Chapter 5: XML Schemas

Chapter 5 Objectives

- Benefits of XML Schemas
- How to create and use XML Schemas
- How to document your XML Schemas

Benefits of XML Schemas

- XML Schemas use XML syntax
- XML Schemas Namespace Support
- XML Schema Data Types
  - Simple and complex
  - Built-in and user-defined
- XML Schema Content Models
  - Reusability
  - Object inheritance
  - Content-model inheritance

Do we still need DTDs?

Simple Type vs. Complex Type

- Simple type – text only
  - Element containing text only – no other element or attribute
  - Attribute
- Complex type
  - Other cases

Is an empty element of simple or complex type?

The XML Schema Document

Example: nam5.xsd

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:target="http://www.example.com/name"
 targetNamespace="http://www.example.com/name"
 elementFormDefault="qualified">
 <element name="name">
  <complexType>
   <sequence>
    <element name="first" type="string"/>
    <element name="middle" type="string"/>
    <element name="last" type="string"/>
   </sequence>
   <attribute name="title" type="string"/>
  </complexType>
 </element>
</schema>
```


The XML Schema Instance Document

Example: name5.xml

```
<?xml version="1.0"?>
<name
 xmlns="http://www.example.com/name"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.example.com/name name5.xsd"
 title="Mr.">
  <first>John</first>
  <middle>Fitzgerald Johansen</middle>
  <last>Doe</last>
</name>
```

Need to match targetNamespace in schema

- Namespace-location pair
- Can have more than one pairs

Schema Declarations

Root element: schema

<schema *W3CXSchemaNamespace declaration*
targetNamespace declaration
targetNamespace="*URI*"
attributeFormDefault="*qualified* or *unqualified*"
elementFormDefault="*qualified* or *unqualified*"
version="version number">

The Schema XML Namespace

Examples

<schema xmlns="http://www.w3.org/2001/XMLSchema">
or
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
or
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

Target Namespaces

XML schema declares vocabularies for the namespace specified in targetNamespace.

Example 1

<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.com/name"
xmlns:target="http://www.example.com/name">

Example 2

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.com/name"
xmlns="http://www.example.com/name">

Element and Attribute Qualification

Qualified – associated with a namespace

```
<name xmlns="http://www.example.com/name">
  <first>John</first>
  <middle>Fitzgerald</middle>
  <last>Doe</last>
</name>
```

Qualified and Prefixed

```
<n:name xmlns:n="http://www.example.com/name">
  <n:first>John</n:first>
  <n:middle>Fitzgerald</n:middle>
  <n:last>Doe</n:last>
</n:name>
```

Unqualified – not associated with a namespace

```
<n:name xmlns:n="http://www.example.com/name">
  <first>John</first>
  <middle>Fitzgerald</middle>
  <last>Doe</last>
</n:name>
```

Which elements are unqualified?

<element> Declarations

<element
id="a unique ID"
name="*name of the element*"
type="*global data type, name of built-in or user-defined*"
ref="*global element declaration*"
form="*qualified or unqualified*"
minOccurs="*non negative number*"
maxOccurs="*non negative number* or '*unbounded*'"
default="*default value*"
fixed="*fixed value*"
:
>

Global versus Local

- ***Global declarations*** are declarations that appear as direct children of the <schema> element. Global element declarations can be *reused* throughout the XML Schema.

- ***Local declarations*** do not have the <schema> element as their direct parent and are valid only in their specific context.

Global versus Local

Example

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:target="http://www.example.com/name"
targetNamespace="http://www.example.com/name"
elementFormDefault="qualified">
  <element name="name">
    <complexType>
      <sequence>
        <element name="first" type="string"/>
        <element name="middle" type="string"/>
        <element name="last" type="string"/>
      </sequence>
      <attribute name="title" type="string"/>
    </complexType>
  </element>
</schema>
```


