Schema documentation for swiML.xsd

november 28, 2024

Table of Contents

Namespace: "https://github.com/bartneck/swiML"

Schema(s)

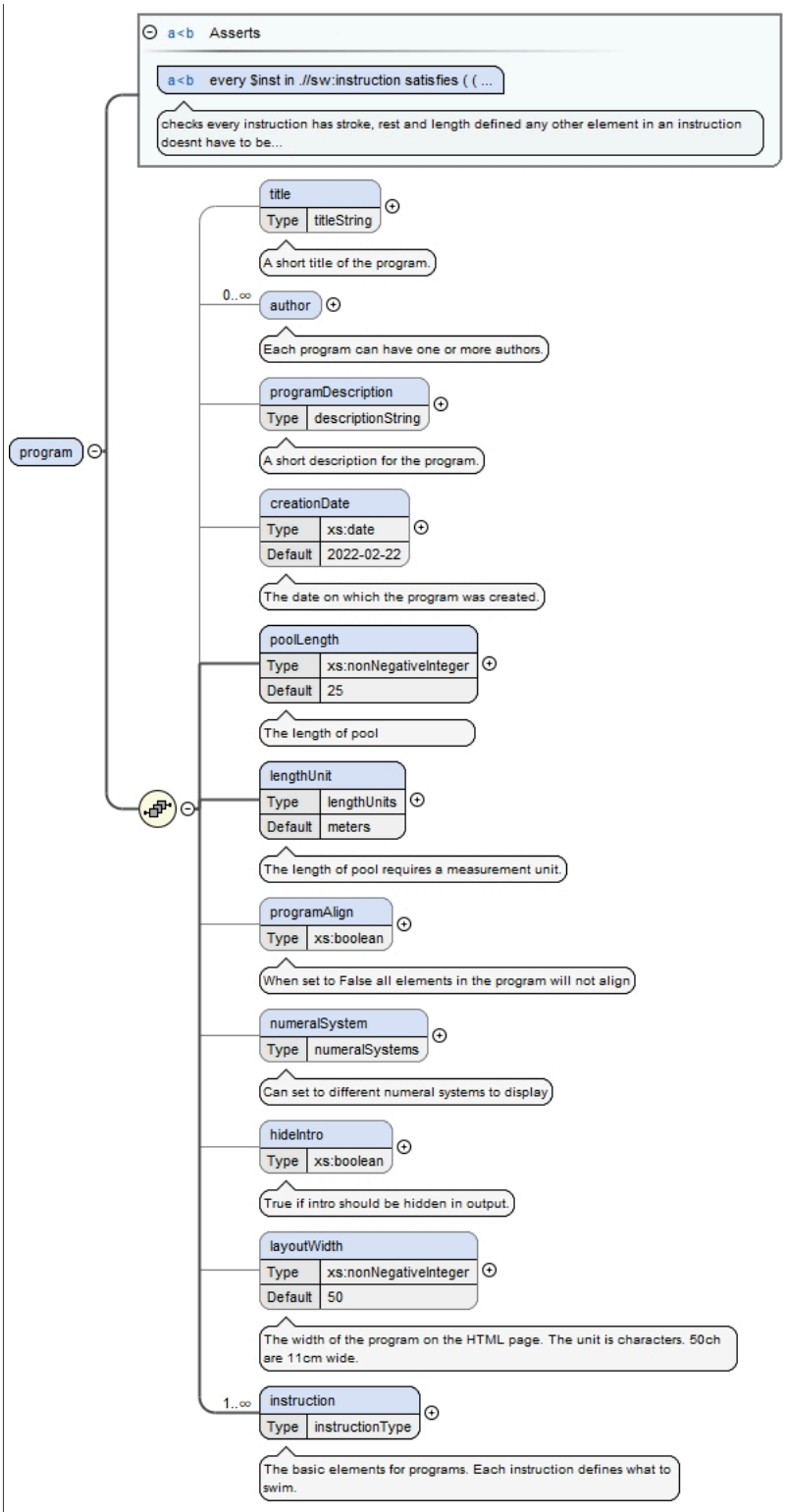
Main schema swiML.xsd

Namespace	https://github.com/bartneck/swiML	
Properties	attribute form default:	unqualified
	element form default:	qualified
	version:	2.3

Element(s)

Element program

Namespace	https://github.com/bartneck/swiML

Diagram	
Properties	content: complex
Model	title{0,1} , author* , programDescription{0,1} , creationDate{0,1} , poolLength , lengthUnit , programAlign{0,1} , numeralSystem{0,1} , hideIntro{0,1} , layoutWidth{0,1} , instruction+
Children	author, creationDate, hideIntro, instruction, layoutWidth, lengthUnit, numeralSystem, poolLength, programAlign, programDescription, title
Instance	<program xmlns="https://github.com/bartneck/swiML"> <title>{0,1}</title> <author>{0,unbounded}</author> <programDescription>{0,1}</programDescription> <creationDate>{0,1}</creationDate> <poolLength>{1,1}</poolLength>

```
  <lengthUnit>{1,1}</lengthUnit>
  <programAlign>{0,1}</programAlign>
  <numeralSystem>{0,1}</numeralSystem>
  <hideIntro>{0,1}</hideIntro>
  <layoutWidth>{0,1}</layoutWidth>
  <instruction>{1,unbounded}</instruction>
</program>
```

Asserts

Test	XPath default namespace
every $inst in .//sw:instruction satisfies (($inst/ancestor-or-self::*/sw:stroke and $inst/ancestor-or-self::*/sw:length) or $inst/sw:repetition or $inst/sw:continue or $inst/sw:pyramid or $inst/sw:segmentName)	

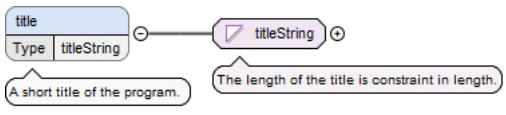
```
checks every instruction has stroke, rest and length defined
any other element in an instruction doesnt have to be defined
```

Source

```
<xs:element name="program">
  <!-- ======================================================= -->
  <!-- The meta information for each program -->
  <!-- ======================================================= -->
  <xs:complexType>
    <xs:sequence>
      <!-- The title of the program -->
      <xs:element name="title" type="titleString" minOccurs="0" maxOccurs="1">
        <xs:annotation>
          <xs:documentation>A short title of the program.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <!-- The author(s) of the program -->
      <xs:element maxOccurs="unbounded" minOccurs="0" name="author">
        <xs:annotation>
          <xs:documentation>Each program can have one or more authors.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="firstName" minOccurs="1" type="xs:string">
              <xs:annotation>
                <xs:documentation>The first name of the author. Can contain middle names if
 necessary.</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="lastName" minOccurs="1" type="xs:string">
              <xs:annotation>
                <xs:documentation>The last name of the author.</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element minOccurs="0" name="email" type="emailAddress">
              <xs:annotation>
                <xs:documentation>The email address of the author (optional).</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <!-- The description of the program -->
      <xs:element name="programDescription" type="descriptionString" minOccurs="0" maxOccurs="1">
        <xs:annotation>
          <xs:documentation>A short description for the program.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <!-- The date -->
      <xs:element minOccurs="0" name="creationDate" type="xs:date" maxOccurs="1"
 default="2022-02-22">
        <xs:annotation>
          <xs:documentation>The date on which the program was created.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="poolLength" minOccurs="1" maxOccurs="1" type="xs:nonNegativeInteger"
 default="25">
        <xs:annotation>
          <xs:documentation>The length of pool</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="lengthUnit" minOccurs="1" maxOccurs="1" type="lengthUnits" default="meters">
        <xs:annotation>
          <xs:documentation>The length of pool requires a measurement unit.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="programAlign" minOccurs="0" maxOccurs="1" type="xs:boolean">
        <xs:annotation>
```

```
          <xs:documentation>When set to False all elements in the program will not align</
xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="numeralSystem" minOccurs="0" maxOccurs="1" type="numeralSystems">
        <xs:annotation>
          <xs:documentation>Can set to different numeral systems to display</xs:documentation>
        </xs:annotation>
      </xs:element>
      <!-- Element to hide the intro text -->
      <xs:element name="hideIntro" minOccurs="0" maxOccurs="1" type="xs:boolean">
        <xs:annotation>
          <xs:documentation>True if intro should be hidden in output.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <!-- Element to set the width of the program in HTML -->
      <!-- The unit is characters. 50ch are 11cm wide -->
      <xs:element name="layoutWidth" minOccurs="0" maxOccurs="1" type="xs:nonNegativeInteger"
 default="50">
        <xs:annotation>
          <xs:documentation>The width of the program on the HTML page. The unit is characters. 50ch
 are 11cm wide.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <!-- ================================================== -->
      <!-- The main element(s) for each program. Each instruction can contain instructions. This is
 recursion. -->
      <!-- This is the main recursive element for a program -->
      <!-- ================================================== -->
      <xs:element name="instruction" type="instructionType" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>The basic elements for programs. Each instruction defines what to
 swim.</xs:documentation>
        </xs:annotation>
        <xs:unique name="mainEquipmentUnique">
          <xs:annotation>
            <xs:documentation>Ensures all equipment values in an instruction are unique</
xs:documentation>
          </xs:annotation>
          <xs:selector xpath="./sw:equipment"/>
          <xs:field xpath="."/>
        </xs:unique>
      </xs:element>
    </xs:sequence>
    <xs:assert test="                    every $inst in .//sw:instruction
  satisfies (                          ($inst/ancestor-or-self::*/sw:stroke
 and $inst/ancestor-or-self::*/sw:length)                        or $inst/sw:repetition
             or $inst/sw:continue                          or $inst/sw:pyramid
  or $inst/sw:segmentName)             ">
      <xs:annotation>
        <xs:documentation>checks every instruction has stroke, rest and length defined any other
 element in an instruction doesnt have to be defined</xs:documentation>
      </xs:annotation>
    </xs:assert>
  </xs:complexType>
</xs:element>
```

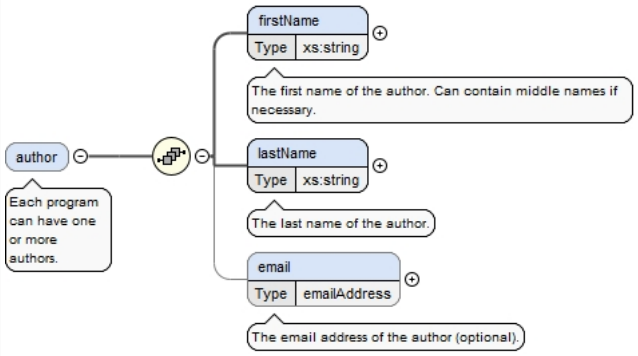
Element `program / title`

Namespace	https://github.com/bartneck/swiML
Annotations	`A short title of the program.`
Diagram	
Type	titleString
Properties	content: `simple` minOccurs: `0` maxOccurs: `1`
Facets	maxLength `60`

Source	`<xs:element name="title" type="titleString" minOccurs="0" maxOccurs="1">`
	`  <xs:annotation>`
	`  <xs:documentation>A short title of the program.</xs:documentation>`
	`  </xs:annotation>`

```
</xs:element>
```

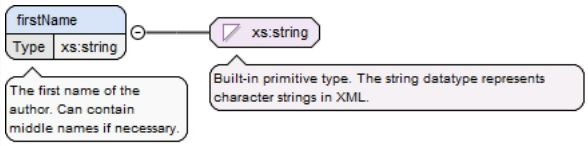
Element program / author

Namespace	https://github.com/bartneck/swiML
Annotations	Each program can have one or more authors.
Diagram	
Properties	content: complex minOccurs: 0 maxOccurs: unbounded
Model	firstName , lastName , email{0,1}
Children	email, firstName, lastName

Instance	`<author xmlns="https://github.com/bartneck/swiML">`
	`  <firstName>{1,1}</firstName>`
	`  <lastName>{1,1}</lastName>`
	`  <email>{0,1}</email>`
	`</author>`

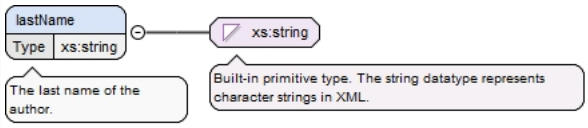
Source	`<xs:element maxOccurs="unbounded" minOccurs="0" name="author">`
	`  <xs:annotation>`
	`  <xs:documentation>Each program can have one or more authors.</xs:documentation>`
	`  </xs:annotation>`
	`  <xs:complexType>`
	`  <xs:sequence>`
	`  <xs:element name="firstName" minOccurs="1" type="xs:string">`
	`  <xs:annotation>`
	`  <xs:documentation>The first name of the author. Can contain middle names if necessary.</xs:documentation>`
	`  </xs:annotation>`
	`  </xs:element>`
	`  <xs:element name="lastName" minOccurs="1" type="xs:string">`
	`  <xs:annotation>`
	`  <xs:documentation>The last name of the author.</xs:documentation>`
	`  </xs:annotation>`
	`  </xs:element>`
	`  <xs:element minOccurs="0" name="email" type="emailAddress">`
	`  <xs:annotation>`
	`  <xs:documentation>The email address of the author (optional).</xs:documentation>`
	`  </xs:annotation>`
	`  </xs:element>`
	`  </xs:sequence>`
	`  </xs:complexType>`
	`</xs:element>`

Element program / author / firstName

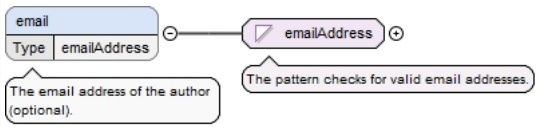
Namespace	https://github.com/bartneck/swiML
Annotations	The first name of the author. Can contain middle names if necessary.
Diagram	
Type	xs:string

Properties	content: simple minOccurs: 1
Source	<xs:element name="firstName" minOccurs="1" type="xs:string"> <xs:annotation> <xs:documentation>The first name of the author. Can contain middle names if necessary.</ xs:documentation> </xs:annotation> </xs:element>

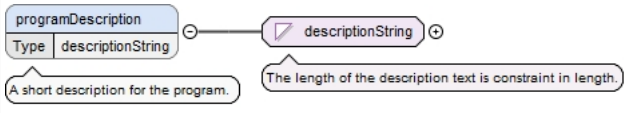
Element program / author / lastName

Namespace	https://github.com/bartneck/swiML
Annotations	The last name of the author.
Diagram	
Type	xs:string
Properties	content: simple minOccurs: 1
Source	<xs:element name="lastName" minOccurs="1" type="xs:string"> <xs:annotation> <xs:documentation>The last name of the author.</xs:documentation> </xs:annotation> </xs:element>

Element program / author / email

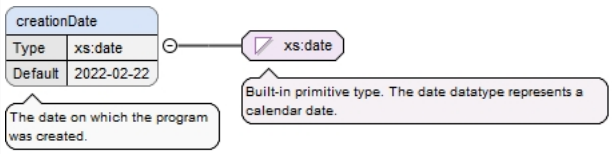
Namespace	https://github.com/bartneck/swiML
Annotations	The email address of the author (optional).
Diagram	
Type	emailAddress
Properties	content: simple minOccurs: 0
Facets	pattern [^@]+@[^\.]+\..+
Source	<xs:element minOccurs="0" name="email" type="emailAddress"> <xs:annotation> <xs:documentation>The email address of the author (optional).</xs:documentation> </xs:annotation> </xs:element>

Element program / programDescription

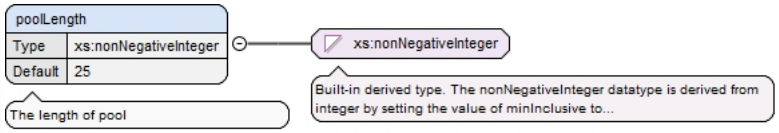
Namespace	https://github.com/bartneck/swiML
Annotations	A short description for the program.
Diagram	
Type	descriptionString
Properties	content: simple minOccurs: 0 maxOccurs: 1

Facets	maxLength 400
Source	<xs:element name="programDescription" type="descriptionString" minOccurs="0" maxOccurs="1"> <xs:annotation> <xs:documentation>A short description for the program.</xs:documentation> </xs:annotation> </xs:element>

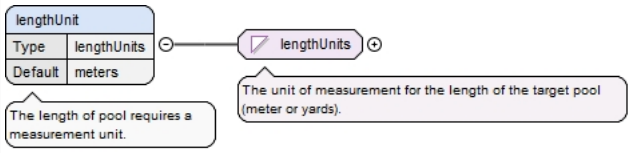
Element program / creationDate

Namespace	https://github.com/bartneck/swiML
Annotations	The date on which the program was created.
Diagram	
Type	xs:date
Properties	content: simple minOccurs: 0 maxOccurs: 1 default: 2022-02-22
Source	<xs:element minOccurs="0" name="creationDate" type="xs:date" maxOccurs="1" default="2022-02-22"> <xs:annotation> <xs:documentation>The date on which the program was created.</xs:documentation> </xs:annotation> </xs:element>

