element into certain total length

```
self.AdrLine->collect(a| a.size())->sum() <= 140
```

--Comparing two ints

```
self.AIBTrailerRow.TotalNumber = self.AIBDetailRow->collect(q | q->size())->sum()
```

--Only 140 characters of Strd allowed in finnish, non-aos2, non taxs payments

```
(
self.CdtrAcct.Id.IBAN->size() = 1 and
self.CdtrAcct.Id.IBAN.matches('[F][I].*') and
self.RmtInf.Strd->size() = 1 and
self.PmtTpInf.CtgyPurp <> "TAXS"
)
implies
self.RmtInf.Strd->forAll(s | s.xmlElementBlockSize() <= 140)
```

--Finding a specific occurrence and limiting it to a value

```
self.InitgPty.Id.OrgId.Othr.Issr->asSequence()->at(0) = "CBI"
```

--Query

```
self.RmtInf.Strd->select(q | q.xmlElementBlockSize() > 140)
```

--Summing multiple child elements

```
let s : AT_PostalAddress6 = self.UltmtDbtr.PstlAdr in
s.hasValue() and s.AdrLine->size() = 0 implies (
s.AdrTp.size() +
s.Dept.size() +
s.SubDept.size() +
s.StrtNm.size() +
s.BldgNb.size() +
s.PstCd.size() +
s.TwnNm.size() +
s.CtrySubDvsn.size() +
s.Ctry.size()  < 140
)
```

--IBAN and bic match for finland example

```
((self.DbtrAgt.FinInstnId.BIC = "AABAFI22" and self.DbtrAcct.Id.IBAN->size() = 1 )implies
```

```
self.DbtrAcct.Id.IBAN.matches('FI\d\d6[0-9]+'))
```

-- Query for targeting all duplicate EndToEndId's

```
let allIds : Sequence(String) = self.PmtId.EndToEndId->asSequence() in
                            allIds->asBag()->iterate(a : String; acc : Set(String) = Set{} |
                           if(allIds->count(a) > 1) then
                             acc->including(a.toString())
                           else
                             acc
                           endif)
```

-- -- or

```
let allIds : Sequence(Max35Text) = self.PmtId.EndToEndId->asSequence() in
                           allIds->asBag()->iterate(a : Max35Text; acc : Set(Max35Text) = Set{} |
                           if(allIds->count(a) > 1) then
                             acc->including(a)
                           else
                             acc
                           endif)
```

--Clearing has to match with currency example

```
(self.CdtrAgt.FinInstnId.CmbndId.ClrSysMmbId.Id.matches('AUBSB.*') or
 self.CdtrAgt.FinInstnId.CmbndId.ClrSysMmbId.Id.matches('//AU.*')
implies
  self.Amt.InstdAmt.Ccy = "AUD")
```

--Locating specific occurrence

```
self.InitgPty.Id.OrgId.Othr->asSequence()->at(0).Issr->size() = 1
implies
self.InitgPty.Id.OrgId.Othr->asSequence()->at(0).Issr = "CBI"
--And same for query
self.InitgPty.Id.OrgId.Othr->asSequence()->at(0).Issr
```

--2 queries inside one. This returns the locations of both Ctry and PstlAdr
--So both are shown if both exist. Only PstlAdr shown if Ctry doesn't exist

```
self.FwdgAgt.BrnchId.PstlAdr.Ctry->union(self.FwdgAgt.BrnchId.PstlAdr)
```

-- Match 13 numbers, 10 numbers and 1 decimal or 10 numbers and 2 decimals.
-- Basically just always check that 10 first are numbers and user OR for the three alternative endings

```
let s : string = self.TransactionAmount.oclAsType(decimal).toString()
in
s.matches('[0-9]{10}(([0-9]{3})|([0-9]{1}\.[0-9]{1})|(\.[0-9]{2}))')
-- Matches patterns:
```

```
-- 0123456789012
-- 01234567890.1
-- 0123456789.12
```

--Sum length of elements with multiple occurrence (+ element with single occurrence in this case)

```
self.Dbtr.PstlAdr.AdrLine->collect(q | q.size() + self.Dbtr.PstlAdr.Ctry.size() )->sum() <= 70
```