Creating a Local Type

Either complexType or simpleType:

complexType

```
<element name="name">
  <complexType>
    <sequence>
      <element name="first" type="string"/>
      <element name="middle" type="string"/>
      <element name="last" type="string"/>
    </sequence>
    <attribute name="title" type="string"/>
  </complexType>
</element>
```

Creating a Local Type

simpleType

```
<element name="name">
  <simpleType>
    <restriction base="string">
      <enumeration value="Home"/>
      <enumeration value="Work"/>
      <enumeration value="Cell"/>
      <enumeration value="Fax"/>
    </restriction>
  </simpleType>
</element>
```


Using a Global Type

Example 1: name6.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:target="http://www.example.com/name"
  targetNamespace="http://www.example.com/name"
  elementFormDefault="qualified">

  <complexType name="NameType">
    <sequence>
      <element name="first" type="string"/>
      <element name="middle" type="string"/>
      <element name="last" type="string"/>
    </sequence>
    <attribute name="title" type="string"/>
  </complexType>

  <element name="name" type="target:NameType"/>
</schema>
```

Using a Global Type

Example 2

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.example.com/name"
  targetNamespace="http://www.example.com/name"
  elementFormDefault="qualified">

  <xs:complexType name="NameType">
    <xs:sequence>
      <xs:element name="first" type="xs:string"/>
      <xs:element name="middle" type="xs:string"/>
      <xs:element name="last" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="title" type="xs:string"/>
  </xs:complexType>

  <xs:element name="name" type="NameType"/>
</xs:schema>
```

Note the xs prefixes..

Using a Global Element

Example: name7.xsd

```
<element name="first" type="string"/>
<element name="middle" type="string"/>
<element name="last" type="string"/>

<complexType name="NameType">
  <sequence>
    <element ref="target:first"/>
    <element ref="target:middle"/>
    <element ref="target:last"/>
  </sequence>
  <attribute name="title" type="string"/>
</complexType>

<element name="name" type="target:NameType"/>
```

Note the target prefixes..

Naming Elements

<element name="first" type="string"/>
<element name="description" type="string"/>

<element name="2ndElement" type="string"/>
<element name="target:middle" type="string"/>

Element Qualified Form

form

- Is an attribute that can be set to
  - qualified
  - unqualified
- Used to override default element qualification
- Global elements are always qualified
- Default
  - value of elementFormDefault attribute defined in the schema element

Cardinality

It is important to note that the minOccurs and maxOccurs attributes are not permitted within global element declarations. Instead, you should use these attributes within the element references in your content models.

```
<element name="first" type="string" minOccurs="2" maxOccurs="2"/>

<element ref="target:first" maxOccurs="10"/>

<element name="location" minOccurs="0" maxOccurs="unbounded"/>
```


Default and Fixed Values

Default

```
<element name="last" type="string" default="Doe"/>
```

Fixed

```
<element name="version" type="string" fixed="1.0"/>
```

Element Wildcards

<any
minOccurs="*non negative number*"
maxOccurs="*non negative number* or *unbounded*"
namespace="*allowable namespaces*"
processContents="*lax* or *skip* or *strict*">

Value of namespace attr	Description
##any	Allows elements from all namespaces to be included as part of the wildcard. (This is default.)
##other	Allows elements from namespaces other than the targetNamespace to be included as part of the wildcard.
##targetNamespace	Allows elements from only the targetNamespace to be included as part of the wildcard.
##local	Allows any well-formed elements that are not qualified by a namespace to be included as part of the wildcard.
Whitespace-separated list of allowable namespace URIs	Allows elements from any listed namespaces to be included as part of the wildcard. Possible list values also include ##targetNamespace and ##local.