Element program / poolLength

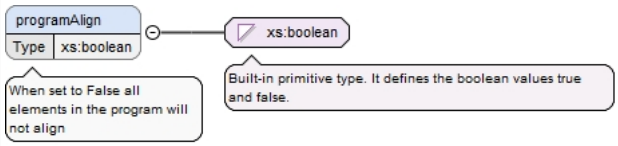
Namespace	https://github.com/bartneck/swiML
Annotations	The length of pool
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple minOccurs: 1 maxOccurs: 1 default: 25
Source	<xs:element name="poolLength" minOccurs="1" maxOccurs="1" type="xs:nonNegativeInteger" default="25"> <xs:annotation> <xs:documentation>The length of pool</xs:documentation> </xs:annotation> </xs:element>

Element program / lengthUnit

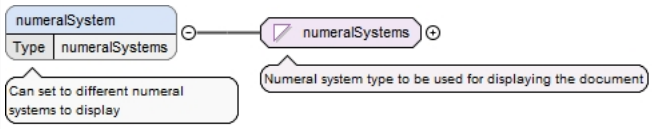
Namespace	https://github.com/bartneck/swiML
Annotations	The length of pool requires a measurement unit.
Diagram	
Type	lengthUnits

Properties	content:	simple
	minOccurs:	1
	maxOccurs:	1
	default:	meters
Facets	enumeration	meters
	enumeration	kilometers
	enumeration	miles
	enumeration	yards
Source	<xs:element name="lengthUnit" minOccurs="1" maxOccurs="1" type="lengthUnits" default="meters"> <xs:annotation> <xs:documentation>The length of pool requires a measurement unit.</xs:documentation> </xs:annotation> </xs:element>	

Element program / programAlign

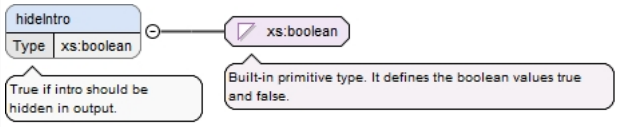
Namespace	https://github.com/bartneck/swiML
Annotations	When set to False all elements in the program will not align
Diagram	
Type	xs:boolean
Properties	content: simple minOccurs: 0 maxOccurs: 1
Source	<xs:element name="programAlign" minOccurs="0" maxOccurs="1" type="xs:boolean"> <xs:annotation> <xs:documentation>When set to False all elements in the program will not align</ xs:documentation> </xs:annotation> </xs:element>

Element program / numeralSystem

Namespace	https://github.com/bartneck/swiML
Annotations	Can set to different numeral systems to display
Diagram	
Type	numeralSystems
Properties	content: simple minOccurs: 0 maxOccurs: 1
Facets	enumeration decimal enumeration roman
Source	<xs:element name="numeralSystem" minOccurs="0" maxOccurs="1" type="numeralSystems"> <xs:annotation> <xs:documentation>Can set to different numeral systems to display</xs:documentation> </xs:annotation> </xs:element>

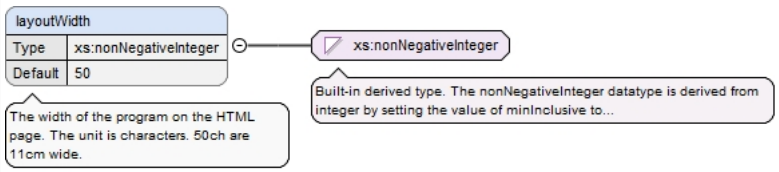
Element program / hideIntro

Namespace	https://github.com/bartneck/swiML

Annotations	True if intro should be hidden in output.
Diagram	
Type	xs:boolean
Properties	content: simple minOccurs: 0 maxOccurs: 1

Source	`<xs:element name="hideIntro" minOccurs="0" maxOccurs="1" type="xs:boolean">`
	`  <xs:annotation>`
	`  <xs:documentation>True if intro should be hidden in output.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element program / layoutWidth

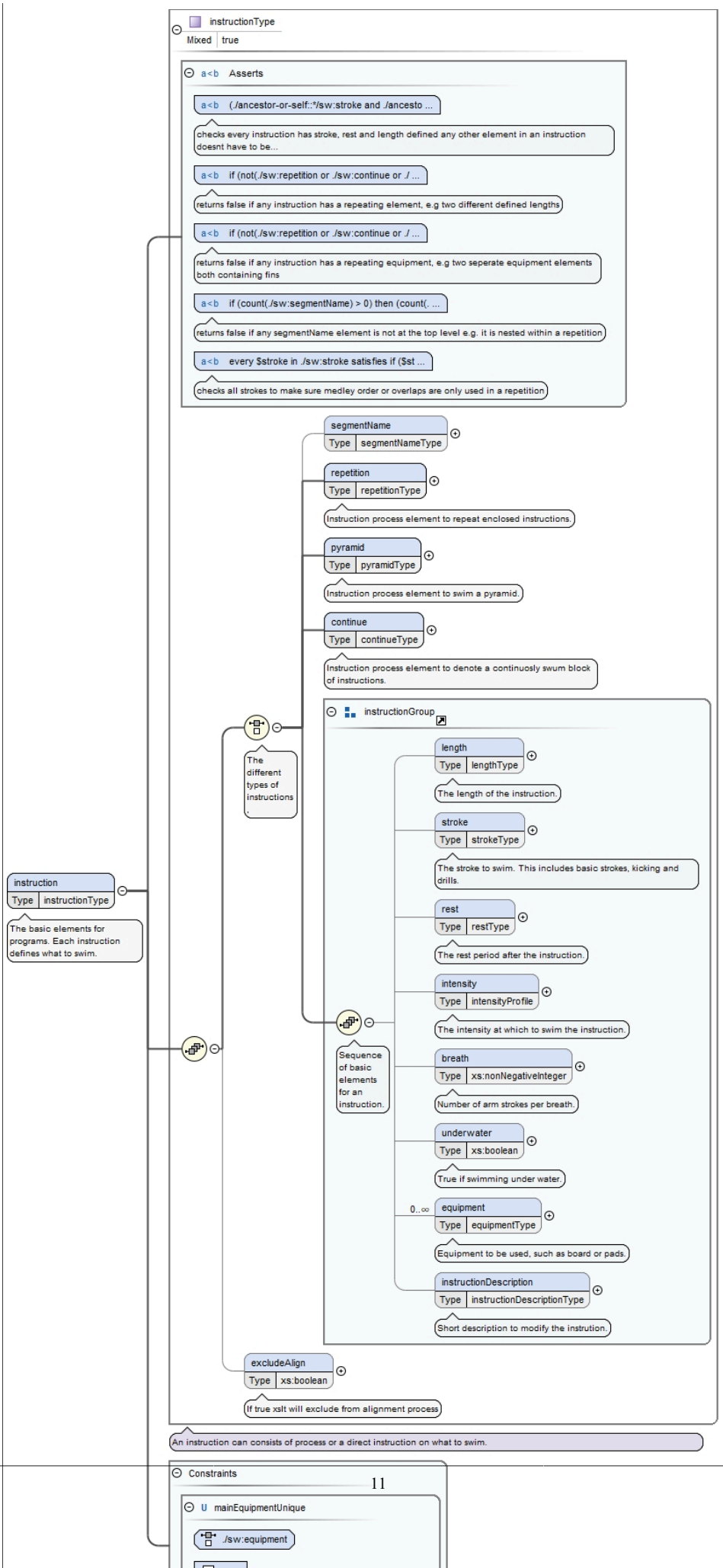
Namespace	https://github.com/bartneck/swiML
Annotations	The width of the program on the HTML page. The unit is characters. 50ch are 11cm wide.
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple minOccurs: 0 maxOccurs: 1 default: 50

Source	`<xs:element name="layoutWidth" minOccurs="0" maxOccurs="1" type="xs:nonNegativeInteger"`
	` default="50">`
	`  <xs:annotation>`
	`  <xs:documentation>The width of the program on the HTML page. The unit is characters. 50ch are`
	` 11cm wide.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element program / instruction

Namespace	https://github.com/bartneck/swiML
Annotations	The basic elements for programs. Each instruction defines what to swim.

Diagram


instructionType
Mixed true
a<b Asserts
a<b (./ancestor-or-self::*/sw:stroke and ./ancesto ...
checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be...
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating element, e.g two different defined lengths
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins
a<b if (count(./sw:segmentName) > 0) then (count(. ...
returns false if any segmentName element is not at the top level e.g. it is nested within a repetition
a<b every $stroke in ./sw:stroke satisfies if ($st ...
checks all strokes to make sure medley order or overlaps are only used in a repetition
segmentName
Type segmentNameType
repetition
Type repetitionType
Instruction process element to repeat enclosed instructions.
pyramid
Type pyramidType
Instruction process element to swim a pyramid.
continue
Type continueType
Instruction process element to denote a continuosly swum block of instructions.
instructionGroup
The different types of instructions.
length
Type lengthType
The length of the instruction.
stroke
Type strokeType
The stroke to swim. This includes basic strokes, kicking and drills.
rest
Type restType
The rest period after the instruction.
intensity
Type intensityProfile
The intensity at which to swim the instruction.
Sequence of basic elements for an instruction.
breath
Type xs:nonNegativeInteger
Number of arm strokes per breath.
underwater
Type xs:boolean
True if swimming under water.
0..∞
equipment
Type equipmentType
Equipment to be used, such as board or pads.
instructionDescription
Type instructionDescriptionType
Short description to modify the instrution.
instruction
Type instructionType
The basic elements for programs. Each instruction defines what to swim.
excludeAlign
Type xs:boolean
If true xslt will exclude from alignment process
An instruction can consists of process or a direct instruction on what to swim.
Constraints
U mainEquipmentUnique
/sw:equipment

Type	instructionType
Properties	content: complex minOccurs: 1 maxOccurs: unbounded mixed: true
Model	(segmentName{0,1} | repetition | pyramid | continue | (length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1})) , excludeAlign{0,1}
Children	breath, continue, equipment, excludeAlign, instructionDescription, intensity, length, pyramid, repetition, rest, segmentName, stroke, underwater

Instance

```
<instruction xmlns="https://github.com/bartneck/swiML">
  <segmentName>{0,1}</segmentName>
  <repetition>{1,1}</repetition>
  <pyramid>{1,1}</pyramid>
  <continue>{1,1}</continue>
  <length>{0,1}</length>
  <stroke>{0,1}</stroke>
  <rest>{0,1}</rest>
  <intensity>{0,1}</intensity>
  <breath>{0,1}</breath>
  <underwater>{0,1}</underwater>
  <equipment>{0,unbounded}</equipment>
  <instructionDescription>{0,1}</instructionDescription>
  <excludeAlign>{0,1}</excludeAlign>
</instruction>
```

Asserts

Test	XPath default namespace
(./ancestor-or-self::*/sw:stroke and ./ancestor-or-self::*/sw:length) or ./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName	
`checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be defined`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./* satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'length' or name() = 'stroke' or name() = 'rest' or name() = 'intensity' or name() = 'breath' or name() = 'underwater'] satisfies not(name($element) = name($match)))) else (true())	
`returns false if any instruction has a repeating element, e.g two different defined lengths`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./*[name() = 'equipment'] satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'equipment'] satisfies not($element/text() = $match/text()))) else (true())	
`returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins`	
if (count(./sw:segmentName) > 0) then (count(./sw:segmentName/../../ancestor::*) = 0) else (true())	
`returns false if any segmentName element is not at the top level e.g. it is nested within a repetition`	
every $stroke in ./sw:stroke satisfies if ($stroke/sw:standardStroke = 'individualMedleyOverlap' or $stroke/sw:standardStroke = 'individualMedleyOrder' or $stroke/sw:standardStroke = 'reverseIndividualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOverlap' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'reverseIndividualMedleyOrder' or$stroke/sw:drill/sw:drillStroke = 'individualMedleyOverlap' or $stroke/sw:drill/sw:drillStroke = 'individualMedleyOrder' or $stroke/sw:drill/sw:drillStroke = 'reverseIndividualMedleyOrder') then ($stroke/ancestor::*/sw:repetition) or ($stroke/ancestor::*/sw:continue/sw:continueLength) else ($stroke/parent::*)	

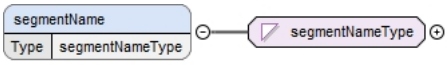
	Test	**XPath default namespace**
	checks all strokes to make sure medley order or overlaps are only used in a repetition	

Identity constraints	**QName**	**Type**	**Refer**	**Selector**	**Field(s)**
	mainEquipmentUnique	unique		./sw:equipment	.

Source

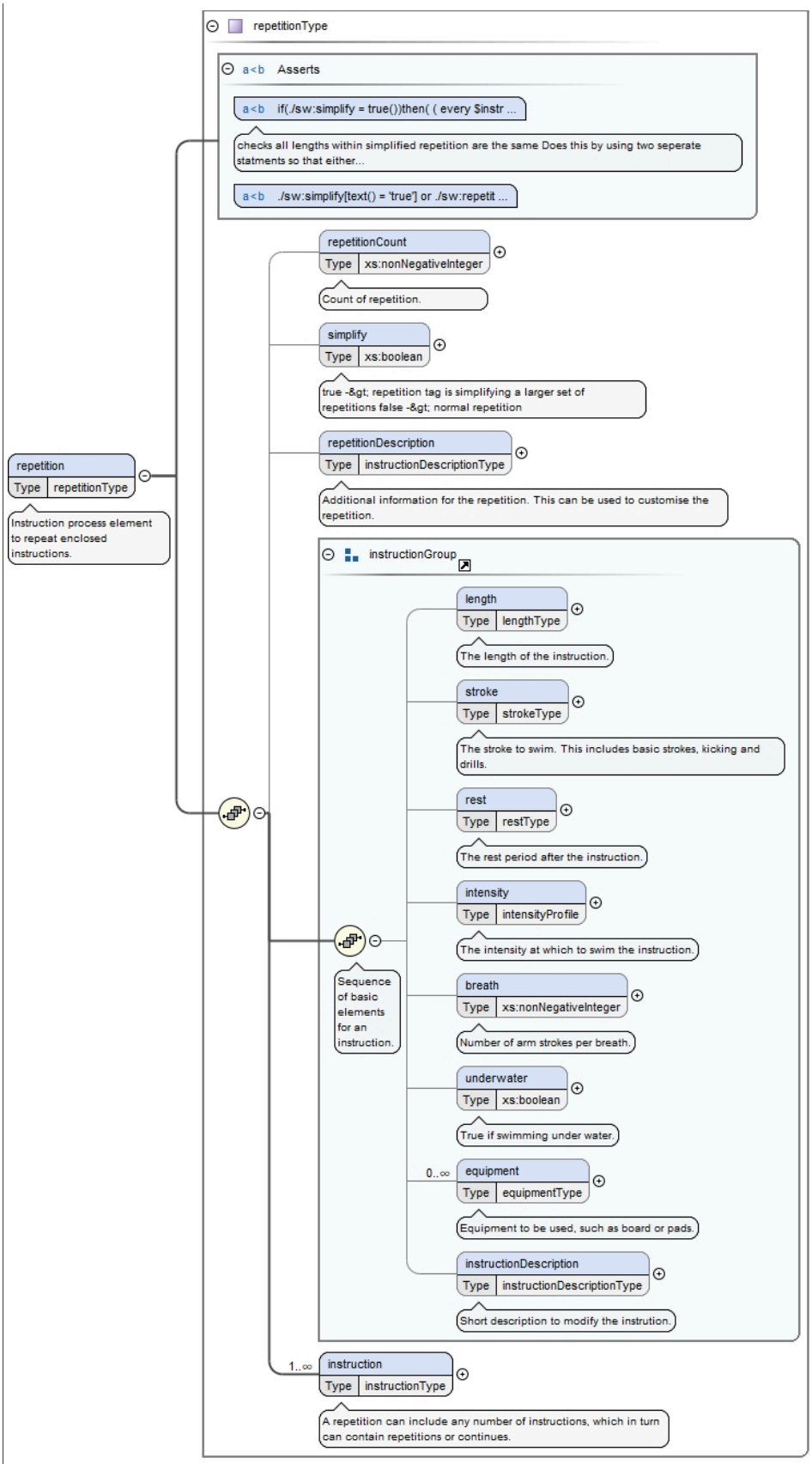
```
<xs:element name="instruction" type="instructionType" minOccurs="1" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>The basic elements for programs. Each instruction defines what to swim.</
xs:documentation>
  </xs:annotation>
  <xs:unique name="mainEquipmentUnique">
    <xs:annotation>
      <xs:documentation>Ensures all equipment values in an instruction are unique</xs:documentation>
    </xs:annotation>
    <xs:selector xpath="./sw:equipment"/>
    <xs:field xpath="."/>
  </xs:unique>
</xs:element>
```

Element `instructionType / segmentName`

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	segmentNameType
Properties	content: simple minOccurs: 0 maxOccurs: 1
Facets	maxLength 50
Source	`<xs:element name="segmentName" minOccurs="0" maxOccurs="1" type="segmentNameType"/>`

Element `instructionType / repetition`

Namespace	https://github.com/bartneck/swiML
Annotations	Instruction process element to repeat enclosed instructions.

