```
<!--
 The description of the XML document which must fit the X-definition 4. 2
 specification.
 The meta namespace of X-definition ("METAXD" is http://www.xdef.org/xdef/4.2).
 The prefix of objects describing the X-definition 4.2 is XD4.2.
-->

<xd:def xmlns:xd  = "METAXD"
  name            = "Ver4.2"
  root            = "XD4.2:def | XD4.2:collection | XD4.2:lexicon
                     | XD4.2:declaration | XD4.2:BNFGrammar | XD4.2:component"
  xmlns:XD4.2     = "http://www.xdef.org/xdef/4.2"
  xmlns:w         = "http://www.xdef.org/xdef/4.2"
  w:metaNamespace = "METAXD" >

/*****************************************************************************/
/*X-definition of X-definitions ver 4.2, metanamespace: METAXD, prefix: XD4.2 */
/*****************************************************************************/

    <XD4.2:collection xd:script = "init NS = @metaNamespace
      ? (String) @metaNamespace : 'http://www.xdef.org/xdef/4.2';
      options moreAttributes"
    include       = "optional uriList; options acceptQualifiedAttr"
    metaNamespace = "optional uri; options acceptQualifiedAttr"
    xd:attr       = "occurs * getAttrName().startsWith('impl-');">
    <xd:mixed>
      <!-- Here may be objects from all versions of X-definition -->
      <XD4.2:def xd:script = "occurs *; ref XD4.2:def" />
      <XD4.2:declaration xd:script = "occurs *; ref XD4.2:declaration" />
      <XD4.2:BNFGrammar xd:script = "occurs *; ref XD4.2:BNFGrammar" />
      <XD4.2:component xd:script = "occurs *; ref XD4.2:component"/>
      <XD4.2:lexicon xd:script = "occurs *; ref XD4.2:lexicon" />
    </xd:mixed>
  </XD4.2:collection>

  <XD4.2:def xd:script = "init NS = @metaNamespace
      ? (String) @metaNamespace : 'http://www.xdef.org/xdef/4.2';
      options moreAttributes"
    name          = "optional QName; options acceptQualifiedAttr"
    metaNamespace = "optional uri; options acceptQualifiedAttr"
    root          = "optional rootList; options acceptQualifiedAttr"
    include       = "optional uriList; options acceptQualifiedAttr"
    script        = "optional xdefScript; options acceptQualifiedAttr"
    importLocal   = "optional importLocal; options acceptQualifiedAttr"
    xd:attr       = "occurs * getAttrName().startsWith('impl-');">
    <!-- Names of other attributes (see xd:attr) must start with "impl-" -->

    <xd:mixed>
      <XD4.2:macro xd:script = "occurs *; ref XD4.2:macro" />
      <XD4.2:declaration xd:script = "occurs *; ref XD4.2:declaration"
        scope = "optional enum('global','local'); options acceptQualifiedAttr"/>
      <XD4.2:lexicon xd:script = "occurs *; ref XD4.2:lexicon" />
      <XD4.2:BNFGrammar xd:script = "occurs *; ref XD4.2:BNFGrammar"/>
      <XD4.2:component xd:script = "occurs *; ref XD4.2:component"/>
      <xd:choice occurs = "*">
          <XD4.2:choice xd:script = "occurs *;
                            match @name || @XD4.2:name; ref XD4.2:choiceDef"
              name = "required QName; options acceptQualifiedAttr" />
          <XD4.2:mixed xd:script = "occurs *;
                            match @name || @XD4.2:name; ref XD4.2:mixedDef"
              name = "required QName; options acceptQualifiedAttr" />
          <XD4.2:sequence xd:script = "occurs *;
                            match @name || @XD4.2:name; ref XD4.2:sequenceDef"
              name = "required QName; options acceptQualifiedAttr" />
          <XD4.2:list xd:script = "occurs *;
                            match @name || @XD4.2:name; ref XD4.2:listDef"
              name = "required QName; options acceptQualifiedAttr" />
          <XD4.2:json  xd:script = "occurs *;"
              name = "required QName; options acceptQualifiedAttr" >
              required xonScript;
          </XD4.2:json>
          <XD4.2:xon  xd:script = "occurs *;"
              name = "required QName; options acceptQualifiedAttr" >
```

```
            required xonScript;
        </XD4.2:xon>
        <XD4.2:any xd:script = "occurs *; match @XD4.2:name; ref XD4.2:anyDef;
                         options moreAttributes, moreElements"
            XD4.2:name = "required QName;" />
        <xd:any xd:script = "occurs *; ref xelement" />
        optional valueScript;
    </xd:choice>
  </xd:mixed>

</XD4.2:def>

<XD4.2:macro xd:script = "occurs *; options moreAttributes"
  name    = "required QName; options acceptQualifiedAttr"
  xd:attr = "occurs * string()" >
  optional string();
</XD4.2:macro>

<XD4.2:declaration>
   <xd:mixed>
     <XD4.2:macro xd:script = "occurs *; ref XD4.2:macro" />
     ? declarationScript;
   </xd:mixed>
</XD4.2:declaration>

<XD4.2:BNFGrammar extends="optional xdIdentifier; options acceptQualifiedAttr"
   name = "xdIdentifier; options acceptQualifiedAttr"
   scope = "optional enum('global','local'); options acceptQualifiedAttr">
     optional bnfGrammar; /* may be nothing */
</XD4.2:BNFGrammar>