Element Wildcards

<any
minOccurs="*non negative number*"
maxOccurs="*non negative number* or *unbounded*"
namespace="*allowable namespaces*"
processContents="*lax* or *skip* or *strict*">

Value of processContents	Description
skip	• Skip validation of wildcard elements
lax	• Attempt to validate wildcard elements if it has access to a global XML Schema definition for them • No validity error raised if the definition not found
strict	• Attempt to validate wildcard elements • Validity error raised if a global XML Schema definition for the elements not found • Default value of processContents

Element Wildcards

Example

```
<complexType name="NameType">
  <sequence>
    <element ref="target:first"/>
    <element ref="target:middle"/>
    <element ref="target:last"/>
    <!-- allow any element from any namespace -->
    <any namespace="##any"
         processContents="lax"
         minOccurs="0"
         maxOccurs="unbounded"/>
  </sequence>
  <attribute name="title" type="string"/>
</complexType>
```


<complexType> Declarations

```
<complexType
 mixed="true or false"
 name="Name of complexType">
```

Example 1

```
<element name="name">
  <complexType>
    <sequence>
      <element name="first" type="string"/>
      <element name="middle" type="string"/>
      <element name="last" type="string"/>
    </sequence>
    <attribute name="title" type="string"/>
  </complexType>
</element>
```

<complexType> Declarations

mixed="*true*“ for mixed content

Example 2

```
<element name=“description“>
   <complexType mixed=“true”>
     <choice minOccurs="0" maxOccurs="unbounded">
       <element name="em" type="string"/>
       <element name="strong" type="string"/>
       <element name="br" type="string"/>
     </choice>
   </complexType>
</element>
```

In XML instance document:

```
<description>Jeff is a developer and author for Beginning XML
    <em>4th edition</em>.<br/>Jeff <strong>loves</strong> XML!
</description>
```


<complexType> Declarations

Example 3 – Empty element

```
<element name=“knows“>
   <complexType>
   </complexType>
</element>

<element name=“knows“>
   <complexType/>
</element>
```

Empty elements may have attributes

```
<element name=“knows“>
   <complexType>
     <attribute name=“contacts” type=“IDREFS”/>
   </complexType>
</element>
```

<group> Declarations

Example: name8.xsd

```
<group name="NameGroup">
  <sequence>
    <element name="first" type="string" minOccurs="1"
                                        maxOccurs="unbounded"/>
    <element name="middle" type="string" minOccurs="0" maxOccurs="1"/>
    <element name="last" type="string"/>
  </sequence>
</group>
```

```
<complexType name="NameType">
  <group ref="target:NameGroup"/>
  <attribute name="title" type="string"/>
</complexType>

<element name="name" type="target:NameType"/>
```


Content Models

- XML Schemas provide a greater flexibility than DTDs in specifying an element's content model
- Content models
  - <sequence> declaration
  - <choice> declaration
  - reference to a global <group> declaration
  - <all> declaration
- Each of the primary declaration may contain
  - Inner content model
  - Element declaration
  - Element wildcard

<sequence> Declarations

<sequence
 minOccurs="*non negative number*"
 maxOccurs="*non negative number* or *unbounded*">

Example

```
<complexType>
 <sequence>
  <element name="first" type="string" minOccurs="1"
                                        maxOccurs="unbounded"/>
  <element name="middle" type="string" minOccurs="0" maxOccurs="1"/>
  <element name="last" type="string"/>
 </sequence>
 <attribute name="title" type="string"/>
</complexType>
```


<choice> Declarations

<choice
 minOccurs="*non negative number*"
 maxOccurs="*non negative number or unbounded*">

Example 2

```
<choice>
  <element name="first" type="string" minOccurs="1"
     maxOccurs="unbounded"/>
  <element name="middle" type="string" minOccurs="0" maxOccurs="1"/>
  <element name="last" type="string"/>
</choice>
```

<group> References

```
<group
  ref="global group definition"
  minOccurs="non negative number"
  maxOccurs="non negative number or unbounded">
```

Example

```
<group name="NameGroup">
  <sequence>
    <element name="first" type="string"/>
    <element name="middle" type="string"/>
    <element name="last" type="string"/>
  </sequence>
</group>
<element name="name">
  <complexType>
    <group ref="target:NameGroup"/>
    <attribute name="title" type="string"/>
  </complexType>
</element>
```


<all>Declarations

- Elements can appear in any order
- Rules
  - The **<all>** declaration must be the only content model declaration that appears as a child of a <complexType> definition.
  - The **<all>** declaration may contain only <element> declarations as its children. It is not permitted to contain <sequence>, <choice>, or <group> declarations.
  - The **<all>** declaration's children may appear once each in the instance document. This means that within the <all> declaration the values for minOccurs is limited to 0 or 1 and for maxOccurs is 1.