Diagram	
Type	repetitionType
Properties	content: complex
Model	repetitionCount{0,1} , simplify{0,1} , repetitionDescription{0,1} , length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1} , instruction+

Children	breath, equipment, instruction, instructionDescription, intensity, length, repetitionCount, repetitionDescription, rest, simplify, stroke, underwater

Instance

```
<repetition xmlns="https://github.com/bartneck/swiML">
  <repetitionCount>{0,1}</repetitionCount>
  <simplify>{0,1}</simplify>
  <repetitionDescription>{0,1}</repetitionDescription>
  <length>{0,1}</length>
  <stroke>{0,1}</stroke>
  <rest>{0,1}</rest>
  <intensity>{0,1}</intensity>
  <breath>{0,1}</breath>
  <underwater>{0,1}</underwater>
  <equipment>{0,unbounded}</equipment>
  <instructionDescription>{0,1}</instructionDescription>
  <instruction>{1,unbounded}</instruction>
</repetition>
```

Asserts

Test	XPath default namespace
if(./sw:simplify = true())then((every $instruction in ./sw:instruction[not(./sw:pyramid or ./sw:segmentName)] satisfies((if($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) then(if(count($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) = 1) then(number(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance)) else(sum(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length \| ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance))) else(0)+(if($instruction/descendant-or-self::sw:continueLength) then(number($instruction/descendant-or-self::sw:continueLength)) else(0))) = number((((./descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance) | (./descendant-or-self::sw:continueLength))[1])))or(every $instruction in ./sw:instruction[not(./sw:pyramid or ./sw:segmentName)] satisfies((if($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) then(if(count($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) = 1) then(number(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps)) else(sum(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length \| ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps))) else(0)+(if($instruction/descendant-or-self::sw:continueLength) then(number($instruction/descendant-or-self::sw:continueLength)) else(0))) = number((((./descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps) | (./descendant-or-self::sw:continueLength))[1]))))else(true())	

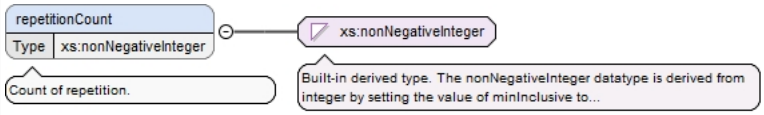
```
checks all lengths within simplified repetition are the same
Does this by using two seperate statments so that either all lengthAsDistances are the same or all
 lengthAsLaps
checks that every instruction, repetition and continue has the same base length e.g. 100m , 4x100m
 and 50,50 swim as 100 ; these are all of length 100 so can be simplified in the same repetition
has to choose whether to use continueLength or add up all instructions within a continue
```

	Test	XPath default namespace
	./sw:simplify[text() = 'true'] or ./sw:repetitionCount and not(./sw:simplify[text() = 'true'] and ./sw:repetitionCount)	

Source

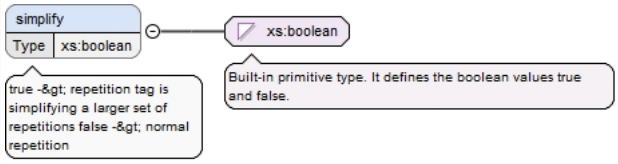
```
<xs:element name="repetition" type="repetitionType">
  <xs:annotation>
    <xs:documentation>Instruction process element to repeat enclosed instructions.</
xs:documentation>
  </xs:annotation>
</xs:element>
```

Element repetitionType / repetitionCount

Namespace	https://github.com/bartneck/swiML
Annotations	Count of repetition.
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple minOccurs: 0 maxOccurs: 1
Source	

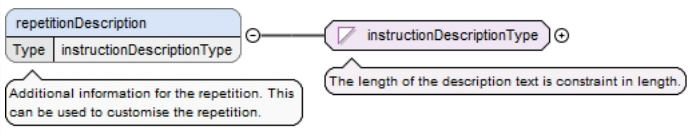

```
<xs:element name="repetitionCount" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>Count of repetition.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Element repetitionType / simplify

Namespace	https://github.com/bartneck/swiML
Annotations	true -> repetition tag is simplifying a larger set of repetitions false -> normal repetition
Diagram	
Type	xs:boolean
Properties	content: simple minOccurs: 0 maxOccurs: 1
Source	

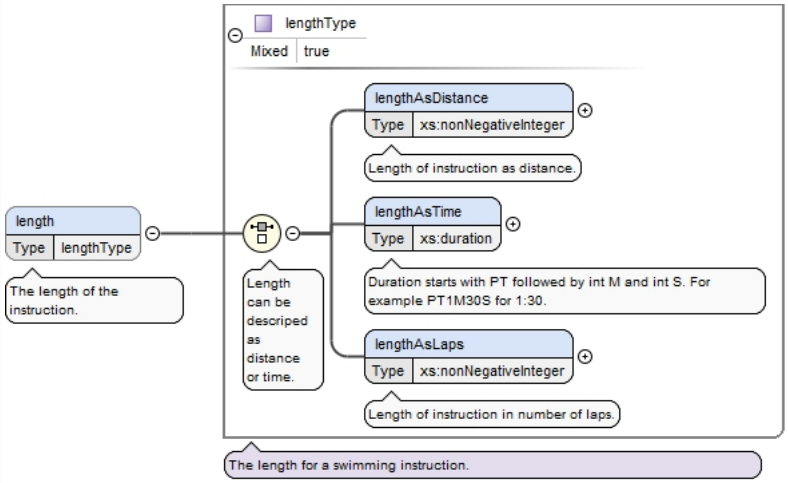

```
<xs:element name="simplify" minOccurs="0" maxOccurs="1" type="xs:boolean">
  <xs:annotation>
    <xs:documentation>true -> repetition tag is simplifying a larger set of repetitions false ->
 normal repetition</xs:documentation>
  </xs:annotation>
</xs:element>
```

Element repetitionType / repetitionDescription

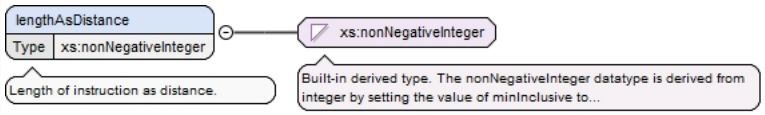
Namespace	https://github.com/bartneck/swiML
Annotations	Additional information for the repetition. This can be used to customise the repetition.
Diagram	
Type	instructionDescriptionType
Properties	content: simple

	minOccurs:	0
	maxOccurs:	1
Facets	maxLength	100
Source	<xs:element name="repetitionDescription" minOccurs="0" maxOccurs="1" type="instructionDescriptionType"> <xs:annotation> <xs:documentation>Additional information for the repetition. This can be used to customise the repetition.</xs:documentation> </xs:annotation> </xs:element>	

Element instructionGroup / length

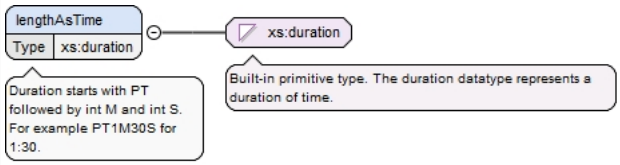
Namespace	https://github.com/bartneck/swiML
Annotations	The length of the instruction.
Diagram	
Type	lengthType
Properties	content: complex minOccurs: 0 maxOccurs: 1 mixed: true
Model	lengthAsDistance | lengthAsTime | lengthAsLaps
Children	lengthAsDistance, lengthAsLaps, lengthAsTime
Instance	<length xmlns="https://github.com/bartneck/swiML"> <lengthAsDistance>{1,1}</lengthAsDistance> <lengthAsTime>{1,1}</lengthAsTime> <lengthAsLaps>{1,1}</lengthAsLaps> </length>
Source	<xs:element name="length" minOccurs="0" maxOccurs="1" type="lengthType"> <xs:annotation> <xs:documentation>The length of the instruction.</xs:documentation> </xs:annotation> </xs:element>

Element lengthType / lengthAsDistance

Namespace	https://github.com/bartneck/swiML
Annotations	Length of instruction as distance.
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple
Source	<xs:element name="lengthAsDistance" type="xs:nonNegativeInteger">

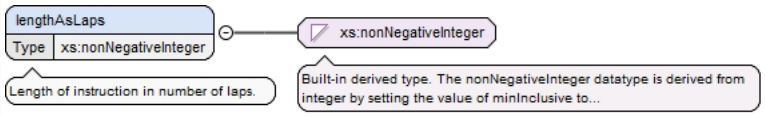
```
  <xs:annotation>
    <xs:documentation>Length of instruction as distance.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Element lengthType / lengthAsTime

Namespace	https://github.com/bartneck/swiML
Annotations	Duration starts with PT followed by int M and int S. For example PT1M30S for 1:30.
Diagram	
Type	xs:duration
Properties	content: simple

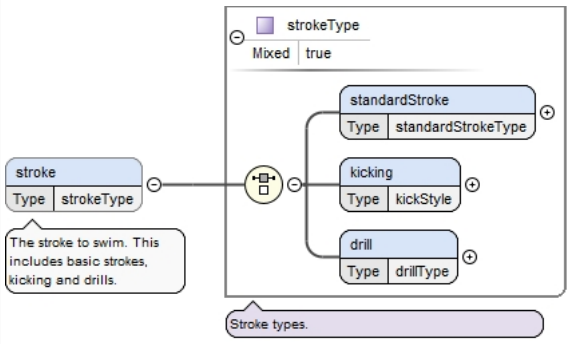
Source	`<xs:element name="lengthAsTime" type="xs:duration">`
	`  <xs:annotation>`
	`  <xs:documentation>Duration starts with PT followed by int M and int S. For example PT1M30S for 1:30.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element lengthType / lengthAsLaps

Namespace	https://github.com/bartneck/swiML
Annotations	Length of instruction in number of laps.
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple

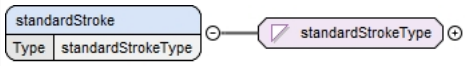
Source	`<xs:element name="lengthAsLaps" type="xs:nonNegativeInteger">`
	`  <xs:annotation>`
	`  <xs:documentation>Length of instruction in number of laps.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element instructionGroup / stroke

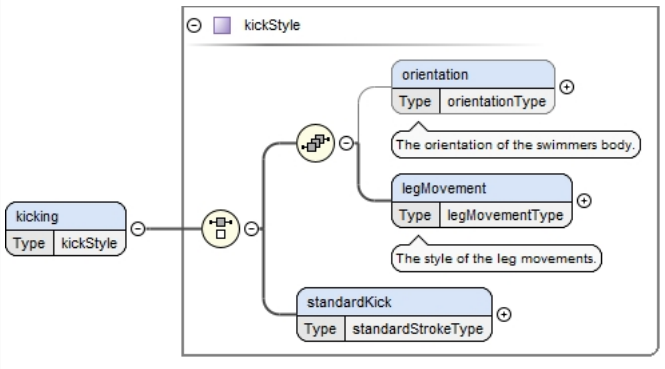
Namespace	https://github.com/bartneck/swiML
Annotations	The stroke to swim. This includes basic strokes, kicking and drills.
Diagram	
Type	strokeType
Properties	content: complex minOccurs: 0 maxOccurs: 1 mixed: true

Model	standardStroke | kicking | drill
Children	drill, kicking, standardStroke
Instance	<stroke xmlns="https://github.com/bartneck/swiML"> <standardStroke>{1,1}</standardStroke> <kicking>{1,1}</kicking> <drill>{1,1}</drill> </stroke>
Source	<xs:element name="stroke" minOccurs="0" maxOccurs="1" type="strokeType"> <xs:annotation> <xs:documentation>The stroke to swim. This includes basic strokes, kicking and drills.</ xs:documentation> </xs:annotation> </xs:element>

Element strokeType / standardStroke

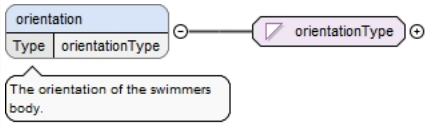
Namespace	https://github.com/bartneck/swiML	
Diagram		
Type	standardStrokeType	
Properties	content:	simple
Facets	enumeration	butterfly
	enumeration	backstroke
	enumeration	breaststroke
	enumeration	freestyle
	enumeration	individualMedley
	enumeration	reverseIndividualMedley
	enumeration	individualMedleyOverlap
	enumeration	individualMedleyOrder
	enumeration	reverseIndividualMedley- Order
	enumeration	any
	enumeration	nr1
	enumeration	nr2
	enumeration	nr3
	enumeration	nr4
	enumeration	notButterfly
	enumeration	notBackstroke
	enumeration	notBreaststroke
	enumeration	notFreestyle
Source	<xs:element name="standardStroke" type="standardStrokeType"/>	

Element strokeType / kicking

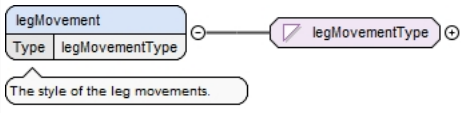
Namespace	https://github.com/bartneck/swiML
Diagram	

Type	kickStyle
Properties	content: complex
Model	(orientation{0,1} , legMovement) | standardKick
Children	legMovement, orientation, standardKick
Instance	<kicking xmlns="https://github.com/bartneck/swiML"> <orientation>{0,1}</orientation> <legMovement>{1,1}</legMovement> <standardKick>{1,1}</standardKick> </kicking>
Source	<xs:element name="kicking" type="kickStyle"/>

Element kickStyle / orientation

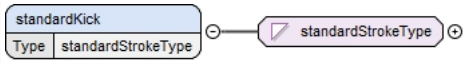
Namespace	https://github.com/bartneck/swiML
Annotations	The orientation of the swimmers body.
Diagram	
Type	orientationType
Properties	content: simple minOccurs: 0 maxOccurs: 1
Facets	enumeration front enumeration back enumeration left enumeration right enumeration side enumeration vertical enumeration waka
Source	<xs:element name="orientation" type="orientationType" minOccurs="0" maxOccurs="1"> <xs:annotation> <xs:documentation>The orientation of the swimmers body.</xs:documentation> </xs:annotation> </xs:element>

Element kickStyle / legMovement

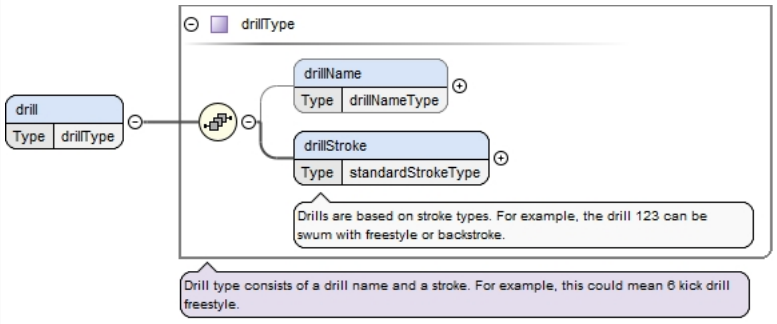
Namespace	https://github.com/bartneck/swiML
Annotations	The style of the leg movements.
Diagram	
Type	legMovementType
Properties	content: simple minOccurs: 1 maxOccurs: 1
Facets	enumeration flutter enumeration dolphin enumeration scissor
Source	<xs:element name="legMovement" type="legMovementType" minOccurs="1" maxOccurs="1"> <xs:annotation> <xs:documentation>The style of the leg movements.</xs:documentation> </xs:annotation>

```
</xs:element>
```

Element kickStyle / standardKick

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	standardStrokeType
Properties	content: simple minOccurs: 1 maxOccurs: 1
Facets	enumeration butterfly enumeration backstroke enumeration breaststroke enumeration freestyle enumeration individualMedley enumeration reverseIndividualMedley enumeration individualMedleyOverlap enumeration individualMedleyOrder enumeration reverseIndividualMedley-Order enumeration any enumeration nr1 enumeration nr2 enumeration nr3 enumeration nr4 enumeration notButterfly enumeration notBackstroke enumeration notBreaststroke enumeration notFreestyle
Source	`<xs:element name="standardKick" minOccurs="1" maxOccurs="1" type="standardStrokeType"/>`