<XD4.2:component>
    required xcomponent;
</XD4.2:component>

<XD4.2:lexicon language = "javaIdentifier" default = "optional yesNo">
  optional thesaurus();
</XD4.2:lexicon>

<!-- model of element -->
<xelement xd:script = "match getNamespaceURI() NE NS; options moreAttributes"
  xd:attr   = "occurs * attributeScript"
  xd:text   = "occurs * valueScript"
  XD4.2:script = "optional elementScript" >
  <xd:choice occurs = "*" ref = "xcontent" />
</xelement>

<xd:choice name = "xcontent">
  <XD4.2:choice xd:script = "occurs *;
                  match @ref || @XD4.2:ref; ref XD4.2:choiceRef" />
  <XD4.2:choice xd:script = "occurs *; ref XD4.2:choiceDef" />
  <XD4.2:mixed xd:script  = "occurs *;
                  match @ref || @XD4.2:ref; ref XD4.2:mixedRef" />
  <XD4.2:mixed xd:script  = "occurs *; ref XD4.2:mixedDef" />
  <XD4.2:sequence xd:script = "occurs *;
                  match @ref || @XD4.2:ref; ref XD4.2:sequenceRef" />
  <XD4.2:sequence xd:script = "occurs *; ref XD4.2:sequenceDef" />
  <XD4.2:list xd:script = "occurs *;
                  match @ref || @XD4.2:ref; ref XD4.2:listRef"/>
  <XD4.2:any xd:script = "occurs *; match @XD4.2:ref; ref XD4.2:anyRef"/>
  <XD4.2:any xd:script = "occurs *; match !@XD4.2:ref; ref xelement"/>
  <xd:any xd:script = "occurs *; ref xelement" />
  <XD4.2:text> optional valueScript; </XD4.2:text>
  optional valueScript;
</xd:choice>

<XD4.2:choiceRef occurs = "optional Occurrence; options acceptQualifiedAttr"
  create = "optional elementCreateCode; options acceptQualifiedAttr"
  ref    = "required xposition; options acceptQualifiedAttr"
  script = "optional string; options acceptQualifiedAttr" />

<XD4.2:choiceDef occurs = "optional Occurrence; options acceptQualifiedAttr"
  create = "optional elementCreateCode; options acceptQualifiedAttr"
  ref    = "illegal; options acceptQualifiedAttr"
```

```
    script = "optional groupScript; options acceptQualifiedAttr" >
    <xd:choice ref = "xcontent" occurs = "*" />
  </XD4.2:choiceDef>

  <XD4.2:sequenceRef occurs = "optional Occurrence; options acceptQualifiedAttr"
    create = "optional elementCreateCode; options acceptQualifiedAttr"
    ref    = "required xposition; options acceptQualifiedAttr"
    script = "optional groupScript; options acceptQualifiedAttr" />

  <XD4.2:sequenceDef occurs = "optional Occurrence; options acceptQualifiedAttr"
    create = "optional elementCreateCode; options acceptQualifiedAttr"
    ref    = "illegal; options acceptQualifiedAttr"
    script = "optional groupScript; options acceptQualifiedAttr" >
    <xd:choice ref = "xcontent" occurs = "*" />
  </XD4.2:sequenceDef>

  <XD4.2:mixedRef ref = "required xposition; options acceptQualifiedAttr"
    empty  = "optional booleanLiteral; options acceptQualifiedAttr"
    create = "optional elementCreateCode; options acceptQualifiedAttr"
    script = "optional groupScript; options acceptQualifiedAttr" />

  <XD4.2:mixedDef ref = "optional xposition; options acceptQualifiedAttr"
    empty  = "optional booleanLiteral; options acceptQualifiedAttr"
    create = "optional elementCreateCode; options acceptQualifiedAttr"
    script = "optional groupScript; options acceptQualifiedAttr" >
    <xd:choice xd:script = "*; ref xcontent;" />
  </XD4.2:mixedDef>

  <XD4.2:listRef ref = "required xposition; options acceptQualifiedAttr" />

  <XD4.2:listDef ref = "illegal; options acceptQualifiedAttr">
    <xd:choice xd:script = "*; ref xcontent;" />
  </XD4.2:listDef>

  <XD4.2:anyDef XD4.2:name = "required QName"
    XD4.2:script = "optional groupScript;
                    options moreAttributes, moreElements" />

  <XD4.2:anyRef XD4.2:script = "optional groupScript;
                    options moreAttributes, moreElements" />

/*****************************************************************************/
/*    Declaration of value types.                                            */
/*****************************************************************************/
<xd:declaration>
  /* Variable NS is used as the namespace of the checked source.*/
  String NS;

/*****************************************************************************/
/* Types of values see BNF grammar below                                     */
/*****************************************************************************/
  type rootList XDScript.rule('RootList');
  type xdefScript XDScript.rule('XdefScript');
  type declarationScript XDScript.rule('DeclarationScript');
  type valueScript XDScript.rule('ValueScript');
  type attributeScript XDScript.rule('AttributeScript');
  type elementScript XDScript.rule('ElementScript');
  type groupScript XDScript.rule('ElementScript');
  type groupModelScript XDScript.rule('ElementScript');
  type Occurrence XDScript.rule('Occurrence');
  type elementCreateSection XDScript.rule('ElementCreateSection');
  type elementCreateCode XDScript.rule('ElementCreateCode');