-- Modulus 97 check example. Prequisites have to be taken care of with other rules. Done with two rules here when result 0 defaulted to 97

```
self.CdtrRefInf.Tp.Issr  = "BBA" and
self.CdtrRefInf.Ref->size() = 1  and
self.CdtrRefInf.Ref.size() = 12 and
self.CdtrRefInf.Ref.matches('[0-9]+') and
self.CdtrRefInf.Ref.substring(0,10).toInteger() mod 97 = 0
implies (
 self.CdtrRefInf.Ref.substring(10,12).toInteger()
 =
 97 )

self.CdtrRefInf.Tp.Issr  = "BBA" and
self.CdtrRefInf.Ref->size() = 1  and
self.CdtrRefInf.Ref.size() = 12 and
self.CdtrRefInf.Ref.matches('[0-9]+') and
self.CdtrRefInf.Ref.substring(0,10).toInteger() mod 97 <> 0
implies (
 self.CdtrRefInf.Ref.substring(0,10).toInteger() mod 97
 =
 self.CdtrRefInf.Ref.substring(10,12).toInteger()
 )
```

--isValidRF()

```
self.Strd->forAll(s |
  s.CdtrRefInf.Ref.matches('RF[A-Za-z0-9]+') implies
  s.CdtrRefInf.Ref.isValidRF()
)
```

--isValidFinnish reference

--Triggered when ref starts with number or when SCOR and no Issr is given. (Issr ISO triggers RF Creditor Reference)

```
((self.CdtrRefInf.CdtrRef->size() = 1 and self.CdtrRefInf.CdtrRef.matches('\d.*') ) or
(self.CdtrRefInf.CdtrRefTp.Issr->size() = 0 and self.CdtrRefInf.CdtrRefTp.Cd = "SCOR")
)
```

implies (self.CdtrRefInf.CdtrRef.isValidReference("FI"))

--isValidIBAN()
self.IBAN->size() = 1 implies self.IBAN.isValidIBAN()

--allowedDaysInPast() / future
ReqdExctnDt.allowedDaysInPast(5)

--Date between certain range (+5 and +90(yes, +90))
(self.ReqdColltnDt->size() = 1 and
self.PmtTpInf.LclInstrm.Cd->size() = 1 and
self.PmtTpInf.LclInstrm.Cd = 'CORE'
)
implies
self.ReqdColltnDt.allowedDaysInPast(0) and
not(self.ReqdColltnDt.allowedDaysInFuture(5)) and
self.ReqdColltnDt.allowedDaysInFuture(91)

--Date Comparison
self.ReqdExctnDt.after(self.PoolgAdjstmntDt)

--Date within two weeks of another date

self.PoolgAdjstmntDt->size() >= 1 implies self.PoolgAdjstmntDt.before(self.ReqdExctnDt.plusDays(14)) and self.PoolgAdjstmntDt.after(self.ReqdExctnDt.minusDays(14))

--Date exactly after two weeks of another date

self.PoolgAdjstmntDt->size() >= 1 implies self.PoolgAdjstmntDt.after(self.ReqdExctnDt.plusDays(13)) and self.PoolgAdjstmntDt.before(self.ReqdExctnDt.plusDays(15))

--DateTime in following format Usage rule: Only valid ISO Date Time is allowed: YYYY-MM-DDThh:mm:ss
self.CreDtTm.toString().matches("^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}$")

--Alternative way for writing rules, using let, in
self.PmtTpInf.LclInstrm.Cd->size() = 1 implies
self.PmtTpInf.LclInstrm.Cd = 'B2B' or
self.PmtTpInf.LclInstrm.Cd = 'CORE'

- ->

```
let s : string = self.PmtTpInf.LclInstrm.Cd in
s.hasValue() implies (s = 'CORE' or s = 'B2B')
```

also

```
self.CdtrSchmeId.Id.PrvtId.Othr->exists(q | q.Id->size()= 1) and
self.CdtrSchmeId.Id.PrvtId.Othr->exists(q | q.Id.matches('BE.*'))
implies
self.CdtrSchmeId.Id.PrvtId.Othr->forAll(q | q.Id.matches('^..[0-9][0-9].*'))
```

- ->

-- Two lets:
-- Note that part before implies has to refer to actual paths and not the variables. Variables are apparently filled with something
-- even when element doesn't exist. This might be substring related as hasValue() works as it should normally

```
let IBAN : string = DbtrAcct.Id.IBAN.substring(0,2)
let BIC : string = DbtrAgt.FinInstnId.BIC.substring(4,6) in
DbtrAcct.Id.IBAN->size() = 1 and DbtrAgt.FinInstnId.BIC->size() = 1  implies
IBAN = BIC
```

-- let and bag

```
let s : Bag(string) = self.CdtrSchmeId.Id.PrvtId.Othr.Id in
s->exists(q | q.hasValue()) and s->exists(q | q.matches('BE.*')) implies (
s->exists(q | q.matches('^..[0-9][0-9].*') )
)
```