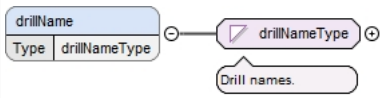
Element strokeType / drill

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	drillType
Properties	content: complex
Model	drillName{0,1} , drillStroke
Children	drillName, drillStroke

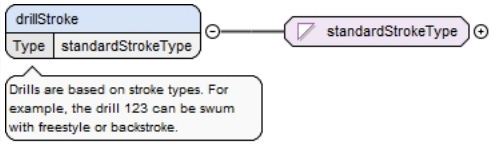
Instance	`<drill xmlns="https://github.com/bartneck/swiML">`
	`<drillName>{0,1}</drillName>`
	`<drillStroke>{1,1}</drillStroke>`

	`</drill>`
Source	`<xs:element name="drill" type="drillType"/>`

Element drillType / drillName

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	drillNameType
Properties	content: simple minOccurs: 0 maxOccurs: 1
Facets	enumeration 6KickDrill enumeration 8KickDrill enumeration 10KickDrill enumeration 12KickDrill enumeration fingerTrails enumeration 123 enumeration bigDog enumeration scull enumeration singleArm enumeration any enumeration technic enumeration dogPaddle enumeration tarzan enumeration fist enumeration 2Kick1Pull enumeration 3Kick1Pull enumeration 2Pull1Kick enumeration 3Pull1Kick enumeration 3Right3Left enumeration 2Right2Left enumeration other
Source	`<xs:element name="drillName" minOccurs="0" maxOccurs="1" type="drillNameType"/>`

Element drillType / drillStroke

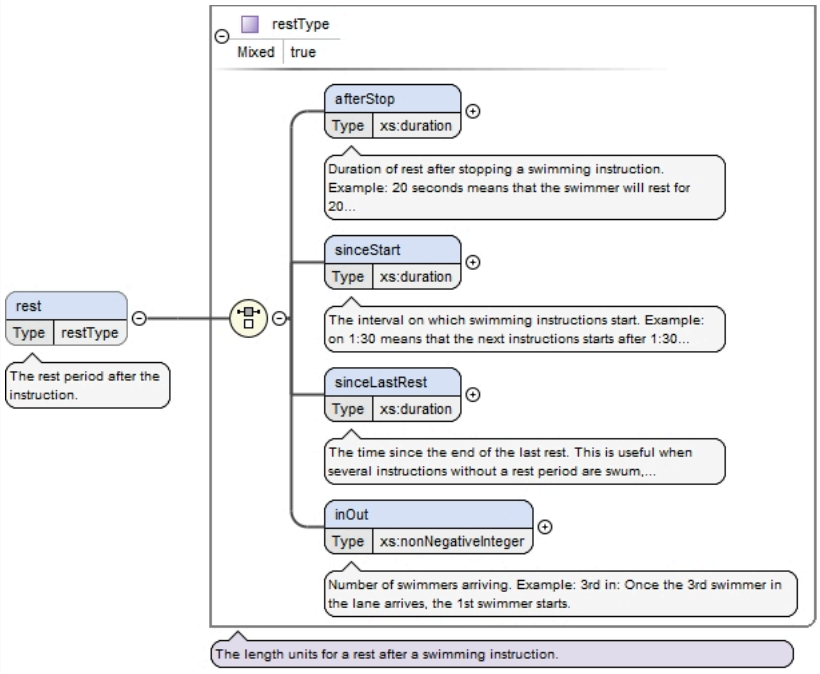
Namespace	https://github.com/bartneck/swiML
Annotations	Drills are based on stroke types. For example, the drill 123 can be swum with freestyle or backstroke.
Diagram	
Type	standardStrokeType
Properties	content: simple minOccurs: 1 maxOccurs: 1
Facets	enumeration butterfly

	enumeration	backstroke
	enumeration	breaststroke
	enumeration	freestyle
	enumeration	individualMedley
	enumeration	reverseIndividualMedley
	enumeration	individualMedleyOverlap
	enumeration	individualMedleyOrder
	enumeration	reverseIndividualMedley-Order
	enumeration	any
	enumeration	nr1
	enumeration	nr2
	enumeration	nr3
	enumeration	nr4
	enumeration	notButterfly
	enumeration	notBackstroke
	enumeration	notBreaststroke
	enumeration	notFreestyle

Source

```
<xs:element name="drillStroke" type="standardStrokeType" maxOccurs="1" minOccurs="1">
  <xs:annotation>
    <xs:documentation>Drills are based on stroke types. For example, the drill 123 can be swum with
 freestyle or backstroke.</xs:documentation>
  </xs:annotation>
</xs:element>
```

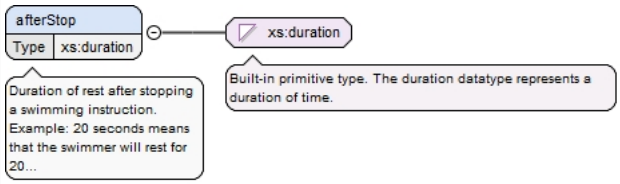
Element `instructionGroup / rest`

Namespace	https://github.com/bartneck/swiML
Annotations	The rest period after the instruction.
Diagram	
Type	restType

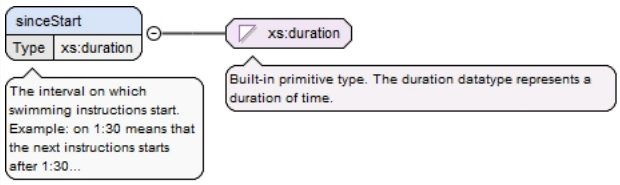
Properties	content:	complex
	minOccurs:	0
	maxOccurs:	1
	mixed:	true

Model	afterStop | sinceStart | sinceLastRest | inOut
Children	afterStop, inOut, sinceLastRest, sinceStart
Instance	<rest xmlns="https://github.com/bartneck/swiML"> <afterStop>{1,1}</afterStop> <sinceStart>{1,1}</sinceStart> <sinceLastRest>{1,1}</sinceLastRest> <inOut>{1,1}</inOut> </rest>
Source	<xs:element name="rest" minOccurs="0" maxOccurs="1" type="restType"> <xs:annotation> <xs:documentation>The rest period after the instruction.</xs:documentation> </xs:annotation> </xs:element>

Element restType / afterStop

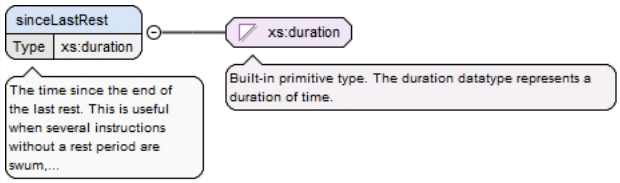
Namespace	https://github.com/bartneck/swiML
Annotations	Duration of rest after stopping a swimming instruction. Example: 20 seconds means that the swimmer will rest for 20 seconds after stopping the current instructions.
Diagram	
Type	xs:duration
Properties	content: simple
Source	<xs:element name="afterStop" type="xs:duration"> <xs:annotation> <xs:documentation>Duration of rest after stopping a swimming instruction. Example: 20 seconds means that the swimmer will rest for 20 seconds after stopping the current instructions.</ xs:documentation> </xs:annotation> </xs:element>

Element restType / sinceStart

Namespace	https://github.com/bartneck/swiML
Annotations	The interval on which swimming instructions start. Example: on 1:30 means that the next instructions starts after 1:30 from starting the current instruction.
Diagram	
Type	xs:duration
Properties	content: simple
Source	<xs:element name="sinceStart" type="xs:duration"> <xs:annotation> <xs:documentation>The interval on which swimming instructions start. Example: on 1:30 means that the next instructions starts after 1:30 from starting the current instruction.</xs:documentation> </xs:annotation> </xs:element>

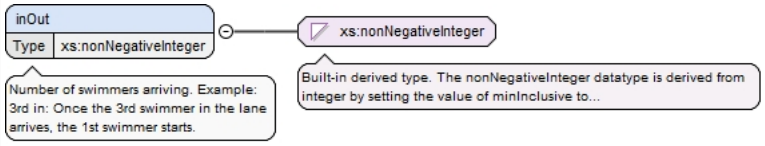
Element restType / sinceLastRest

Namespace	https://github.com/bartneck/swiML
Annotations	The time since the end of the last rest. This is useful when several instructions without a rest period are swum, followed by a since start type rest.

Diagram	
Type	xs:duration
Properties	content: simple

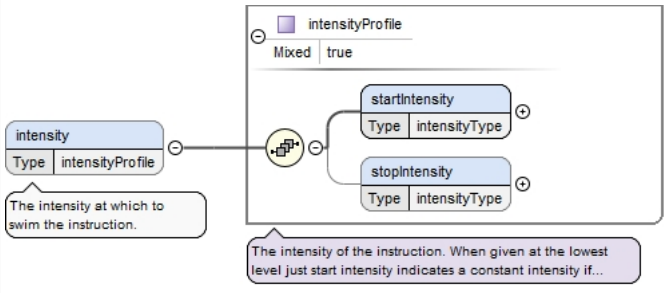
Source	`<xs:element name="sinceLastRest" type="xs:duration">`
	`  <xs:annotation>`
	`  <xs:documentation>The time since the end of the last rest. This is useful when several instructions without a rest period are swum, followed by a since start type rest.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element `restType / inOut`

Namespace	https://github.com/bartneck/swiML
Annotations	Number of swimmers arriving. Example: 3rd in: Once the 3rd swimmer in the lane arrives, the 1st swimmer starts.
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple

Source	`<xs:element name="inOut" type="xs:nonNegativeInteger">`
	`  <xs:annotation>`
	`  <xs:documentation>Number of swimmers arriving. Example: 3rd in: Once the 3rd swimmer in the lane arrives, the 1st swimmer starts.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element `instructionGroup / intensity`

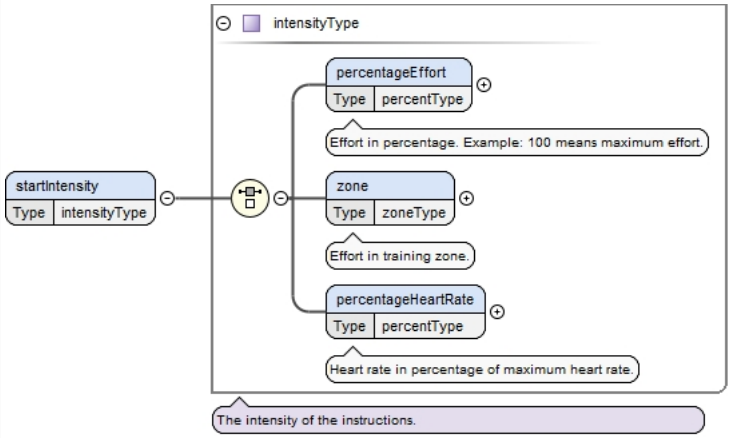
Namespace	https://github.com/bartneck/swiML
Annotations	The intensity at which to swim the instruction.
Diagram	
Type	intensityProfile
Properties	content: complex minOccurs: 0 maxOccurs: 1 mixed: true
Model	startIntensity , stopIntensity{0,1}
Children	startIntensity, stopIntensity

Instance	`<intensity xmlns="https://github.com/bartneck/swiML">`
	`  <startIntensity>{1,1}</startIntensity>`
	`  <stopIntensity>{0,1}</stopIntensity>`

	`</intensity>`

Source	`<xs:element name="intensity" minOccurs="0" maxOccurs="1" type="intensityProfile">`
	`  <xs:annotation>`
	`  <xs:documentation>The intensity at which to swim the instruction.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

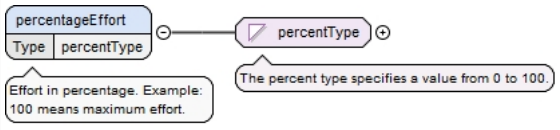
Element `intensityProfile / startIntensity`

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	intensityType
Properties	content: `complex` minOccurs: `1` maxOccurs: `1`
Model	percentageEffort | zone | percentageHeartRate
Children	percentageEffort, percentageHeartRate, zone

Instance	`<startIntensity xmlns="https://github.com/bartneck/swiML">`
	`  <percentageEffort>{1,1}</percentageEffort>`
	`  <zone>{1,1}</zone>`
	`  <percentageHeartRate>{1,1}</percentageHeartRate>`
	`</startIntensity>`

Source	`<xs:element name="startIntensity" minOccurs="1" maxOccurs="1" type="intensityType"/>`

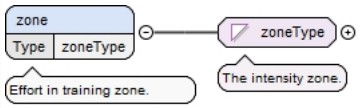
Element `intensityType / percentageEffort`

Namespace	https://github.com/bartneck/swiML
Annotations	`Effort in percentage. Example: 100 means maximum effort.`
Diagram	
Type	percentType
Properties	content: `simple`
Facets	maxInclusive: `100` minInclusive: `0`

Source	`<xs:element name="percentageEffort" type="percentType">`
	`  <xs:annotation>`
	`  <xs:documentation>Effort in percentage. Example: 100 means maximum effort.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

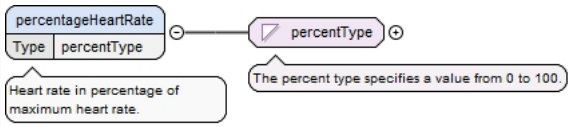
Element `intensityType / zone`

Namespace	https://github.com/bartneck/swiML

Annotations	Effort in training zone.
Diagram	
Type	zoneType
Properties	content: simple
Facets	enumeration: easy enumeration: threshold enumeration: endurance enumeration: racePace enumeration: max

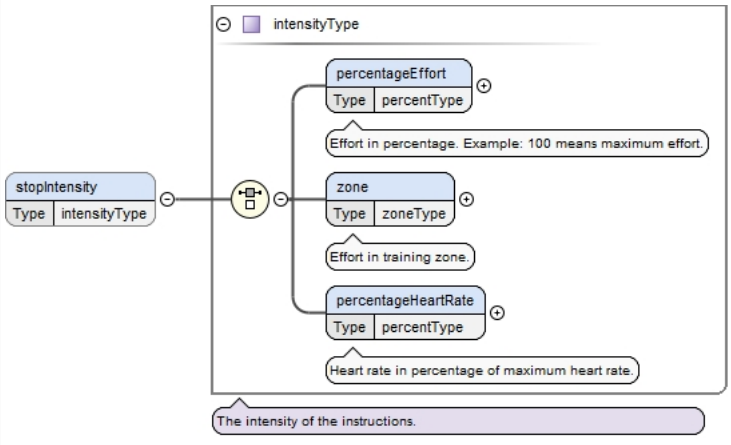
Source	`<xs:element name="zone" type="zoneType">`
	`  <xs:annotation>`
	`  <xs:documentation>Effort in training zone.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element intensityType / percentageHeartRate

Namespace	https://github.com/bartneck/swiML
Annotations	Heart rate in percentage of maximum heart rate.
Diagram	
Type	percentType
Properties	content: simple
Facets	maxInclusive: 100 minInclusive: 0

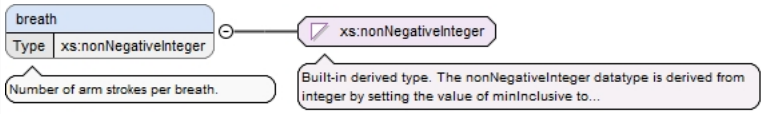
Source	`<xs:element name="percentageHeartRate" type="percentType">`
	`  <xs:annotation>`
	`  <xs:documentation>Heart rate in percentage of maximum heart rate.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element intensityProfile / stopIntensity

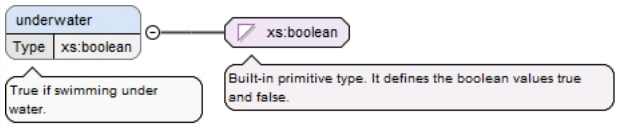
Namespace	https://github.com/bartneck/swiML
Diagram	
Type	intensityType
Properties	content: complex

	minOccurs: 0 maxOccurs: 1
Model	percentageEffort | zone | percentageHeartRate
Children	percentageEffort, percentageHeartRate, zone
Instance	<stopIntensity xmlns="https://github.com/bartneck/swiML"> <percentageEffort>{1,1}</percentageEffort> <zone>{1,1}</zone> <percentageHeartRate>{1,1}</percentageHeartRate> </stopIntensity>
Source	<xs:element name="stopIntensity" minOccurs="0" maxOccurs="1" type="intensityType"/>