  type xdIdentifier XDScript.rule('Identifier');
  type xposition XDScript.rule('XPosition');
  type booleanLiteral XDScript.rule('BooleanLiteral');
  type bnfGrammar XDScript.rule('BNFGrammar');
  type xcomponent XDScript.rule('XCComponent');
  type languageId XDScript.rule('LanguageId');
  type javaIdentifier XDScript.rule('JavaIdentifier');
  type yesNo XDScript.rule('YesNo');
  type thesaurus XDScript.rule('Lexicon');
  type importLocal XDScript.rule('importLocal');
  type xonScript XDScript.rule('XONScript');
  type iniprops XDScript.rule('INIProps');
  /** Check element name and namespace URI (used in match section) */
  boolean xdName(String name) {
    return getElementLocalName() EQ name AND getNamespaceURI() EQ NS;
  }
</xd:declaration>

/*****************************************************************************/
/* Declaration of BNF grammar for X-Script                                   */
/*****************************************************************************/

  <xd:BNFGrammar name = "XDScript">
<![CDATA[

/*****************************************************************************/
/* x-Script BNF grammar rules                                                */
/*                                                                           */
/* Note the inline BNF method "$rule" is used to generate code for the       */
/* compiler of X-definitions. To understand the syntax you can simply        */
/* ignore them.                                                              */
/*****************************************************************************/

Letter ::= $letter /* any letter.*/

Char ::= $xmlChar /* any XML character.*/

WhiteSpace ::= [#9#10#13 ]

Comment ::= "/*" ([^*]+ | "*" - "*/")* "*/"

S ::= (WhiteSpace | Comment)+ /* Sequence of whitespaces or comments */

Digit ::= [0-9]

Digits ::= [0-9]+

HexaDigit ::= Digit | [a-fA-F]

HexaDigits ::= (Digit | [a-fA-F])+

SemicolonOrSourceEnd ::= S? (";" S? | $eos)
  /* At the end of source text the semicolon is not required. */

/* Keywords of the X-script */
Keyword ::= "if" | "else" | "do" | "while" | "continue" | "break" | "switch"
  | "case" | "for" | "return" | "def" | "try" | "catch" | "throw" | "finally"
  /* | "fixed" */ | "external" | "new" | "required" | "optional" | "ignore"
  | "illegal" | "occurs" | "onTrue" | "onError" | "onAbsence" | "default"
  | "onExcess" | "onStartElement" | "onIllegalAttr" | "onIllegalText" | "var"
  | "onIllegalElement" | "onIllegalRoot" | "onXmlError" | "create" | "init"
  | "options" | "option" | "ref" | "match" | "final" | "forget" | "template"
  | "type" | "uniqueSet" | "EQ" | "NE" | "LT" | "LE" | "GT" | "GE" | "LSH"
  | "RSH"  | "RRSH" | "AND" | "OR" | "XOR" | "MOD" | "NOT" | "NEG" | "OOR"
  | "AAND" | "CHECK" | "true" | "false" | "implements" | "extends" | "onFalse"
  | "$$$script"
  /* Note "fixed" is not the keyword since it is used also as the name of
   validation method. */

/* Predefined constants */
PredefinedConstant ::= ("$MAXFLOAT" | "$MAXINT" | "$MININT" | "$MINFLOAT" |
  "$PI" |"$E" | "$NEGATIVEINFINITY" | "$POSITIVEINFINITY" | "null") $rule

Identifier ::= JavaIdentifier - Keyword

LanguageId ::= [a-z] {2,3}

YesNo ::= "yes" | "no"

JavaIdentifier ::= ((Letter | "_" | "$")  (Letter | Digit | "_" | "$")* )

RawIdentifier ::= (Letter | "_")  (Letter | Digit | "_")*

QualifiedIdentifier ::= JavaIdentifier ("." JavaIdentifier)+ | JavaIdentifier
```

```
BooleanLiteral ::= ("true" | "false") $rule

DecimalInteger ::= ("_"* Digits)+ ("_")*

IntegerLiteral ::= (("0" [DdIi])? DecimalInteger
  | "0" [Xx] ("_"* HexaDigits)+ ("_")*) $rule
  /* Inside a number specification it is possible to insert the character "_".
    This character does not influence the value of the number, it just makes
    a number more readable. E.g. the number 123456789 you can be written
    as 123_456_789 (or 0x0f123456 as 0x0f_12_34_56). */

FloatLiteral ::= (("0" [Dd]) DecimalInteger "." DecimalInteger? )
  | ((DecimalInteger ("." DecimalInteger Exponent? | Exponent))) $rule

Exponent ::= [Ee] [-+]? [0-9]+

NumberLiteral ::= FloatLiteral | IntegerLiteral

SpecChar ::= "\" ("\" | '"' | "'" | "n" | "r" | "t") | UnicodeCharSpecification

UnicodeCharSpecification ::= "\u" HexaDigit{4}

StringLiteral ::= ("'" ("''" | [^'\] | SpecChar)* "'" |
  '"' ('""' | [^"\] | SpecChar)* '"') $rule
  /* The opening and closing delimiter must be either """ or "'". The occurrence
     of this delimiter inside of literal can be recorded as a double delimiter
     or in the form of SpecChar. */

Literal ::= BooleanLiteral | NumberLiteral | StringLiteral

XMLName ::= $xmlName /* XMLName see XML specification */

KeyName ::= "%" XMLName

AttributeName ::= "@" XMLName

XDefName ::= XMLName

XModelName ::= XMLName

importLocal ::= S? ( (XMLName | "#") S? ("," S? (XMLName | "#") S?)*)?