<all> Declaration

Example

```
<element name="name">
 <complexType>
  <all>
   <element name="first" type="string"/>
   <element name="middle"
    type="string"/>
   <element name="last" type="string"/>
  </all>
  <attribute name="title" type="string"/>
 </complexType>
</element>
```

```
<first>John</first>
<middle>Fitzgerald</middle>
<last>Doe</last>
or
<first>John</first>
<last>Doe</last>
<middle>Fitzgerald</middle>
```

Let's examine contacts6.xsd

<attribute> Declarations

```
<attribute
  name="name of the attribute"
  type="global type"
  ref="global attribute declaration"
  form="qualified or unqualified"
  use="optional or prohibited or required"
  default="default value"
  fixed="fixed value"
  id=ID
  :
>
```

Simple type only, no complex type!

Creating a Local Type

Example

```
<attribute name="title">
  <simpleType>
    <!-- type information -->
  </simpleType>
</element>
```


Using a Global Type

Example (no prefix)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:contacts="http://www.example.com/contacts"
  targetNamespace="http://www.example.com/contacts"
  elementFormDefault="qualified">

  <simpleType name="KindType">
    <!-- type information -->
  </simpleType>

  <element name="phone">
    <complexType>
      <!-- content model information -->
      <attribute name="kind" type="contacts:KindType"/>
    </complexType>
  </element>
</schema>
```

Using a Global Type

Example (prefix)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.example.com/contacts"
  targetNamespace="http://www.example.com/contacts"
  elementFormDefault="qualified">

  <xs:simpleType name="KindType">
    <!-- type information -->
  </xs:simpleType>

  <xs:element name="phone">
    <xs:complexType>
      <!-- content model information -->
      <xs:attribute name="kind" type="KindType"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


Attribute Wildcards

Example

```
<complexType>
   <anyAttribute
   namespace="##local http://www.w3.org/XML/1998/namespace"
            processContents="lax"/>
</complexType>
```

Namespace Attribute Values in Wildcards

Value	Description
##any	Allows attributes from all namespaces to be included as part of the wildcard.
##other	Allows attributes from namespaces other than the targetNamespace to be included as part of the wildcard.
##targetNamespace	Allows attributes from only the targetNamespace to be included as part of the wildcard.
##local	Allows attributes that are not qualified by a namespace to be included as part of the wildcard.
Whitespace separated list of allowable namespace URIs	Allows attributes from any listed namespaces to be included as part of the wildcard. Possible list values also include ##targetNamespace and ##local.

Let's examine contacts7.xsd

<attributeGroup> Declarations

```
<attributeGroup
  name="name of global attribute group">
```

Example

```
<attributeGroup name="ContactAttributes">
  <!-- attribute declarations go here -->
</attributeGroup>

<attributeGroup ref="contacts:ContactAttributes"/>
```

Illegal <attributeGroup> Declarations

Example 1

```
<attributeGroup name="AttGroup1">
  <attributeGroup ref="target:AttGroup1"/>
</attributeGroup >
```

Example 2

```
<attributeGroup name="AttGroup1">
  <attributeGroup ref="target:AttGroup2"/>
</attributeGroup >
<attributeGroup name="AttGroup2">
  <attributeGroup ref="target:AttGroup1"/>
</attributeGroup >
```

Let's examine contacts8.xsd

Creating Elements with Simple Content

- simpleContent (in complexType only):
  - Contians extension or restriction on
    - simpleType
    - text-only complexType
  - Used to add attributes

- complexContent vs simpleContent
  - both must be in complexType
  - simpleContent cannot contain element

Creating Elements with Simple Content

Example (simple content): adding kind attribute to phone