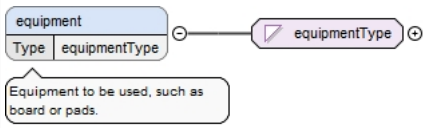
Element instructionGroup / breath

Namespace	https://github.com/bartneck/swiML
Annotations	Number of arm strokes per breath.
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple minOccurs: 0 maxOccurs: 1
Source	<xs:element name="breath" minOccurs="0" maxOccurs="1" type="xs:nonNegativeInteger"> <xs:annotation> <xs:documentation>Number of arm strokes per breath.</xs:documentation> </xs:annotation> </xs:element>

Element instructionGroup / underwater

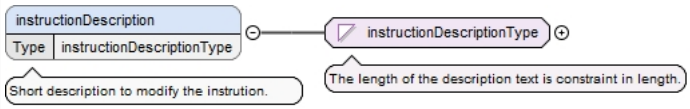
Namespace	https://github.com/bartneck/swiML
Annotations	True if swimming under water.
Diagram	
Type	xs:boolean
Properties	content: simple minOccurs: 0 maxOccurs: 1
Source	<xs:element name="underwater" minOccurs="0" maxOccurs="1" type="xs:boolean"> <xs:annotation> <xs:documentation>True if swimming under water.</xs:documentation> </xs:annotation> </xs:element>

Element instructionGroup / equipment

Namespace	https://github.com/bartneck/swiML
Annotations	Equipment to be used, such as board or pads.
Diagram	
Type	equipmentType

Properties	content:	simple
	minOccurs:	0
	maxOccurs:	unbounded
Facets	enumeration	board
	enumeration	pads
	enumeration	pullBuoy
	enumeration	fins
	enumeration	snorkle
	enumeration	chute
	enumeration	stretchCord
Source	<xs:element name="equipment" minOccurs="0" maxOccurs="unbounded" type="equipmentType"> <xs:annotation> <xs:documentation>Equipment to be used, such as board or pads.</xs:documentation> </xs:annotation> </xs:element>	

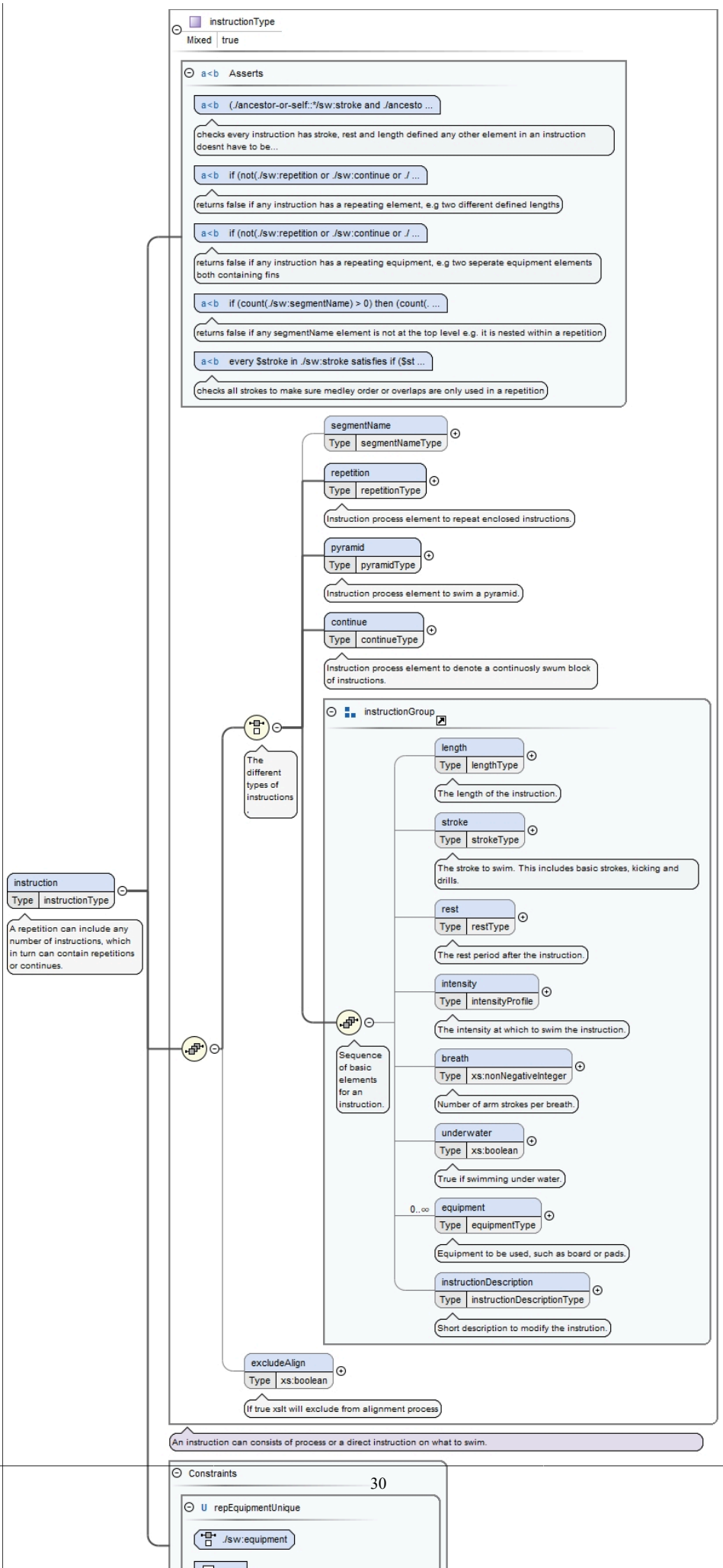
Element instructionGroup / instructionDescription

Namespace	https://github.com/bartneck/swiML
Annotations	Short description to modify the instrution.
Diagram	
Type	instructionDescriptionType
Properties	content: simple minOccurs: 0 maxOccurs: 1
Facets	maxLength 100
Source	<xs:element name="instructionDescription" type="instructionDescriptionType" minOccurs="0" maxOccurs="1"> <xs:annotation> <xs:documentation>Short description to modify the instrution.</xs:documentation> </xs:annotation> </xs:element>

Element repetitionType / instruction

Namespace	https://github.com/bartneck/swiML
Annotations	A repetition can include any number of instructions, which in turn can contain repetitions or continues.

Diagram


instructionType
Mixed true
a<b Asserts
a<b (./ancestor-or-self::*/sw:stroke and ./ancesto ...
checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be...
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating element, e.g two different defined lengths
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins
a<b if (count(./sw:segmentName) > 0) then (count(. ...
returns false if any segmentName element is not at the top level e.g. it is nested within a repetition
a<b every $stroke in ./sw:stroke satisfies if ($st ...
checks all strokes to make sure medley order or overlaps are only used in a repetition
segmentName
Type segmentNameType
repetition
Type repetitionType
Instruction process element to repeat enclosed instructions.
pyramid
Type pyramidType
Instruction process element to swim a pyramid.
continue
Type continueType
Instruction process element to denote a continuosly swum block of instructions.
instructionGroup
The different types of instructions.
length
Type lengthType
The length of the instruction.
stroke
Type strokeType
The stroke to swim. This includes basic strokes, kicking and drills.
rest
Type restType
The rest period after the instruction.
intensity
Type intensityProfile
The intensity at which to swim the instruction.
Sequence of basic elements for an instruction.
breath
Type xs:nonNegativeInteger
Number of arm strokes per breath.
underwater
Type xs:boolean
True if swimming under water.
0..∞
equipment
Type equipmentType
Equipment to be used, such as board or pads.
instructionDescription
Type instructionDescriptionType
Short description to modify the instrution.
instruction
Type instructionType
A repetition can include any number of instructions, which in turn can contain repetitions or continues.
excludeAlign
Type xs:boolean
If true xslt will exclude from alignment process
An instruction can consists of process or a direct instruction on what to swim.
Constraints
U repEquipmentUnique
./sw:equipment

Type	instructionType
Properties	content: complex minOccurs: 1 maxOccurs: unbounded mixed: true
Model	(segmentName{0,1} | repetition | pyramid | continue | (length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1})) , excludeAlign{0,1}
Children	breath, continue, equipment, excludeAlign, instructionDescription, intensity, length, pyramid, repetition, rest, segmentName, stroke, underwater

Instance

```
<instruction xmlns="https://github.com/bartneck/swiML">
  <segmentName>{0,1}</segmentName>
  <repetition>{1,1}</repetition>
  <pyramid>{1,1}</pyramid>
  <continue>{1,1}</continue>
  <length>{0,1}</length>
  <stroke>{0,1}</stroke>
  <rest>{0,1}</rest>
  <intensity>{0,1}</intensity>
  <breath>{0,1}</breath>
  <underwater>{0,1}</underwater>
  <equipment>{0,unbounded}</equipment>
  <instructionDescription>{0,1}</instructionDescription>
  <excludeAlign>{0,1}</excludeAlign>
</instruction>
```

Asserts

Test	XPath default namespace
(./ancestor-or-self::*/sw:stroke and ./ancestor-or-self::*/sw:length) or ./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName	
`checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be defined`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./* satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'length' or name() = 'stroke' or name() = 'rest' or name() = 'intensity' or name() = 'breath' or name() = 'underwater'] satisfies not(name($element) = name($match)))) else (true())	
`returns false if any instruction has a repeating element, e.g two different defined lengths`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./*[name() = 'equipment'] satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'equipment'] satisfies not($element/text() = $match/text()))) else (true())	
`returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins`	
if (count(./sw:segmentName) > 0) then (count(./sw:segmentName/../../ancestor::*) = 0) else (true())	
`returns false if any segmentName element is not at the top level e.g. it is nested within a repetition`	
every $stroke in ./sw:stroke satisfies if ($stroke/sw:standardStroke = 'individualMedleyOverlap' or $stroke/sw:standardStroke = 'individualMedleyOrder' or $stroke/sw:standardStroke = 'reverseIndividualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOverlap' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'reverseIndividualMedleyOrder' or$stroke/sw:drill/sw:drillStroke = 'individualMedleyOverlap' or $stroke/sw:drill/sw:drillStroke = 'individualMedleyOrder' or $stroke/sw:drill/sw:drillStroke = 'reverseIndividualMedleyOrder') then ($stroke/ancestor::*/sw:repetition) or ($stroke/ancestor::*/sw:continue/sw:continueLength) else ($stroke/parent::*)	

	Test	**XPath default namespace**
	checks all strokes to make sure medley order or overlaps are only used in a repetition	

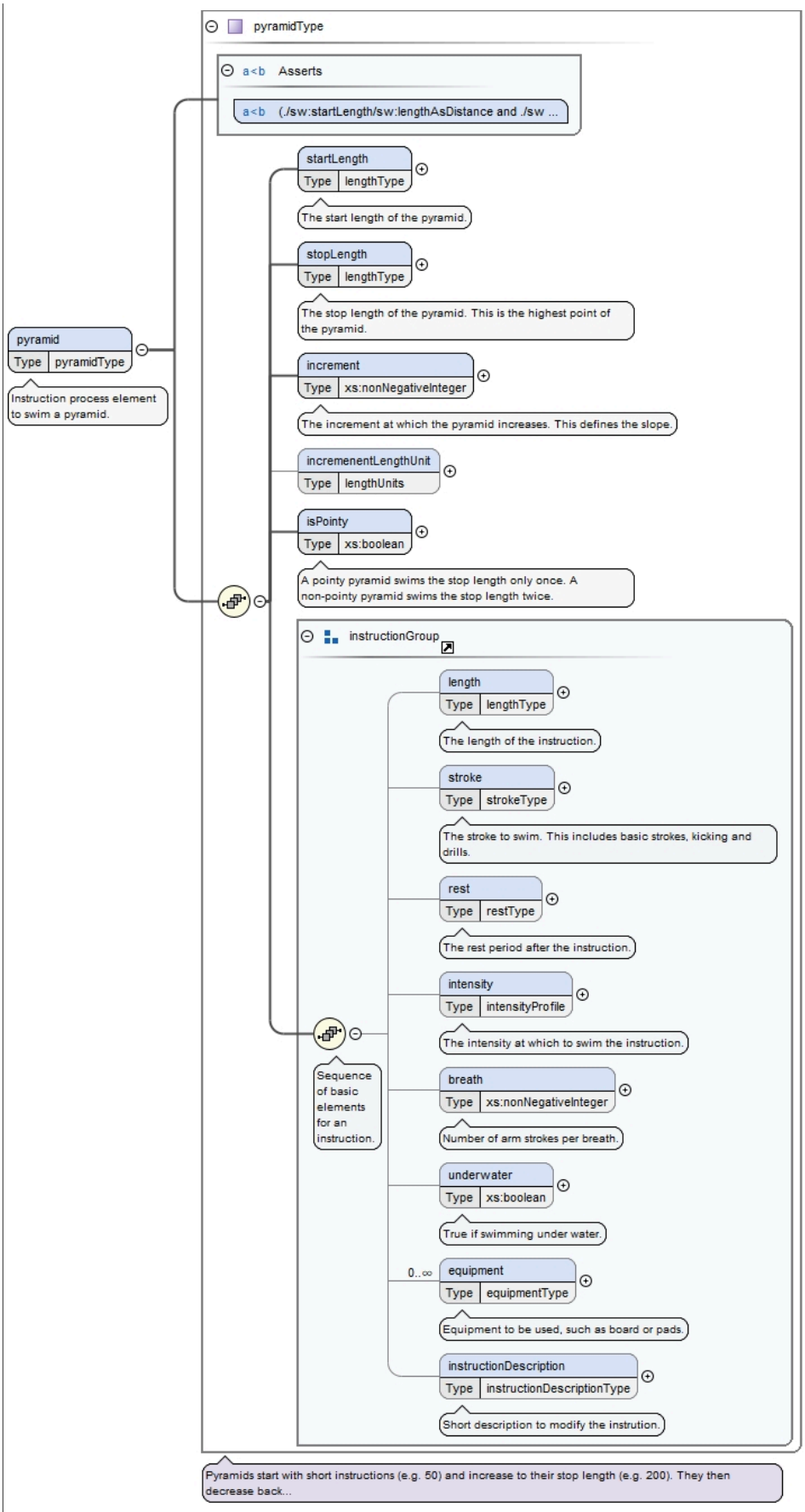
Identity constraints	**QName**	**Type**	**Refer**	**Selector**	**Field(s)**
	repEquipmentUnique	unique		./sw:equipment	.

Source

```
<xs:element name="instruction" minOccurs="1" maxOccurs="unbounded" type="instructionType">
  <xs:annotation>
    <xs:documentation>A repetition can include any number of instructions, which in turn can contain
 repetitions or continues.</xs:documentation>
  </xs:annotation>
  <xs:unique name="repEquipmentUnique">
    <xs:annotation>
      <xs:documentation>Ensures all equipment values in an instruction are unique</xs:documentation>
    </xs:annotation>
    <xs:selector xpath="./sw:equipment"/>
    <xs:field xpath="."/>
  </xs:unique>
</xs:element>
```

Element `instructionType / pyramid`

Namespace	https://github.com/bartneck/swiML
Annotations	Instruction process element to swim a pyramid.