XPosition ::= (XDefName? "#")? XModelName
  ("/" (XMLName | XGroupRererence | XAnyReference) XPositionIndex?)*
  ("/" (XAttrReference | XTextReference))?

XPositionIndex ::= "[" Digits "]"

XGroupRererence ::= "$mixed" | "$choice" | "$sequence"

XAnyReference ::= "$any"

XAttrReference ::= "@" XMLName

XTextReference ::= "$text" XPositionIndex?

RootList ::= S? RootSpecification (S? "|" S? RootSpecification)* S?
  /* all rootspecification in the list must be unique */

RootSpecification ::= XPosition | "*"

ExternalMethodType ::= QualifiedIdentifier (S?"[" S? "]")?

ExternalMethod ::= ExternalMethodType S? QualifiedIdentifier S?
  "(" S? ExternalMethodParamList? S? ")" S? ("as" S? Identifier)?

ExternalMethodParam ::= ExternalMethodType (S JavaIdentifier)?
  /* the parameter name (JavaIdentifier) is optional and ignored */

ExternalMethodParamList ::= ExternalMethodParam (S? "," S? ExternalMethodParam)*

MacroReference ::= "$" "{" S? XMLName S? MacroParams? S? "}"

MacroParams ::= "(" S? Identifier (S? "," S? Identifier)* S? ")"
```

```
/*****************************************************************************/
/* Script expression                                                         */
/*****************************************************************************/

Expression ::= Expr1 (S? "?" S? Expression S? ":" S? Expression)?

OperatorLevel_1 ::= ("AND" | "AAND" | "&&" | "&") $rule

Expr1 ::= Expr2 (S? OperatorLevel_1 S? Expr2)*

OperatorLevel_2 ::= ("OR" | "OOR" | "XOR" | "||" | "|" | "^") $rule

Expr2 ::= Expr3 (S?  OperatorLevel_2 S? Expr3)*

OperatorLevel_3 ::= ( "LT" | "<" | "GT" | ">" | "==" | "EQ" | "LE"
  | "<=" | "GE" | ">=" | "!=" | "NE" | "<<" | "LSH" | ">>" | "RSH" | ">>>"
  | "RRSH" | "CHECK" ) $rule

Expr3 ::= Expr4 (S? OperatorLevel_3 S? Expr4)*

OperatorLevel_4 ::= ("*" | "/" | "%") $rule

Expr4 ::= Expr5 (S? OperatorLevel_4 S? Expr5)*

OperatorLevel_5 ::= ("+" | "-") $rule

Expr5 ::= Expr (S? OperatorLevel_5 S? Expr)*

Expr ::= (UnaryOperator S? | CastRequest S?)*
  (Value | Literal | "(" S? Expression S? ")") (S? "." S? Method)?

ConstantExpression ::= Literal /* ConstantExpression must be a Literal. */

CastRequest ::= S? "(" S? $rule TypeIdentifier S? ")" S?

UnaryOperator ::= "+" | ("-" | "!" | "NOT" | "~") $rule

TypeIdentifier ::= ("int" | "String" | "float" | "boolean" | "char" | "Datetime"
  | "Decimal" | "Duration" | "Exception" | "Container" | "Element" | "Message"
  | "Bytes" | "XmlOutStream" | "BNFGrammar" | "BNFRule" | "Parser" | "Service"
  | "ResultSet" | "Statement" | "ParseResult" | "Locale" | "uniqueSetKey"
  | "AnyValue" | "Output" | "Input" | "NamedValue" | "Regex" | "GPSPosition"
  | "Price" | "Telephone" | "URI" | "EmailAddr") $rule

Value ::= (Constructor | PredefinedConstant | Increment
  | Method | VariableReference | KeyParameterReference | Literal | AttributeName
  | $rule AssignmentStatement) (S? "." S? Method)?

NewValue ::= "new" S $rule TypeIdentifier S? ParameterList

Constructor ::= NewValue | NamedValue | ContainerValue

NamedValue ::= KeyName S? "=" S? Expression

ContainerValueStart ::= (NamedValue (S? "," S? NamedValue)* ) | Expression

ContainerValue ::= "[" S? (ContainerValueStart (S? "," S? Expression)* )? S? "]"

KeyParameterReference ::= KeyName

Method ::= (SchemaTypeName | (QualifiedIdentifier - Keyword)) $rule
  S? ParameterList?