```
<element name="phone">
    <complexType>
      <simpleContent>
        <!-- Specify type here -->
      </simpleContent>
    </complexType>
</element>
```


Creating Elements with Simple Content and Attributes

Example (simple content and attribute)

```
<element name="phone">
    <complexType>
      <simpleContent>
        <extension base="string">
          <attribute name="kind" type="string" default="Home" />
        </extension>
      </simpleContent>
    </complexType>
</element>
```

Data Types

Type	Description
string	Any character data
normalizedString	A whitespace normalized string where all spaces, tabs, carriage returns, and line feed characters are converted to single spaces
token	A string that does not contain sequences of two or more spaces, tabs, carriage returns, or line feed characters, and does not have leading or trailing spaces
byte	A numeric value from -128 to 127
unsignedByte	A numeric value from 0 to 255
base64Binary	Base64 encoded binary information
hexBinary	Hexadecimal encoded binary information
integer	A numeric value representing a whole number
positiveInteger	An integer whose value is greater than 0

See book for complete list.

Let's examine contacts9.xsd

User-Defined Data Types

<simpleType
  name="*name of the simpleType*"
  other attributes >

There are three primary derived types:

* Restriction types – restrictions on base type
* List types – list of items
* Union types – combining two or more types

<restriction> Declarations

<restriction base="name of simpleType to derive from">

Example

```
<attribute name="kind">
  <simpleType>
    <restriction base="string">
      <enumeration value="Home"/>
      <enumeration value="Work"/>
      <enumeration value="Cell"/>
      <enumeration value="Fax"/>
    </restriction>
  </simpleType>
</attribute>
```

enumeration is one of *constraining facets* (see next slide)

Let's examine contacts10.xsd after next slide

Constraint/Facet	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

<list> Declarations

<list
itemType="name of simpleType used for validating items in list">

- **A list of items separated by whitesspace**
  ➔ itemType cannot have whitespace
- **Example**

```
<simpleType name="ContactTagsType">
  <restriction base="string">
    <enumeration value="author"/>
    <enumeration value="xml"/>
    <enumeration value="poetry"/>
    <enumeration value="consultant"/>
    <enumeration value="CGI"/>
    <enumeration value="semantics"/>
  </restriction>
</simpleType>

<simpleType name="ContactTagsListType">
  <list itemType="contacts:ContactTagsType"/>
 </simpleType>
```


<union> Declarations

<union
memberTypes="white space separated list of types">

Example – in .xsd

```
<simpleType name="UnknownString">
   <restriction base="string">
     <enumeration value="Unknown"/>
   </restriction>
 </simpleType>

 <simpleType name="UnknownOrFloatType">
   <union memberTypes="float contacts:UnknownString"/>
 </simpleType>

<element name="latitude"
    type="contacts:UnknownStringOrFloatType"/>
<element name="longitude"
    type="contacts:UnknownStringOrFloatType"/>
```

<union> Declarations

Example – in .xml

Valid

<latitude>43.847156</latitude>
<longitude>Unknown</longitude>

Not valid

<latitude>unknown</latitude>
<longitude>43.847156 Unknown</longitude>

Let's examine contacts10.xsd

Creating a Schema from Multiple Documents

<import> Declarations

Combine XML Schemas from different targetNamespace

<import
namespace="*namespaceURI*"
schemaLocation="*schemaLocationURI*">

or

<include> Declarations

Combine XML Schemas to define a vocabulary

<include
schemaLocation="*schemaLocationURI*">

Let's examine contacts11.xsd and contacts12.xsd

Documenting XML Schemas

There are three ways

- Comments
- Attributes from other namespaces
- Annotations

Comments

Example