Diagram	
Type	pyramidType

Properties	content: complex
Model	startLength , stopLength , increment , incremenentLengthUnit{0,1} , isPointy , length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1}
Children	breath, equipment, incremenentLengthUnit, increment, instructionDescription, intensity, isPointy, length, rest, startLength, stopLength, stroke, underwater

Instance

```
<pyramid xmlns="https://github.com/bartneck/swiML">
  <startLength>{1,1}</startLength>
  <stopLength>{1,1}</stopLength>
  <increment>{1,1}</increment>
  <incremenentLengthUnit>{0,1}</incremenentLengthUnit>
  <isPointy>{1,1}</isPointy>
  <length>{0,1}</length>
  <stroke>{0,1}</stroke>
  <rest>{0,1}</rest>
  <intensity>{0,1}</intensity>
  <breath>{0,1}</breath>
  <underwater>{0,1}</underwater>
  <equipment>{0,unbounded}</equipment>
  <instructionDescription>{0,1}</instructionDescription>
</pyramid>
```

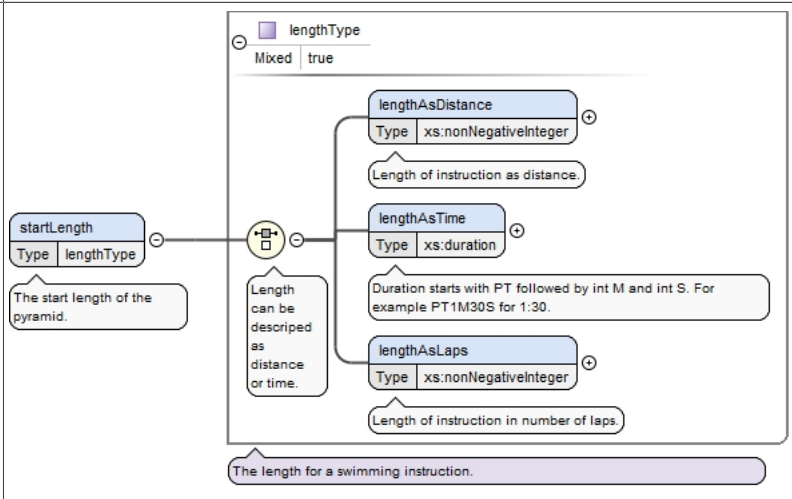
Asserts

Test	XPath default namespace
(./sw:startLength/sw:lengthAsDistance and ./sw:stopLength/sw:lengthAsDistance) or (./sw:startLength/sw:lengthAsLaps and ./sw:stopLength/sw:lengthAsLaps) or (./sw:startLength/sw:lengthAsTime and ./sw:stopLength/sw:lengthAsTime)	

Source

```
<xs:element name="pyramid" type="pyramidType">
  <xs:annotation>
    <xs:documentation>Instruction process element to swim a pyramid.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Element pyramidType / startLength

Namespace	https://github.com/bartneck/swiML
Annotations	The start length of the pyramid.
Diagram	

Type	lengthType
Properties	content: complex minOccurs: 1 maxOccurs: 1 mixed: true
Model	lengthAsDistance | lengthAsTime | lengthAsLaps
Children	lengthAsDistance, lengthAsLaps, lengthAsTime

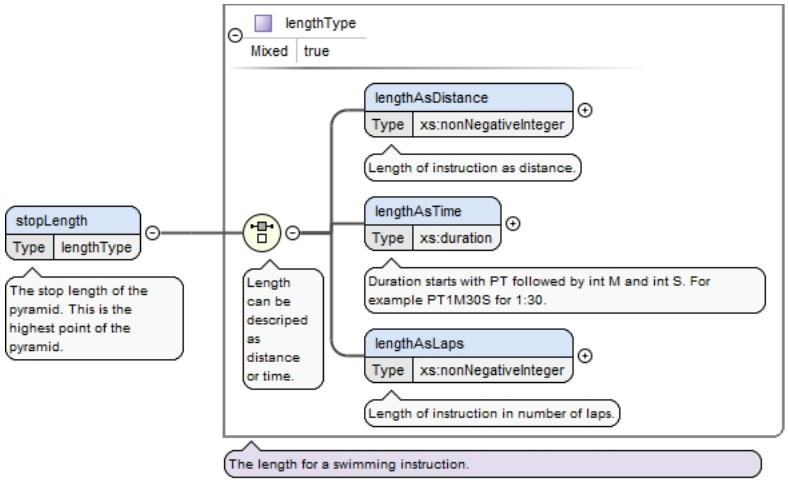
Instance

```
<startLength xmlns="https://github.com/bartneck/swiML">
  <lengthAsDistance>{1,1}</lengthAsDistance>
  <lengthAsTime>{1,1}</lengthAsTime>
  <lengthAsLaps>{1,1}</lengthAsLaps>
```

	`</startLength>`

Source	`<xs:element name="startLength" minOccurs="1" maxOccurs="1" type="lengthType">`
	`  <xs:annotation>`
	`  <xs:documentation>The start length of the pyramid.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

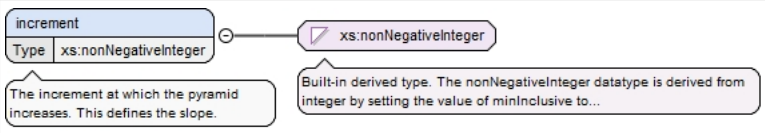
Element pyramidType / stopLength

Namespace	https://github.com/bartneck/swiML
Annotations	The stop length of the pyramid. This is the highest point of the pyramid.
Diagram	
Type	lengthType
Properties	content: complex minOccurs: 1 maxOccurs: 1 mixed: true
Model	lengthAsDistance | lengthAsTime | lengthAsLaps
Children	lengthAsDistance, lengthAsLaps, lengthAsTime

Instance	`<stopLength xmlns="https://github.com/bartneck/swiML">`
	`  <lengthAsDistance>{1,1}</lengthAsDistance>`
	`  <lengthAsTime>{1,1}</lengthAsTime>`
	`  <lengthAsLaps>{1,1}</lengthAsLaps>`
	`</stopLength>`

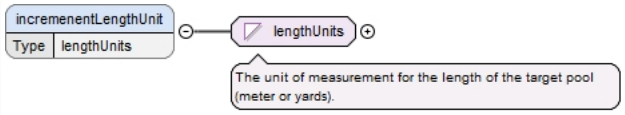
Source	`<xs:element name="stopLength" minOccurs="1" maxOccurs="1" type="lengthType">`
	`  <xs:annotation>`
	`  <xs:documentation>The stop length of the pyramid. This is the highest point of the pyramid.</xs:documentation>`
	`  </xs:annotation>`
	`</xs:element>`

Element pyramidType / increment

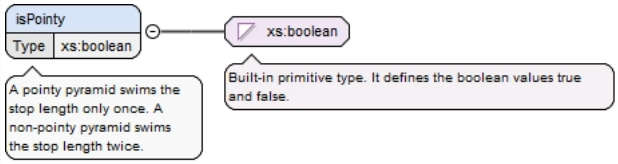
Namespace	https://github.com/bartneck/swiML
Annotations	The increment at which the pyramid increases. This defines the slope.
Diagram	
Type	xs:nonNegativeInteger
Properties	content: simple minOccurs: 1 maxOccurs: 1

Source	<xs:element name="increment" minOccurs="1" maxOccurs="1" type="xs:nonNegativeInteger"> <xs:annotation> <xs:documentation>The increment at which the pyramid increases. This defines the slope.</ xs:documentation> </xs:annotation> </xs:element>

Element pyramidType / incremenentLengthUnit

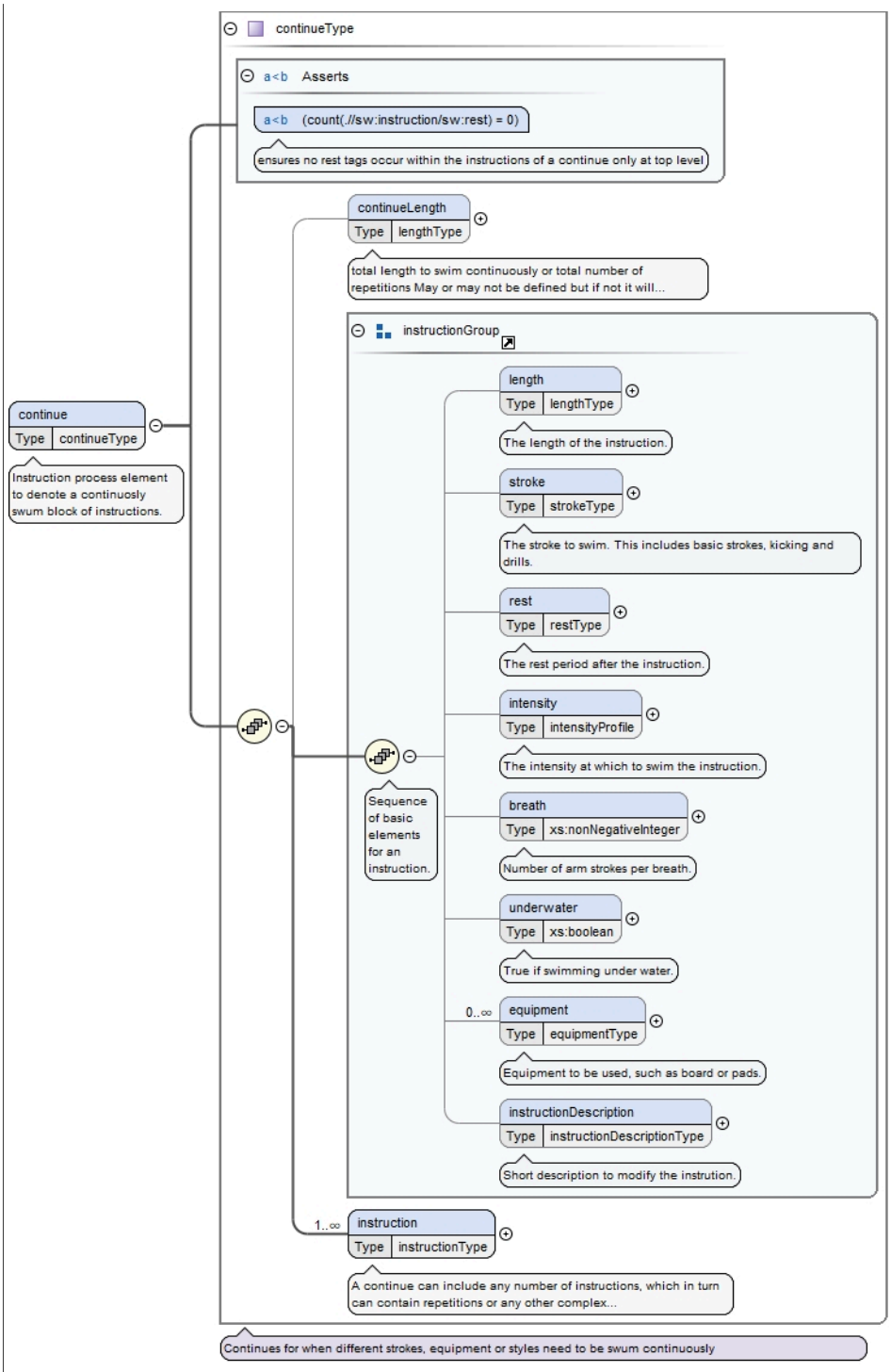
Namespace	https://github.com/bartneck/swiML
Diagram	
Type	lengthUnits
Properties	content: simple minOccurs: 0 maxOccurs: 1
Facets	enumeration meters enumeration kilometers enumeration miles enumeration yards
Source	<xs:element name="incremenentLengthUnit" type="lengthUnits" minOccurs="0" maxOccurs="1"/>

Element pyramidType / isPointy

Namespace	https://github.com/bartneck/swiML
Annotations	A pointy pyramid swims the stop length only once. A non-pointy pyramid swims the stop length twice.
Diagram	
Type	xs:boolean
Properties	content: simple minOccurs: 1 maxOccurs: 1
Source	<xs:element name="isPointy" minOccurs="1" maxOccurs="1" type="xs:boolean"> <xs:annotation> <xs:documentation>A pointy pyramid swims the stop length only once. A non-pointy pyramid swims the stop length twice.</xs:documentation> </xs:annotation> </xs:element>

Element instructionType / continue

Namespace	https://github.com/bartneck/swiML
Annotations	Instruction process element to denote a continuosly swum block of instructions.

Diagram	
Type	continueType
Properties	content: complex
Model	continueLength{0,1} , length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1} , instruction+
Children	breath, continueLength, equipment, instruction, instructionDescription, intensity, length, rest, stroke, underwater
Instance	<continue xmlns="https://github.com/bartneck/swiML"> <continueLength>{0,1}</continueLength> <length>{0,1}</length> <stroke>{0,1}</stroke> <rest>{0,1}</rest> <intensity>{0,1}</intensity> <breath>{0,1}</breath> <underwater>{0,1}</underwater> <equipment>{0,unbounded}</equipment>

```
  <instructionDescription>{0,1}</instructionDescription>
  <instruction>{1,unbounded}</instruction>
</continue>
```

Asserts

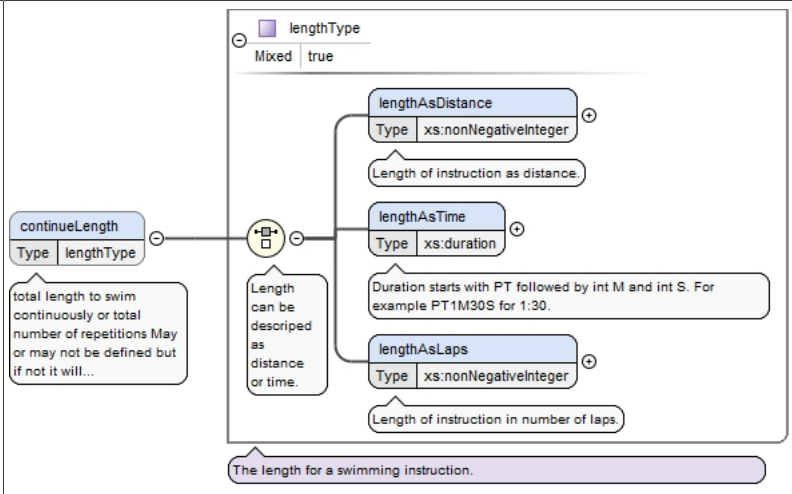
Test	XPath default namespace
(count(.//sw:instruction/sw:rest) = 0)	

```
ensures no rest tags occur within the instructions of a continue
only at top level
```

Source

```
<xs:element name="continue" type="continueType">
  <xs:annotation>
    <xs:documentation>Instruction process element to denote a continuosly swum block of
 instructions.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Element `continueType / continueLength`

Namespace	https://github.com/bartneck/swiML
Annotations	`total length to swim continuously or total number of repetitions May or may not be defined but if not it will automatically calculated from given instructions`
Diagram	
Type	lengthType

Properties

content:	`complex`
minOccurs:	`0`
maxOccurs:	`1`
mixed:	`true`

Model	lengthAsDistance | lengthAsTime | lengthAsLaps
Children	lengthAsDistance, lengthAsLaps, lengthAsTime

Instance

```
<continueLength xmlns="https://github.com/bartneck/swiML">
  <lengthAsDistance>{1,1}</lengthAsDistance>
  <lengthAsTime>{1,1}</lengthAsTime>
  <lengthAsLaps>{1,1}</lengthAsLaps>
</continueLength>
```

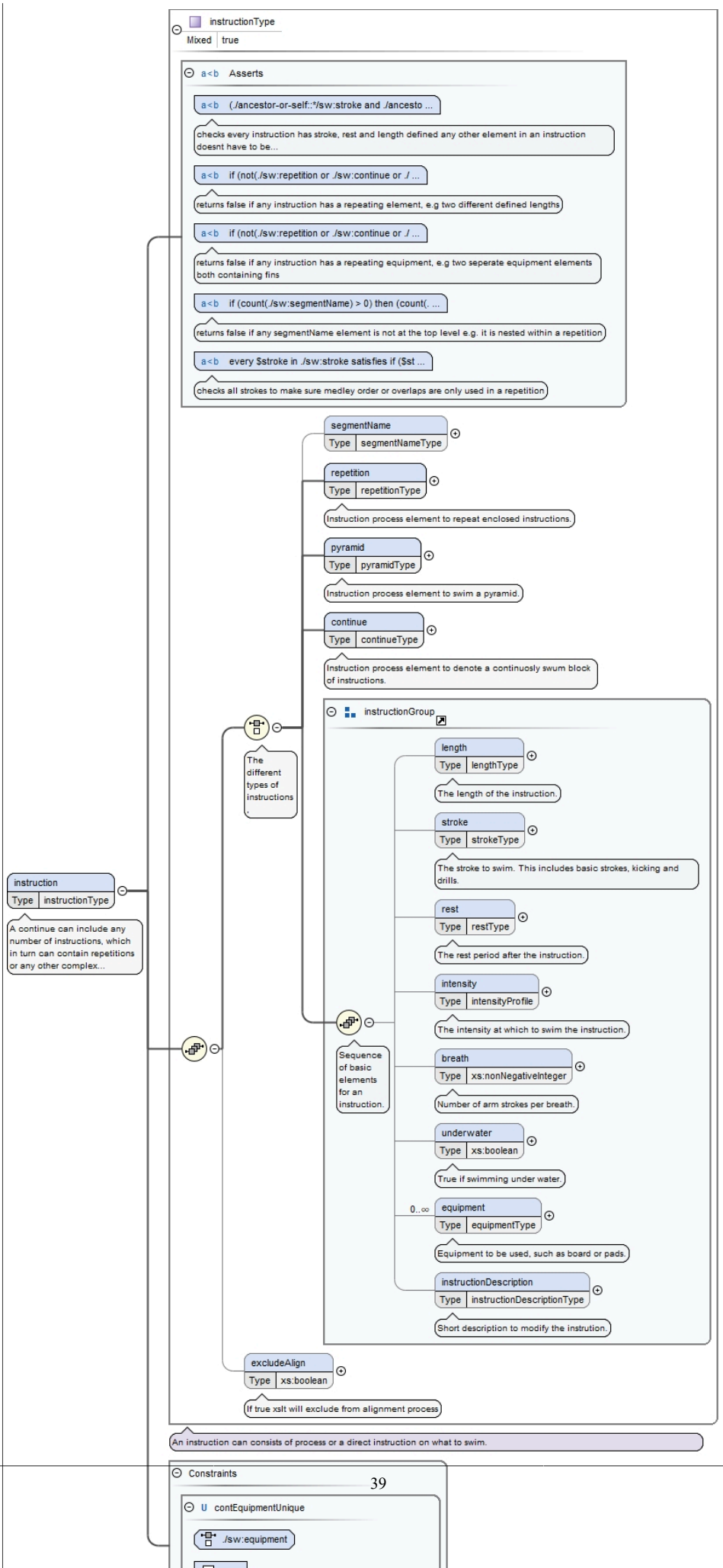
Source

```
<xs:element name="continueLength" minOccurs="0" maxOccurs="1" type="lengthType">
  <xs:annotation>
    <xs:documentation>total length to swim continuously or total number of repetitions May
 or may not be defined but if not it will automatically calculated from given instructions</
xs:documentation>
  </xs:annotation>
</xs:element>
```