SchemaTypeName ::= "xs:" Identifier /* prefix “xs:” is deprecated */

incAfter ::= ("++" | "--") S? $rule VariableReference
incBefore ::= VariableReference S? ("++" | "--") S? $rule

Increment ::= incAfter | incBefore
  /* The type VariableReference must be an integer or a float */

Parameter ::= (Expression | "*") $rule
```

```
ParameterList ::= "(" S? (Parameter
   (S? "," S? Parameter)* (S? "," S? "*")? S?)? ")" $rule

VariableReference ::= (Identifier - TypeIdentifier) $rule

MethodDeclaration ::= ("void" | TypeIdentifier) $rule
  S DeclaredMethodName S? ParameterListDeclaration S? Block

DeclaredMethodName ::= Identifier $rule

ParameterListDeclaration ::=
  "(" S? (SeqParameter (S? "," S? SeqParameter)* )? ")" $rule

SeqParameter ::= TypeIdentifier S ParameterName

KeyParameter ::= KeyName S? "=" S? (TypeIdentifier | ConstantExpression)

ParameterName ::= Identifier

ParentalExpression ::= S? "(" S? Expression S? ")" S?
  /* Result of ParentalExpression must be boolean. */

StatementExpression ::= Expression

/***************************************************************************/
/* Script statement                                                        */
/***************************************************************************/

Statement ::= S? (Statement1 | Statement2)

Statement1 ::= (Block | SwitchStatement | TryStatement)

Statement2 ::= (IfStatement | ForStatement | WhileStatement | DoStatement
  | ReturnStatement | ThrowStatement | BreakStatement | ContinueStatement
  | Method | Increment | AssignmentStatement)
  $info | EmptyStatement

EmptyStatement ::= ";"

StatementSequence ::= (S? VariableDeclaration S? ";" S? | Statement)*

Block ::= "{" StatementSequence S? "}"

SimpleStatement ::= S? (Statement1 | (Statement2 S? (";" S?)?)) | EmptyStatement

IfStatement ::= "if" ParentalExpression SimpleStatement
  (S? "else" S? SimpleStatement)?

ForStatement ::= "for" $rule S? "(" S? ForInit? S? ";" S?
  ForBooleanExpression? S? ";" S? ForStep? S? ")" S? SimpleStatement

ForBooleanExpression ::= $rule Expression

ForInit ::= $rule (AssignmentStatement | VariableDeclaration)

ForStepStatement ::= $rule (Method | Increment | AssignmentStatement)

ForStep ::= ForStepStatement (S? "," S? ForStepStatement)*

WhileStatement ::= "while" $rule ParentalExpression SimpleStatement

DoStatement ::= "do" $rule (S? Block | S  Statement) WhileCondition

WhileCondition ::= S? "while" $rule ParentalExpression

SwitchStatement ::= "switch" $rule S? "(" S? Expression S? ")" S?
   "{" SwitchBlockStatementVariant* S? "}"
  /* Result of Expression must be integer or string. Each variant may occur
     in the switch statement only once. */

SwitchBlockStatementVariant ::=
   S? (DefaultVariant | CaseVariant) S? ":" S? StatementSequence?
  /* Type of ConstantExpression must be integer or string. */
```

```
DefaultVariant ::= "default" S? $rule

CaseVariant ::= "case" S?  $rule ConstantExpression

ThrowStatement ::= "throw" S? $rule ExceptionValue

ExceptionValue ::= NewException | Identifier

NewException ::= "new" S? $rule "Exception" S? "(" S? (Expression S?)? ")"

TryStatement ::= "try" S? $rule "{" S? StatementSequence S? "}"
  S? CatchStatement

CatchStatement ::=
  "catch" S? $rule "(" S? "Exception" S?  Identifier S? ")" S?
  "{" S? StatementSequence S? "}"

ReturnStatement ::= "return" $rule (S? Expression)?

BreakStatement ::= "break" $rule (S? Identifier)?

ContinueStatement ::= "continue" $rule (S? Identifier)?

AssignmentStatement ::= (Identifier S? ((AssignmentOperator S? Expression) |
 (("=" S? Identifier))+ S? (AssignmentOperator S? Expression)) |
  S? AssignmentOperator S? Expression)

AssignmentOperator ::= (("|" | "OR" | "^" | "+" | "-" | "*" | "/" | "&" | "AND"
  | "%" | "<<" | "LSH" | ">>>"| "RRSH"  | ">>" | "RSH" )? "=") $rule

VariableModifier ::= ("final" | "external") $rule S

VariableDeclaration ::= VariableModifier*
  TypeIdentifier S VariableDeclarator (S? "," VariableDeclarator)*

VariableDeclarator ::= S? (AssignmentStatement | Identifier)

Occurrence ::= ("occurs" S)? ("required" | "optional" | "ignore" | "illegal"
  | "*" | "+" | "?" | ("*" | "+" | "?" |
  (IntegerLiteral (S? ".." (S? ("*" | IntegerLiteral))? )? ))) $rule
  /* The value of the second IntegerLiteral (after "..")  must be greater or
     equal to the first one. */

ExplicitCode ::= Block
  /* If the result value is required it must be returned by the command
   "return" */

/*****************************************************************************/
/* Script of X-definition header                                             */
/*****************************************************************************/

XdefScript ::= (S | XdefInitSection | XdefOnIllegalRoot | XdefOnXmlError
  | XdefOptions | ";")*
  /* Each item can be specified only once. */

XdefInitSection ::= "init" S Statement

XdefOnIllegalRoot ::= "onIllegalRoot" S Statement

XdefOnXmlError ::= "onXmlError" S Statement