--Queries

```
 self.CdtrSchmeId.Id.PrvtId.Othr->select(o | o.SchmeNm->size() = 1
 and o.SchmeNm.Prtry->size() = 0)
```

-- Query targeting the last element if multiple available:
-- can be used e.g. if only one occurrence allowed.
-- Example here for DDv02 schema. Error message targted to last occurrence of RgltryRptg if there are more than one present.

```
context: DirectDebitTransactionInformation9
rule: self.RgltryRptg->size() < 2
query: self.RgltryRptg->asSequence()->at(self.RgltryRptg->size() - 1)
```

---- NOTE regarding rules triggering from PmtInf and in Tx lvls, affecting Tx. Following does not work properly:

```
self.PmtMtd <> 'CHK'
and
self.CdtTrfTxInf->exists(c | c.Purp.Cd = "TAXS" and c.RmtInf->size() =  1)
implies
self.CdtTrfTxInf->forAll(c | c.RmtInf.Ustrd->size() >= 1)
```

--(singlular occurrence in exists triggering all occurrences with forAll)

-- And neither does the following:

```
self.PmtMtd <> 'CHK'
and
self.CdtTrfTxInf->forAll(c | c.Purp.Cd = "TAXS" and c.RmtInf->size() =  1
implies
 c.RmtInf.Ustrd->size() >= 1)
```

-- This does (checks and implies inside forAll)

```
self.CdtTrfTxInf->forAll(c | self.PmtMtd <> 'CHK'  and c.Purp.Cd = "TAXS" and c.RmtInf->size() =  1
implies
 c.RmtInf.Ustrd->size() >= 1)
```

-- No accented chars allowed

```
self.matches('[^àèìòùÀÈÌÒÙáéíóúýÁÉÍÓÚÝâêîôûÂÊÎÔÛãñõÃÑÕäëïöüÿÄËÏÖÜŸçÇßØøÅåÆæœß?]*')
```

-- implies

```
 (
 self.InvoiceTotalVatIncludedAmount->size() > 0 and
 self.InvoiceTotalVatExcludedAmount->size() > 0 and
 self.InvoiceTotalVatAmount->size() > 0
 )
 implies
 (
 self.InvoiceTotalVatIncludedAmount.toReal() =
 self.InvoiceTotalVatExcludedAmount.toReal() + self.InvoiceTotalVatAmount.toReal()
```

-- Invalid characters regex example

```
self.matches('[^ääçéíóöúüýæøåÄÄÇÐÉÍÓÖÚÜÝÆØÅß%&=@§]*')
```

-- also ((?s). makes dot accept spaces)

```
self.matches('(?s).*[ääçéíóöúüýæøåÄÄÇÐÉÍÓÖÚÜÝÆØÅß%&=@§]+.*') implies false
```

)

--Counts the data within complexType

```
self.CdtrRefInf.xmlDataSize() <= 15
```

--SEPA countries 3.7.2015. Canary islands (ES) and Croatia (HR) were added

```
self->size() = 1 implies (
self.matches('FI.+') or
self.matches('AT.+') or
self.matches('PT.+') or
self.matches('BE.+') or
self.matches('BG.+') or
self.matches('ES.+') or
self.matches('HR.+') or
self.matches('CY.+') or
self.matches('CZ.+') or
self.matches('DK.+') or
self.matches('EE.+') or
self.matches('FR.+') or
self.matches('DE.+') or
self.matches('GI.+') or
self.matches('GR.+') or
self.matches('HU.+') or
self.matches('IS.+') or
self.matches('IE.+') or
self.matches('IT.+') or
self.matches('LV.+') or
self.matches('LI.+') or
self.matches('LT.+') or
self.matches('LU.+') or
self.matches('MT.+') or
self.matches('MC.+') or
self.matches('NL.+') or
self.matches('NO.+') or
self.matches('PL.+') or
self.matches('RO.+') or
self.matches('SM.+') or
self.matches('SK.+') or
self.matches('SI.+') or
self.matches('ES.+') or
self.matches('SE.+') or
self.matches('CH.+') or
self.matches('GB.+')
)
```

--SOAP <-> Finvoice crosscheck rule:

-- Context: Finvoice

-- For at least one instance of Envelope/Header/MessageHeader/From/To[Role="Receiver"]/PartyId there must be a Finvoice/MessageTransmissionDetails/MessageReceiverDetails/ToIdentifier with an identical value, in a corresponding position (e.g. Envelope[1] <-> Finvoice[1], etc.).

```
let SFinvoice : Sequence(Finvoice) =
Finvoice.allInstances()->asSequence() in

let SMessageHeader : Sequence(MessageHeader) =
MessageHeader.allInstances()->asSequence() in

let finvoiceId : string =
self.MessageTransmissionDetails.MessageReceiverDetails.ToIdentifier in

let BPartyId : Bag(PartyId) =
SMessageHeader
->at(SFinvoice->indexOf(self)).To
->select(t | t.Role = "Receiver")
->collect(t | t.PartyId) in

BPartyId->size() > 0 implies
BPartyId->exists(soapId | soapId = finvoiceId)
```

-- Same rule as above with MessageHeader (Envelope/Header/MessageHeader) as context:

```
let SFinvoice : Sequence(Finvoice) =
Finvoice.allInstances()->asSequence() in

let SMessageHeader : Sequence(MessageHeader) =
MessageHeader.allInstances()->asSequence() in

let BPartyId : Bag(PartyId) =
self.To
->select(f | f.Role = "Receiver")
->collect(f | f.PartyId) in

BPartyId->size() > 0
implies
BPartyId->exists(id |
   SFinvoice->exists(fv |
      id = fv.MessageTransmissionDetails.MessageReceiverDetails.ToIdentifier and
      SMessageHeader->indexOf(self) = SFinvoice->indexOf(fv)
   )
)
```

-- Query that selects elements that have LanguageCode attributes with values occurring more than once:

```
let sift : Bag(TextLanguageOptional) = self.SellerInstructionFreeText in
sift->select(s1 |
```

```
sift->select(s2 |
s2.LanguageCode = s1.LanguageCode and sift->asSequence()->indexOf(s2) <> sift->asSequence()-
>indexOf(s1)
)->notEmpty()
)
```

-- Checking the date from InstrdId (format: YYYYMMDD)

```
self.InstrId.size() = 35 implies (
   let s1 : string = self.InstrId.substring(0,4)
   in
   let s2 : string = self.InstrId.substring(4,6)
   in
   let s3 : string = self.InstrId.substring(6,8)
   in
   let sAll : string = s1+"-"+s2+"-"+s3
   in (
      sAll.toDate().allowedDaysInFuture(2) and
      sAll.toDate().allowedDaysInPast(2)
      )
   )
```

-- Checking if a pair of elements within a transaction are unique within the file

```
let pairs : Bag(String) = CreditTransferTransaction39.allInstances()->collect(tx |
   tx.PmtId.InstrId.toString() + tx.InstgAgt.FinInstnId.BICFI.toString()) in
(
   let pair : String = self.PmtId.InstrId.toString() + self.InstgAgt.FinInstnId.BICFI.toString() in
   (
      pairs->count(pair) < 2
   )
)
```

-- Validating sequenced number

```
let seq : Sequence(ACHString) = self.Batch.BatchHeader.BatchNumber->asSequence() in
let nbs : Bag(Integer) = seq->collect(a | a.toInteger())->asBag() in
let frst : Integer = nbs->asSequence()->first() in
nbs->iterate(n: integer; i: integer = 0 |
 if(i = -1) then
  i
 else if (n = i + frst) then
   i + 1
 else
  -1
 endif
 endif
) <> -1
```

-- Today's date

```
self.CreDtTm.allowedDaysInPast(0) and
self.CreDtTm.allowedDaysInFuture(1)
```

```
-- Modulus 10 algorithm for 8 char content, with weight 3,7,1,3,7,1,3,7
let sNb : string = self.Field3 in
let iWeightedTotals : interger = (
(sNb.substring(0, 1) * 3) +
(sNb.substring(1, 2) * 7) +
(sNb.substring(2, 3) * 1) +
(sNb.substring(3, 4) * 3) +
(sNb.substring(4, 5) * 7) +
(sNb.substring(5, 6) * 1) +
(sNb.substring(6, 7) * 3) +
(sNb.substring(7, 8) * 7) )

let iCalculatedCheckDigit : integer = iWeightedTotals mod 10 in
let iCheckDigit : integer = self.Field3.toInteger() in
let bIsNbNumeric : boolean = self.Field3.isNumeric() in
let bIsCheckDigitNumeric : boolean = self.Field4.isNumeric() in
let bfieldsAreNumerc : boolean = bIsNbNumeric and bIsCheckDigitNumeric in

bfieldsAreNumerc implies iCalculatedCheckDigit = iCheckDigit
```