Element `continueType / instruction`

Namespace	https://github.com/bartneck/swiML
Annotations	`A continue can include any number of instructions, which in turn can contain repetitions or any other complex instruction type.`

Diagram


instructionType
Mixed true
a<b Asserts
a<b (./ancestor-or-self::*/sw:stroke and ./ancesto ...
checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be...
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating element, e.g two different defined lengths
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins
a<b if (count(./sw:segmentName) > 0) then (count(. ...
returns false if any segmentName element is not at the top level e.g. it is nested within a repetition
a<b every $stroke in ./sw:stroke satisfies if ($st ...
checks all strokes to make sure medley order or overlaps are only used in a repetition
segmentName
Type segmentNameType
repetition
Type repetitionType
Instruction process element to repeat enclosed instructions.
pyramid
Type pyramidType
Instruction process element to swim a pyramid.
continue
Type continueType
Instruction process element to denote a continuosly swum block of instructions.
instructionGroup
The different types of instructions.
length
Type lengthType
The length of the instruction.
stroke
Type strokeType
The stroke to swim. This includes basic strokes, kicking and drills.
rest
Type restType
The rest period after the instruction.
intensity
Type intensityProfile
The intensity at which to swim the instruction.
Sequence of basic elements for an instruction.
breath
Type xs:nonNegativeInteger
Number of arm strokes per breath.
underwater
Type xs:boolean
True if swimming under water.
0..∞
equipment
Type equipmentType
Equipment to be used, such as board or pads.
instructionDescription
Type instructionDescriptionType
Short description to modify the instrution.
instruction
Type instructionType
A continue can include any number of instructions, which in turn can contain repetitions or any other complex...
excludeAlign
Type xs:boolean
If true xslt will exclude from alignment process
An instruction can consists of process or a direct instruction on what to swim.
Constraints
U contEquipmentUnique
./sw:equipment

Type	instructionType
Properties	content: complex minOccurs: 1 maxOccurs: unbounded mixed: true
Model	(segmentName{0,1} | repetition | pyramid | continue | (length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1})) , excludeAlign{0,1}
Children	breath, continue, equipment, excludeAlign, instructionDescription, intensity, length, pyramid, repetition, rest, segmentName, stroke, underwater

Instance

```
<instruction xmlns="https://github.com/bartneck/swiML">
  <segmentName>{0,1}</segmentName>
  <repetition>{1,1}</repetition>
  <pyramid>{1,1}</pyramid>
  <continue>{1,1}</continue>
  <length>{0,1}</length>
  <stroke>{0,1}</stroke>
  <rest>{0,1}</rest>
  <intensity>{0,1}</intensity>
  <breath>{0,1}</breath>
  <underwater>{0,1}</underwater>
  <equipment>{0,unbounded}</equipment>
  <instructionDescription>{0,1}</instructionDescription>
  <excludeAlign>{0,1}</excludeAlign>
</instruction>
```

Asserts

Test	XPath default namespace
(./ancestor-or-self::*/sw:stroke and ./ancestor-or-self::*/sw:length) or ./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName	
`checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be defined`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./* satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'length' or name() = 'stroke' or name() = 'rest' or name() = 'intensity' or name() = 'breath' or name() = 'underwater'] satisfies not(name($element) = name($match)))) else (true())	
`returns false if any instruction has a repeating element, e.g two different defined lengths`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./*[name() = 'equipment'] satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'equipment'] satisfies not($element/text() = $match/text()))) else (true())	
`returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins`	
if (count(./sw:segmentName) > 0) then (count(./sw:segmentName/../../ancestor::*) = 0) else (true())	
`returns false if any segmentName element is not at the top level e.g. it is nested within a repetition`	
every $stroke in ./sw:stroke satisfies if ($stroke/sw:standardStroke = 'individualMedleyOverlap' or $stroke/sw:standardStroke = 'individualMedleyOrder' or $stroke/sw:standardStroke = 'reverseIndividualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOverlap' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'reverseIndividualMedleyOrder' or$stroke/sw:drill/sw:drillStroke = 'individualMedleyOverlap' or $stroke/sw:drill/sw:drillStroke = 'individualMedleyOrder' or $stroke/sw:drill/sw:drillStroke = 'reverseIndividualMedleyOrder') then ($stroke/ancestor::*/sw:repetition) or ($stroke/ancestor::*/sw:continue/sw:continueLength) else ($stroke/parent::*)	

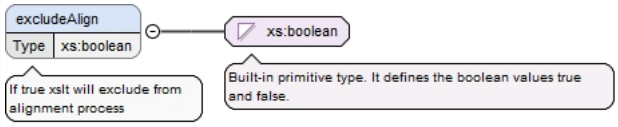
	Test	XPath default namespace
	checks all strokes to make sure medley order or overlaps are only used in a repetition	

Identity constraints	QName	Type	Refer	Selector	Field(s)
	contEquipmentUnique	unique		./sw:equipment	.

Source

```
<xs:element name="instruction" minOccurs="1" maxOccurs="unbounded" type="instructionType">
  <xs:annotation>
    <xs:documentation>A continue can include any number of instructions, which in turn can contain
 repetitions or any other complex instruction type.</xs:documentation>
  </xs:annotation>
  <xs:unique name="contEquipmentUnique">
    <xs:annotation>
      <xs:documentation>Ensures all equipment values in an instruction are unique</xs:documentation>
    </xs:annotation>
    <xs:selector xpath="./sw:equipment"/>
    <xs:field xpath="."/>
  </xs:unique>
</xs:element>
```

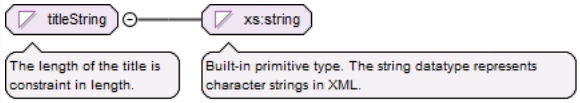
Element `instructionType / excludeAlign`

Namespace	https://github.com/bartneck/swiML
Annotations	If true xslt will exclude from alignment process
Diagram	
Type	xs:boolean
Properties	content: simple minOccurs: 0 maxOccurs: 1
Source	


```
<xs:element name="excludeAlign" type="xs:boolean" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:documentation>If true xslt will exclude from alignment process</xs:documentation>
  </xs:annotation>
</xs:element>
```

Simple Type(s)

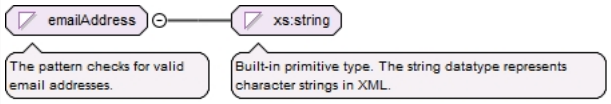
Simple Type `titleString`

Namespace	https://github.com/bartneck/swiML
Annotations	The length of the title is constraint in length.
Diagram	
Type	restriction of xs:string
Facets	maxLength 60
Used by	Element program/title
Source	

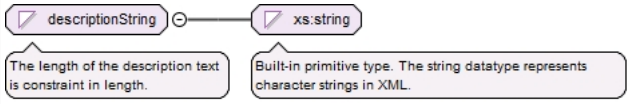

```
<xs:simpleType name="titleString">
  <xs:annotation>
    <xs:documentation>The length of the title is constraint in length.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="60"/>
  </xs:restriction>
</xs:simpleType>
```

Simple Type `emailAddress`

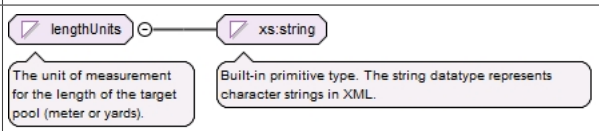
Namespace	https://github.com/bartneck/swiML

Annotations	The pattern checks for valid email addresses.
Diagram	
Type	restriction of xs:string
Facets	pattern [^@]+@[^\.]+\..+
Used by	Element program/author/email
Source	<xs:simpleType name="emailAddress"> <xs:annotation> <xs:documentation>The pattern checks for valid email addresses.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:pattern value="[^@]+@[^\.]+\..+"/> </xs:restriction> </xs:simpleType>

Simple Type descriptionString

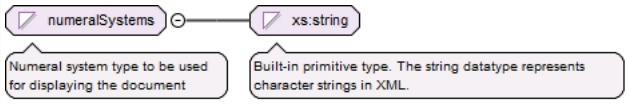
Namespace	https://github.com/bartneck/swiML
Annotations	The length of the description text is constraint in length.
Diagram	
Type	restriction of xs:string
Facets	maxLength 400
Used by	Element program/programDescription
Source	<xs:simpleType name="descriptionString"> <xs:annotation> <xs:documentation>The length of the description text is constraint in length.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:maxLength value="400"/> </xs:restriction> </xs:simpleType>

Simple Type lengthUnits

Namespace	https://github.com/bartneck/swiML
Annotations	The unit of measurement for the length of the target pool (meter or yards).
Diagram	
Type	restriction of xs:string
Facets	enumeration meters enumeration kilometers enumeration miles enumeration yards
Used by	Elements program/lengthUnit, pyramidType/incrementLengthUnit
Source	<xs:simpleType name="lengthUnits"> <xs:annotation> <xs:documentation>The unit of measurement for the length of the target pool (meter or yards).</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="meters"/> <xs:enumeration value="kilometers"/>

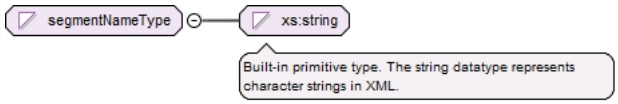
```
    <xs:enumeration value="miles"/>
    <xs:enumeration value="yards"/>
  </xs:restriction>
</xs:simpleType>
```

Simple Type `numeralSystems`

Namespace	https://github.com/bartneck/swiML
Annotations	`Numeral system type to be used for displaying the document`
Diagram	
Type	restriction of xs:string
Facets	enumeration `decimal` enumeration `roman`
Used by	Element program/numeralSystem

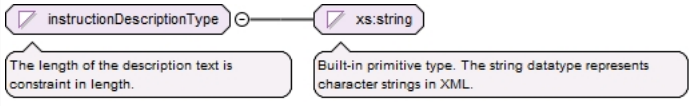
Source	`<xs:simpleType name="numeralSystems">`
	`  <xs:annotation>`
	`  <xs:documentation>Numeral system type to be used for displaying the document</xs:documentation>`
	`  </xs:annotation>`
	`  <xs:restriction base="xs:string">`
	`  <xs:enumeration value="decimal"/>`
	`  <xs:enumeration value="roman"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Simple Type `segmentNameType`

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	restriction of xs:string
Facets	maxLength `50`
Used by	Element instructionType/segmentName

Source	`<xs:simpleType name="segmentNameType">`
	`  <xs:restriction base="xs:string">`
	`  <xs:maxLength value=" 50"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

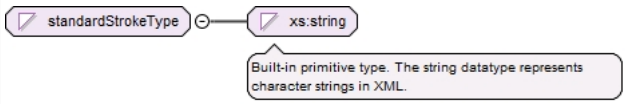
Simple Type `instructionDescriptionType`

Namespace	https://github.com/bartneck/swiML
Annotations	`The length of the description text is constraint in length.`
Diagram	
Type	restriction of xs:string
Facets	maxLength `100`
Used by	Elements instructionGroup/instructionDescription, repetitionType/repetitionDescription

Source	`<xs:simpleType name="instructionDescriptionType">`
	`  <xs:annotation>`
	`  <xs:documentation>The length of the description text is constraint in length.</xs:documentation>`
	`  </xs:annotation>`
	`  <xs:restriction base="xs:string">`
	`  <xs:maxLength value="100"/>`

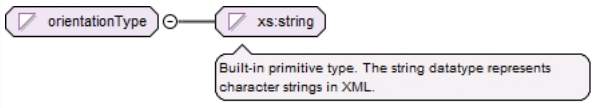
```
  </xs:restriction>
</xs:simpleType>
```

Simple Type `standardStrokeType`

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	restriction of xs:string
Facets	enumeration `butterfly` enumeration `backstroke` enumeration `breaststroke` enumeration `freestyle` enumeration `individualMedley` enumeration `reverseIndividualMedley` enumeration `individualMedleyOverlap` enumeration `individualMedleyOrder` enumeration `reverseIndividualMedley-Order` enumeration `any` enumeration `nr1` enumeration `nr2` enumeration `nr3` enumeration `nr4` enumeration `notButterfly` enumeration `notBackstroke` enumeration `notBreaststroke` enumeration `notFreestyle`
Used by	Elements drillType/drillStroke, kickStyle/standardKick, strokeType/standardStroke

Source	`<xs:simpleType name="standardStrokeType">`
	`  <xs:restriction base="xs:string">`
	`  <xs:enumeration value="butterfly"/>`
	`  <xs:enumeration value="backstroke"/>`
	`  <xs:enumeration value="breaststroke"/>`
	`  <xs:enumeration value="freestyle"/>`
	`  <xs:enumeration value="individualMedley"/>`
	`  <xs:enumeration value="reverseIndividualMedley"/>`
	`  <xs:enumeration value="individualMedleyOverlap"/>`
	`  <xs:enumeration value="individualMedleyOrder"/>`
	`  <xs:enumeration value="reverseIndividualMedleyOrder"/>`
	`  <xs:enumeration value="any"/>`
	`  <xs:enumeration value="nr1"/>`
	`  <xs:enumeration value="nr2"/>`
	`  <xs:enumeration value="nr3"/>`
	`  <xs:enumeration value="nr4"/>`
	`  <xs:enumeration value="notButterfly"/>`
	`  <xs:enumeration value="notBackstroke"/>`
	`  <xs:enumeration value="notBreaststroke"/>`
	`  <xs:enumeration value="notFreestyle"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

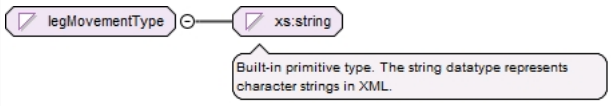
Simple Type `orientationType`

Namespace	https://github.com/bartneck/swiML
Diagram	

Type	restriction of xs:string
Facets	enumeration `front` enumeration `back` enumeration `left` enumeration `right` enumeration `side` enumeration `vertical` enumeration `waka`
Used by	Element kickStyle/orientation

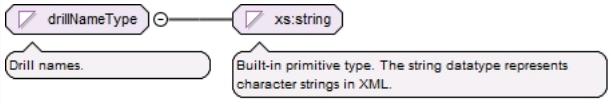
Source	`<xs:simpleType name="orientationType">`
	`  <xs:restriction base="xs:string">`
	`  <xs:enumeration value="front"/>`
	`  <xs:enumeration value="back"/>`
	`  <xs:enumeration value="left"/>`
	`  <xs:enumeration value="right"/>`
	`  <xs:enumeration value="side"/>`
	`  <xs:enumeration value="vertical"/>`
	`  <xs:enumeration value="waka"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Simple Type `legMovementType`

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	restriction of xs:string
Facets	enumeration `flutter` enumeration `dolphin` enumeration `scissor`
Used by	Element kickStyle/legMovement

Source	`<xs:simpleType name="legMovementType">`
	`  <xs:restriction base="xs:string">`
	`  <xs:enumeration value="flutter"/>`
	`  <xs:enumeration value="dolphin"/>`
	`  <xs:enumeration value="scissor"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Simple Type `drillNameType`

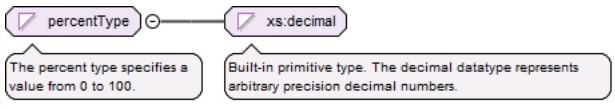
Namespace	https://github.com/bartneck/swiML
Annotations	`Drill names.`
Diagram	
Type	restriction of xs:string
Facets	enumeration `6KickDrill` enumeration `8KickDrill` enumeration `10KickDrill` enumeration `12KickDrill` enumeration `fingerTrails` enumeration `123` enumeration `bigDog` enumeration `scull`

	enumeration	singleArm
	enumeration	any
	enumeration	technic
	enumeration	dogPaddle
	enumeration	tarzan
	enumeration	fist
	enumeration	2Kick1Pull
	enumeration	3Kick1Pull
	enumeration	2Pull1Kick
	enumeration	3Pull1Kick
	enumeration	3Right3Left
	enumeration	2Right2Left
	enumeration	other
Used by	Element	drillType/drillName