XdefOptions ::= ("options" | "option") S XdefOptionsList

XdefOptionsList ::= XdefOption (S? "," S? XdefOption)*
  /* Each option can be specified only once. */

XdefOption ::= "moreAttributes" | "moreElements" | "moreText"
  | "forget" | "notForget" | "clearAdoptedForgets"
  | "resolveEntities" | "ignoreEntities" | "resolveIncludes" | "ignoreIncludes"
  | "preserveComments" | "ignoreComments" | "acceptEmptyAttributes"
  | "preserveEmptyAttributes" | "ignoreEmptyAttributes"
  | "preserveAttrWhiteSpaces" | "ignoreAttrWhiteSpaces"
  | "preserveTextWhiteSpaces" | "ignoreTextWhiteSpaces"
  | "setAttrUpperCase" | "setAttrLowerCase"
  | "setTextUpperCase" | "setTextLowerCase"
  | "acceptQualifiedAttr" | "notAcceptQualifiedAttr"
  | "trimAttr" | "noTrimAttr" | "trimText" | "noTrimText"
  | "resolveEntities" | "ignoreEntities" | "resolveIncludes" | "ignoreIncludes"
  | "preserveComments" | "ignoreComments"

/******************************************************************************/
/* Script of text nodes and attributes                                        */
/******************************************************************************/

AttributeScript ::= ValueScript

ValueScript ::= ("$$$script:"?) (ValueValidationSection | ValueInitSection
  | ValueOnTrueSection | ValueOnFalseSection | ValueOnErrorSection
  | ValueOnAbsenceSection | ValueDefaultSection | ValueCreateSection
  | ValueFinallySection | ValueMatchSection | AttributeOnStartSection
  | AttributeOptions | OnIllegalSection | Reference | S |";")*
  /* The keyword "$$$script" can be specified only in the template mode.
     Each section can be specified only once.*/

OnIllegalSection ::= ("onIllegalAttr" | "onIllegalText") S? Statement?

AttributeOnStartSection ::= "onStartElement" S? Statement?

ValueValidationSection ::= ("fixed" S $rule (Expression | Block)
  | (Occurrence S? CheckValueSpecification?) | CheckValueSpecification)

CheckValueSpecification ::= ExplicitCode | ValidationExpression | TypeMethodName
  /* ExplicitCode must return a value of boolean type or ParseResult. */

ValidationExpression ::= ValidationMethod | Expression
  /* Result of ValidationExpression must be a boolean or ParseResult type. */

ValidationMethod ::= SchemaValidationMethod | XDValidationMethod

SchemaValidationMethod ::= ParseMethod

XDValidationMethod ::= ParseMethod

ParseMethod ::= Method

TypeMethodName ::= Identifier

ValueInitSection ::= "init" S? (ExplicitCode | Statement)

ValueOnTrueSection ::= "onTrue" S? (ExplicitCode | Statement)
  /* If Method or ExplicitCode returns a value, it will be ignored. */

ValueOnFalseSection ::= "onFalse" S? (ExplicitCode | Statement)

ValueOnErrorSection ::= "onError" S? (ExplicitCode | Statement)

ValueOnAbsenceSection ::= "onAbsence" S? (ExplicitCode | Statement)

ValueCreateSection ::= "create" S? (ExplicitCode | ValueCreateExpression)
  (S? ";")* /* ExplicitCode must return a value of String type. */

ValueCreateExpression ::= Expression
  /* ValueCreateExpression must return a value of String type. */

ValueDefaultSection ::= "default" S? (ExplicitCode | Expression)
  /* Expression or ExplicitCode must return the value of the String. */

ValueFinallySection ::= "finally" S? Statement

ValueMatchSection ::= "match" S? (Expression | ExplicitCode)
  /* Expression or ExplicitCode must here return a value of boolean type. */

AttributeOptions ::= ValueOptions

ValueOptions ::= ("options" | "option") S? ValueOptionsList

ValueOptionsList ::= ValueOption (S? "," S? ValueOption)*
  /* Each option can be specified only once. */
```

```
ValueOption ::= "preserveTextWhiteSpaces" | "ignoreTextWhiteSpaces"
  | "setTextUpperCase" | "setTextLowerCase" | "trimText" | "noTrimText"
  | "preserveAttrWhiteSpaces" | "ignoreAttrWhiteSpaces" | "cdata"
  | "setAttrUpperCase" | "setAttrLowerCase" | "trimAttr" | "noTrimAttr"
  | "ignoreEmptyAttributes" | "acceptEmptyAttributes"
  | "acceptQualifiedAttr" | "notAcceptQualifiedAttr" | "preserveTextCase"

Reference ::= "ref" S XPosition

/******************************************************************************/
/* Script of elements                                                         */
/******************************************************************************/

ElementScript ::= $info ElementExecutivePart* S?

ElementExecutivePart ::= "$$$script:"? S? (TemplateSection | Occurrence
  | ElementVarSection | ElementMatchSection | ElementInitSection
  | ElementOnAbsenceSection | ElementOnExcessSection | ElementCreateSection
  | ElementFinallySection | ElementOptions | Reference | ElementForgetSection
  | ElementOnStartSection | ElementOnIllegalSection | ElementStructureCompare
  | ";")
  /* Each item can be specified only once. If the occurrence is not specified,
     the implicit value is "required". The keyword "$$$script" can be specified
     only in the template model. */

TemplateSection ::= "template" $rule

ElementVarSection ::= "var" S?