```
<!-- This complexType allows you to describe a person's name
   broken down by first, middle and last parts of the name. You
   can also specify a greeting by including the title attribute. -->
<complexType name="NameType">
  <!-- The NameGroup is a global group defined in this
     XML Schema. -->
  <group ref="target:NameGroup"/>
  <attribute name="title" type="string"/>
</complexType>
```

Attributes from Other Namespaces

Example

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:target="http://www.example.com/name"
  xmlns:doc="http://www.w3.org/documentation"
  targetNamespace="http://www.example.com/name"
  elementFormDefault="qualified">
 <group name="NameGroup">
   <sequence>
     <element name="first" type="string" minOccurs="1" maxOccurs="unbounded"/>
     <element name="middle" type="string" minOccurs="0" maxOccurs="1"/>
     <element name="last" type="string"/>
   </sequence>
 </group>
 <complexType name="NameType" doc:comments="This complexType allows you to
   describe a person's name broken down by first, middle and last parts of the
   name. You can also specify a greeting by including the title attribute.">
   <group ref="target:NameGroup" doc:comments="The NameGroup is a global
     group defined in this XML Schema."/>
   <attribute name="title" type="string"/>
 </complexType>
 <element name="name" type="target:NameType"/>
</schema>
```


Annotations

Example

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.com/name"
      xmlns:target="http://www.example.com/name" elementFormDefault="qualified">
 <annotation>
  <appinfo source="name-instance.xml"/>
  <documentation xmlns:html="http://www.w3.org/1999/xhtml">
   <html:p>
    The name vocabulary was created for a Chapter 2 sample. We have
    upgraded it to an <html:strong>XML Schema</html:strong>. The appinfo of this
    <html:pre>&lt;annotation&gt;</html:pre> element points to a sample XML file.
    The sample should be used <html:i>only as an example</html:i>
   </html:p>
  </documentation>
 </annotation>
 <element name="name" type="string">
  <annotation>
   <documentation source="name.html"/>
  </annotation>
 </element>
</schema>
```

Application information

Documentation information

Let's examine contacts13.xsd

Other Notes

- Is restriction the **opposite** of extension?
- The pattern facets in restriction
  - Regular expression
  - Example: Defining the Simple Type "SKU“

```
<!-- 3 digits followed by ‘-’ and then by 2 upper
     case letters -->
<xsd:simpleType name="SKU">
 <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{3}-[A-Z]{2}"/>
 </xsd:restriction>
</xsd:simpleType>
```

- XML Schema Part 0: Primer Second Edition
  http://www.w3.org/TR/xmlschema-0/

Examples of Regular Expressions

Expression	Match(es)
Chapter \d	Chapter 0, Chapter 1, Chapter 2 ....
Chapter\s\d	Chapter followed by a single whitespace character (space, tab, newline, etc.), followed by a single digit
Chapter\s\w	Chapter followed by a single whitespace character (space, tab, newline, etc.), followed by a word character
a*x	x, ax, aax, aaax ....
a?x	ax, x
a+x	ax, aax, aaax ....
(a|b)+x	ax, bx, aax, abx, bax, bbx, aaax, aabx, abax, abbx, baax, babx, bbax, bbbx, aaaax, ...
[abcde]x	ax, bx, cx, dx, ex
[a-e]x	ax, bx, cx, dx, ex
[a\-e]x	-x, ax, ex
[^0-9]x	any non-digit character followed by the character x
\Dx	any non-digit character followed by the character x
.x	any character followed by the character x
.*abc.*	1x2abc, abc1x2, z3456abchooray ....
ab{2}x	abbx
ab{2,4}x	abbx, abbbx, abbbbx
ab{2,}x	abbx, abbbx, abbbbx ....
(ab){2}x	ababx

Summary

*: 0 or more
+: 1 or more
?: 0 or 1
.: any
(…): group
| in ( ): or
[…]: class
-in [ ]: range
-^ in [ ]: not
\: escape
\d: digits
\D: non-digits
\s: whitespace
\w: word char