  (("{"(S? ElementVarSectionItem S?)* "}") | ElementVarSectionItem S?)

ElementVarSectionItem ::= TypeDeclaration | VariableDeclaration S? ";"  | S? ";"

ElementInitSection ::= "init" S? Statement?

ElementMatchSection ::= "match" S? (Expression | ExplicitCode)
  /* Expression or ExplicitCode must here return a value of boolean type. */

ElementOnStartSection ::= "onStartElement" S? Statement?

ElementOnExcessSection ::= "onExcess" S? Statement?

ElementOnAbsenceSection ::= "onAbsence" S? Statement?

ElementOnIllegalSection ::= "onIllegalElement" S? Statement?

ElementCreateSection ::= "create" ElementCreateCode

ElementCreateCode ::= S? (Expression | ExplicitCode) S?
  /* Expression or ExplicitCode must return a value of Container or Element. */

ElementFinallySection ::= "finally" S? Statement?

ElementForgetSection ::= "forget"

ElementStructureCompare ::= ("implements" | "uses") S XPosition

ElementOptions ::= ("options" | "option") S? ElementOptionsList

ElementOptionsList ::=  ElementOption (S?  "," S? ElementOption)*
  /* Each option can be specified only once. */

ElementOption ::= "moreAttributes" | "moreElements" | "moreText"
  | "forget" | "notForget" | "acceptEmptyAttributes" | "clearAdoptedForgets"
  | "preserveEmptyAttributes" | "ignoreEmptyAttributes"
  | "preserveAttrWhiteSpaces" | "ignoreAttrWhiteSpaces"
  | "preserveTextWhiteSpaces" | "ignoreTextWhiteSpaces" | "setAttrUpperCase"
  | "setAttrLowerCase" | "setTextUpperCase" | "setTextLowerCase"
  | "acceptQualifiedAttr" | "notAcceptQualifiedAttr" | "trimAttr" | "noTrimAttr"
  | "trimText" | "noTrimText" | "resolveEntities" | "ignoreEntities"
  | "resolveIncludes" | "ignoreIncludes" | "preserveComments" | "ignoreComments"
  | "preserveTextCase" | "preserveAttrCase" | "acceptOther" | "ignoreOther"
  | "clearReports" | "preserveReports" | "nillable" | "noNillable"
```

```
/*****************************************************************************/
/* Script of the declaration part                                            */
/*****************************************************************************/

DeclarationScript ::= (S? (TypeDeclaration | ExternalMethodDeclaration
  | VariableDeclaration | MethodDeclaration | ";"))* S?

TypeDeclaration ::= ("type" S Identifier S?
  ((Identifier (S? "CHECK" ? Expression)? S? ";" S? )
  | TypeDeclarationBody))
  | UniqueSetDeclaration

TypeDeclarationBody ::= TypeExplicitCode
  | ( Expression (S? "CHECK" ? Expression)? )
  /* The first xpression  must return either a boolean
    or a ParseResult value. The CHECK expression must return boolean value*/

TypeExplicitCode ::= /* only X-definition version 2.0 */
  ("{" S? "parse" S? ":" S? (ExplicitCode | Statement ";"?) S? "}")
  | /* X-definition version 3.1 and higher */ ExplicitCode

ExternalMethodDeclaration ::= "external" S "method"
  ( S? "{" S? (ExternalMethod S? ";" S? )* ExternalMethod? S? ";"? S? "}"
  | (S ExternalMethod) ) S? ";"?

UniqueSetDeclaration ::= "uniqueSet" S Identifier S? UniqueSetDeclarationBody

UniqueSetDeclarationBody ::=
  ("{" UniqueSetItem (S?";" UniqueSetItem)* (S?";"S?)? S? "}") | Method
  /* The method must be a parser. */

UniqueSetItem ::= S? (UniqueSetVar | UniqueSetKey)

UniqueSetKey ::= S? Identifier (S?":"S? (("?" | "optional") S? )? Method )?
  /* The method must be a parser. */

UniqueSetVar ::= S? "var" S TypeIdentifier S Identifier
  (S? "," S? TypeIdentifier S Identifier)*

/*****************************************************************************/
/* BNF grammar                                                               */
/*****************************************************************************/

BNFGrammar ::= S? BNFMethodDeclarationSection? BNFRules S?

BNFMethodDeclarationSection ::= BNFMethodDeclaration (S? BNFMethodDeclaration)*

BNFMethodDeclaration ::= "%define" S BNFDefinedMethodName S? ":"
  S ("$" QualifiedIdentifier | BNFDefinedMethodName) S? BNFMethodparameters?

BNFMethodparameters ::= "(" S?
  (BNFMethodparameter (S? "," S? BNFMethodparameter)* ) ? S? ")"

BNFMethodparameter ::= Digits | BNFTerminalSymbol

BNFDefinedMethodName ::= "$" (BNFRuleName | Digits)

BNFRuleName ::= RawIdentifier

BNFRule ::= BNFRuleDecl BNFExpression

BNFRules ::= BNFRule (BNFRule)*

BNFRuleDecl ::= S? BNFRuleName S? "::=" S?

BNFRuleReference ::= BNFDefinedMethodName | (BNFRuleName - BNFRuleDecl)

BNFTerminalSymbol ::= "'" [^']* "'" | '"' [^"]* '"'
  | (BNFHexaCharacter (S? BNFHexaCharacter)* )

BNFHexaCharacter ::= "#" HexaDigits

BNFQuantifier ::= S? ("+" | "*" | "?" | BNFExplicitQuantifier)
```

```
BNFExplicitQuantifier ::= "{" S? Digits (S? "," S? Digits)? S? "}"

BNFSet ::= "[" ("^"? Char - "]")* "]" BNFQuantifier?

BNFTerm ::= (BNFTerminalSymbol | BNFSet | BNFRuleReference) BNFQuantifier?
  | BNFParentalExpr

BNFParentalExpr ::= "(" S? BNFUnion S? ")" S? BNFQuantifier?

BNFSequence ::= BNFTerm (S? BNFTerm)*

BNFRestriction ::= BNFSequence (S? "-" S? BNFTerm)?

BNFAll ::= BNFRestriction (S? "," S? BNFRestriction)*

BNFUnion ::=  BNFAll (S? "|" S? BNFAll)*

BNFExpression ::= (BNFUnion S?)+

/****************************************************************************/
/* XComponent                                                               */
/****************************************************************************/

JavaTypeName ::= '<' S? $JavaQName (S? JavaTypeName)?
  (S? "," S?  JavaTypeName)* S? '>'

JavaTypedQName ::= $JavaQName (S? JavaTypeName)?

XCComponent ::= S? (XCComponentCommand
  (S? ";" S? XCComponentCommand?)* )

XCComponentCommand ::= (XCBind | XCClass | XCEnum | XCInterface | XCRef)

XCBind ::= "%bind" S XMLName (S? "%with" S $JavaQName)? S? XCLink

XCClass ::= "%class" S JavaTypedQName
  (S "extends" S JavaTypedQName (S? "<" JavaTypedQName ">")? )?
  (S "implements" S JavaTypedQName (S? "," S? JavaTypedQName)*)?
  (S "%interface" S JavaTypedQName)? S? (XCWith)? (XCLink)?

XCWith ::= "%with" S JavaTypedQName

XCEnum ::= "%enum" S JavaTypedQName S (Identifier? "#")? XMLName

XCInterface ::= "%interface" S $JavaQName S? XCLink

XCRef ::= "%ref" S ((JavaTypedQName S XCLink)
  | ("%enum" S JavaTypedQName S (Identifier? "#")? XMLName))

XCLink ::= "%link" S ("*" | XPosition (S? "," S? XPosition)*)

/****************************************************************************/
/* Lexicon grammar rules                                                    */
/****************************************************************************/

Lexicon ::= (S? $rule XPosition S? "=" S? XMLName)* S?

/****************************************************************************/
/* INI/Properties grammar rules                                             */
/****************************************************************************/

INISP ::= [#9 ]+

ININewLine ::= (#13 #10 | #10)+

INIOctalDigit ::= [0-7]

INIEscapedChar ::=
   "\" ( INIOctalDigit | "n" | "t" | "t" | "f" | ":" | HexaDigit{4} )

INIBaseChar ::= INISP | [ -Z^-~]/*ASCII - newline,backslash*/

INIChar ::= INIEscapedChar | INIBaseChar
```

```
INISeparator ::= (ININewLine | INISP)*

INICommand ::= INISeparator (INIComment | INIProp | INISection) INISeparator

INIComment ::= "#" INIChar*

INIProp ::= INISP? (INIChar  - "=")+ INISP? "=" (INIChar - ININewLine)*

INISection::= INISP? "[" INISP? (INIChar - "]")+ INISP? "]"
   INISP? ("%script" INISP? "=" INISP? (INIChar - ININewLine)* )?

INIProps ::= INICommand*


/*****************************************************************************/
/* XON X-script                                                              */
/*****************************************************************************/

XONComment ::= "/*" ([^*]+ | "*" - "*/")* "*/"

XONSP ::= (WhiteSpace | XONComment)+ /* sequence of whitespaces and comments */

XONControlchar ::= '\"' | '\\' | '\/' | '\b' | '\f' | '\n' | '\r' | '\t' |
  ('\u' [0-9a-fA-F]{4})

XONString ::= ('"' ([^\"] | XONControlchar )* '"' |
  "'" ([^\'] | XONControlchar )* "'")

XONXscript ::= XONSP? ( ("%script" XONSP? "=" XONSP? XONString)
  | ( "%oneOf" (XONSP? "=" XONSP? XONString)? ) ) (XONSP? ",")?

XONPair ::= ("%anyName" (XONSP? "=" XONSP? XONString)?
  | $ncName | XONString) XONSP? ":" XONSP? XONScriptValue

XONMembers ::= XONSP? XONPair (XONSP? "," XONSP? XONPair)* (XONSP? ",")?

XONObject ::= "{" XONXscript? XONMembers? XONSP? "}"

XONList ::= XONScriptValue (XONSP? "," XONSP? XONScriptValue )* (XONSP? ",")?

XONArray ::= "[" XONXscript? XONSP? XONList? XONSP? "]"

XONScriptValue ::= XONSP? ( "%any" | XONString | XONArray | XONObject) XONSP?

XONScript ::= XONSP? (XONArray | XONObject | XONString)

]]>
  </xd:BNFGrammar>
```