Source

```
<xs:simpleType name="drillNameType">
  <xs:annotation>
    <xs:documentation>Drill names.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="6KickDrill"/>
    <xs:enumeration value="8KickDrill"/>
    <xs:enumeration value="10KickDrill"/>
    <xs:enumeration value="12KickDrill"/>
    <xs:enumeration value="fingerTrails"/>
    <xs:enumeration value="123"/>
    <xs:enumeration value="bigDog"/>
    <xs:enumeration value="scull"/>
    <xs:enumeration value="singleArm"/>
    <xs:enumeration value="any"/>
    <xs:enumeration value="technic"/>
    <xs:enumeration value="dogPaddle"/>
    <xs:enumeration value="tarzan"/>
    <xs:enumeration value="fist"/>
    <xs:enumeration value="2Kick1Pull"/>
    <xs:enumeration value="3Kick1Pull"/>
    <xs:enumeration value="2Pull1Kick"/>
    <xs:enumeration value="3Pull1Kick"/>
    <xs:enumeration value="3Right3Left"/>
    <xs:enumeration value="2Right2Left"/>
    <xs:enumeration value="other"/>
  </xs:restriction>
</xs:simpleType>
```

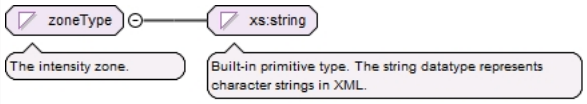
Simple Type `percentType`

Namespace	https://github.com/bartneck/swiML
Annotations	The percent type specifies a value from 0 to 100.
Diagram	
Type	restriction of xs:decimal
Facets	maxInclusive: 100 minInclusive: 0
Used by	Elements: intensityType/percentageEffort, intensityType/percentageHeartRate

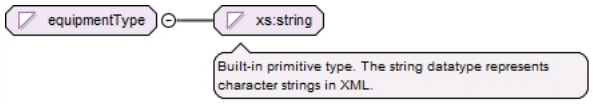
Source

```
<xs:simpleType name="percentType">
  <xs:annotation>
    <xs:documentation>The percent type specifies a value from 0 to 100.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

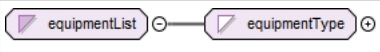
Simple Type zoneType

Namespace	https://github.com/bartneck/swiML
Annotations	The intensity zone.
Diagram	
Type	restriction of xs:string
Facets	enumeration easy enumeration threshold enumeration endurance enumeration racePace enumeration max
Used by	Element intensityType/zone
Source	<xs:simpleType name="zoneType"> <xs:annotation> <xs:documentation>The intensity zone.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="easy"/> <xs:enumeration value="threshold"/> <xs:enumeration value="endurance"/> <xs:enumeration value="racePace"/> <xs:enumeration value="max"/> </xs:restriction> </xs:simpleType>

Simple Type equipmentType

Namespace	https://github.com/bartneck/swiML
Diagram	
Type	restriction of xs:string
Facets	enumeration board enumeration pads enumeration pullBuoy enumeration fins enumeration snorkle enumeration chute enumeration stretchCord
Used by	Element instructionGroup/equipment
Source	<xs:simpleType name="equipmentType"> <xs:restriction base="xs:string"> <xs:enumeration value="board"/> <xs:enumeration value="pads"/> <xs:enumeration value="pullBuoy"/> <xs:enumeration value="fins"/> <xs:enumeration value="snorkle"/> <xs:enumeration value="chute"/> <xs:enumeration value="stretchCord"/> </xs:restriction> </xs:simpleType>

Simple Type equipmentList

Namespace	https://github.com/bartneck/swiML
Diagram	

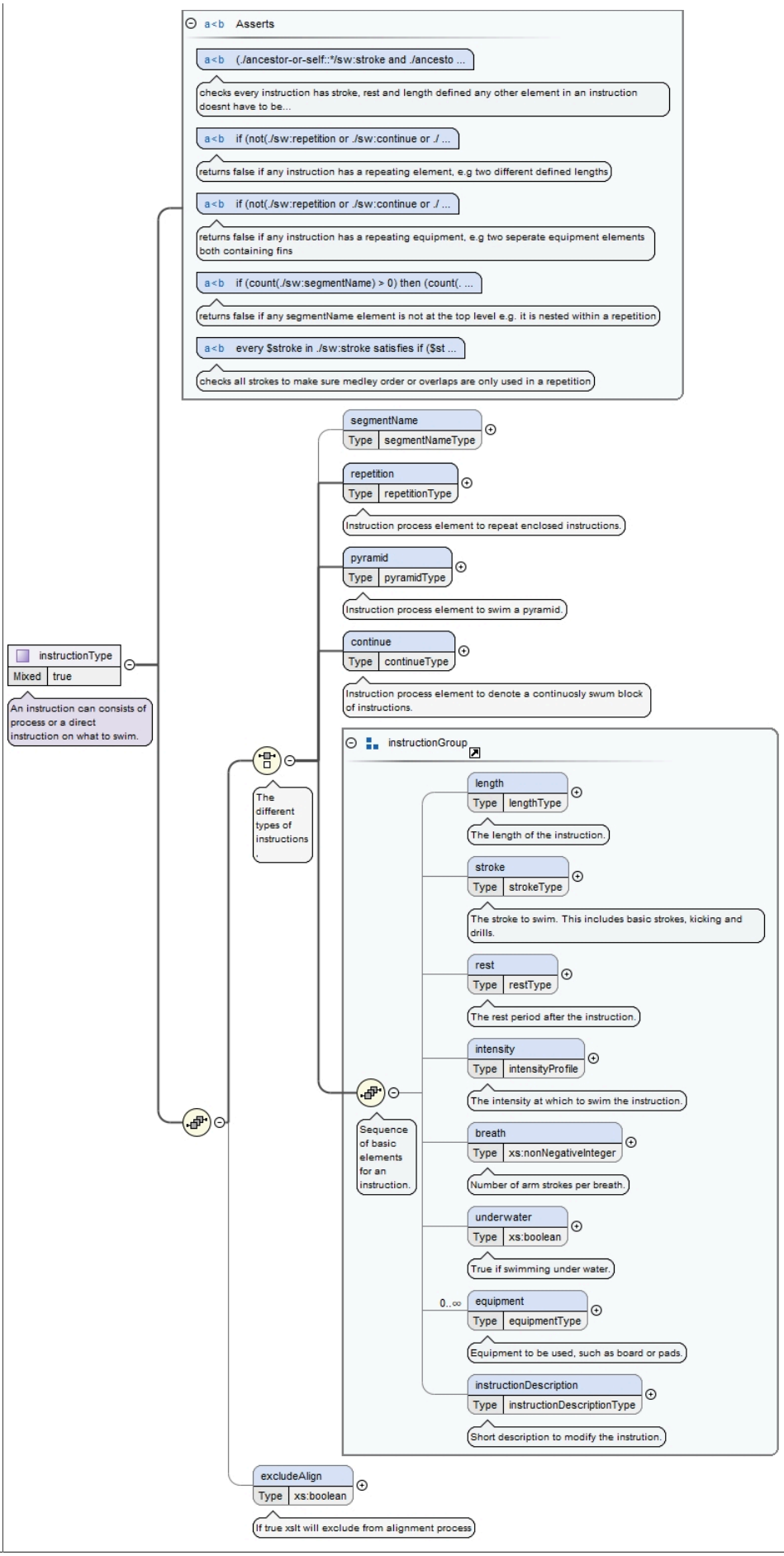
Type	list of equipmentType
Source	<xs:simpleType name="equipmentList"> <xs:list itemType="equipmentType"/> </xs:simpleType>

Complex Type(s)

Complex Type `instructionType`

Namespace	https://github.com/bartneck/swiML
Annotations	An instruction can consists of process or a direct instruction on what to swim.

Diagram


a<b Asserts
a<b (./ancestor-or-self::*/sw:stroke and ./ancesto ...
checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be...
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating element, e.g two different defined lengths
a<b if (not(./sw:repetition or ./sw:continue or ./ ...
returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins
a<b if (count(./sw:segmentName) > 0) then (count(. ...
returns false if any segmentName element is not at the top level e.g. it is nested within a repetition
a<b every $stroke in ./sw:stroke satisfies if ($st ...
checks all strokes to make sure medley order or overlaps are only used in a repetition
instructionType
Mixed true
An instruction can consists of process or a direct instruction on what to swim.
segmentName
Type segmentNameType
repetition
Type repetitionType
Instruction process element to repeat enclosed instructions.
pyramid
Type pyramidType
Instruction process element to swim a pyramid.
continue
Type continueType
Instruction process element to denote a continuosly swum block of instructions.
The different types of instructions.
instructionGroup
length
Type lengthType
The length of the instruction.
stroke
Type strokeType
The stroke to swim. This includes basic strokes, kicking and drills.
rest
Type restType
The rest period after the instruction.
intensity
Type intensityProfile
The intensity at which to swim the instruction.
Sequence of basic elements for an instruction.
breath
Type xs:nonNegativeInteger
Number of arm strokes per breath.
underwater
Type xs:boolean
True if swimming under water.
0..∞
equipment
Type equipmentType
Equipment to be used, such as board or pads.
instructionDescription
Type instructionDescriptionType
Short description to modify the instrution.
excludeAlign
Type xs:boolean
If true xslt will exclude from alignment process

Properties	mixed: `true`
Used by	Elements: continueType/instruction, program/instruction, repetitionType/instruction
Model	(segmentName{0,1} | repetition | pyramid | continue | (length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1})) , excludeAlign{0,1}
Children	breath, continue, equipment, excludeAlign, instructionDescription, intensity, length, pyramid, repetition, rest, segmentName, stroke, underwater

Asserts

Test	XPath default namespace
(./ancestor-or-self::*/sw:stroke and ./ancestor-or-self::*/sw:length) or ./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName	
`checks every instruction has stroke, rest and length defined any other element in an instruction doesnt have to be defined`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./* satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'length' or name() = 'stroke' or name() = 'rest' or name() = 'intensity' or name() = 'breath' or name() = 'underwater'] satisfies not(name($element) = name($match)))) else (true())	
`returns false if any instruction has a repeating element, e.g two different defined lengths`	
if (not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)) then (every $element in ./*[name() = 'equipment'] satisfies (every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'equipment'] satisfies not($element/text() = $match/text()))) else (true())	
`returns false if any instruction has a repeating equipment, e.g two seperate equipment elements both containing fins`	
if (count(./sw:segmentName) > 0) then (count(./sw:segmentName/../../ancestor::*) = 0) else (true())	
`returns false if any segmentName element is not at the top level e.g. it is nested within a repetition`	
every $stroke in ./sw:stroke satisfies if ($stroke/sw:standardStroke = 'individualMedleyOverlap' or $stroke/sw:standardStroke = 'individualMedleyOrder' or $stroke/sw:standardStroke = 'reverseIndividualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOverlap' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOrder' or $stroke/sw:kicking/sw:standardKick = 'reverseIndividualMedleyOrder' or$stroke/sw:drill/sw:drillStroke = 'individualMedleyOverlap' or $stroke/sw:drill/sw:drillStroke = 'individualMedleyOrder' or $stroke/sw:drill/sw:drillStroke = 'reverseIndividualMedleyOrder') then ($stroke/ancestor::*/sw:repetition) or ($stroke/ancestor::*/sw:continue/sw:continueLength) else ($stroke/parent::*)	
`checks all strokes to make sure medley order or overlaps are only used in a repetition`	

Source

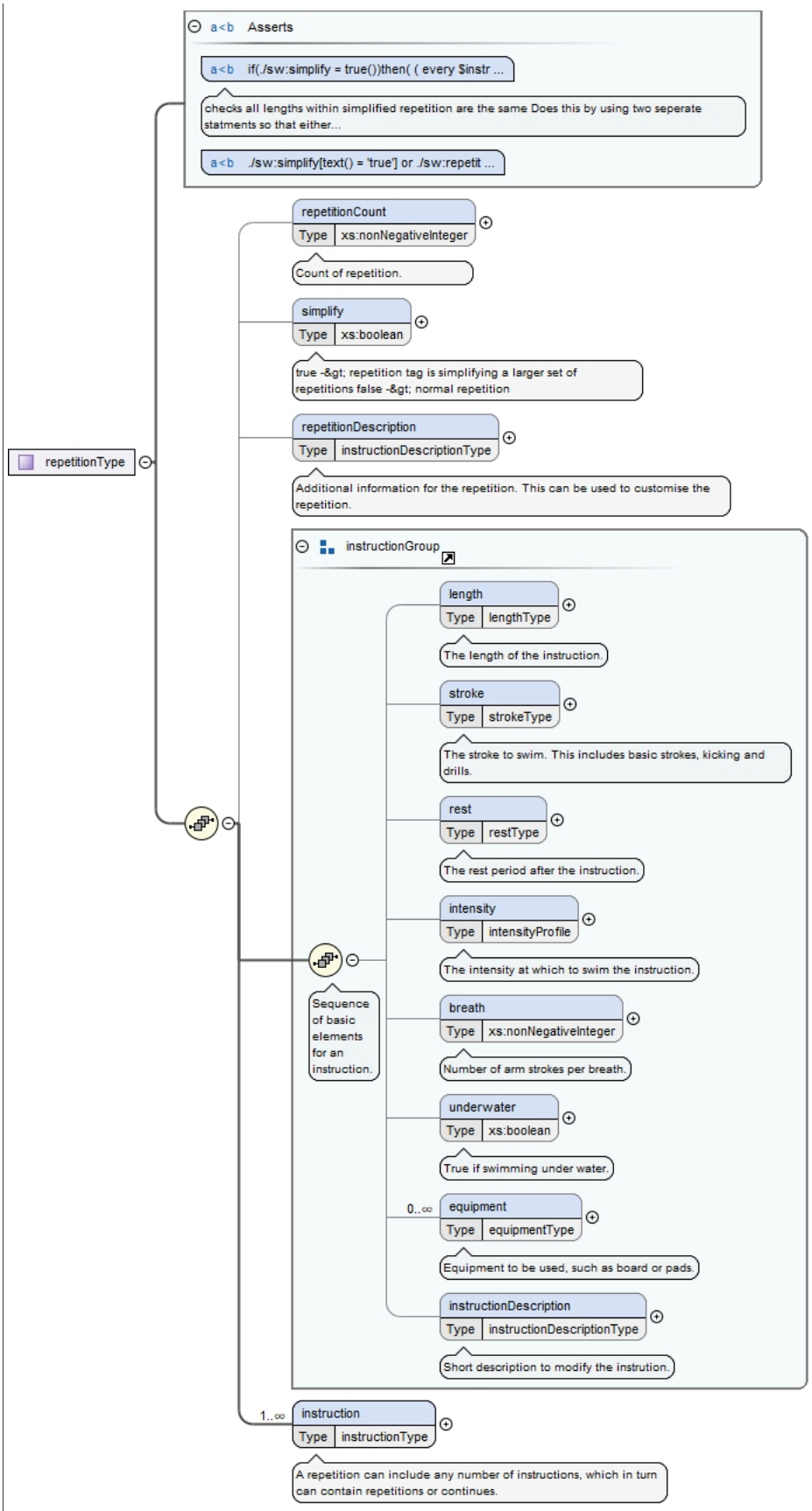
```
<xs:complexType name="instructionType" mixed="true">
  <xs:annotation>
    <xs:documentation>An instruction can consists of process or a direct instruction on what to
 swim.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:choice>
      <xs:annotation>
        <xs:documentation>The different types of instructions,</xs:documentation>
      </xs:annotation>
      <!-- ======================================================= -->
      <!-- Process based elements for instructions -->
      <xs:element name="segmentName" minOccurs="0" maxOccurs="1" type="segmentNameType"/>
      <xs:element name="repetition" type="repetitionType">
        <xs:annotation>
          <xs:documentation>Instruction process element to repeat enclosed instructions.</
xs:documentation>
```

```
      </xs:annotation>
    </xs:element>
    <xs:element name="pyramid" type="pyramidType">
      <xs:annotation>
        <xs:documentation>Instruction process element to swim a pyramid.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="continue" type="continueType">
      <xs:annotation>
        <xs:documentation>Instruction process element to denote a continuosly swum block of
 instructions.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <!-- ================================================== -->
    <!-- Direct instruction on what to swim -->
    <xs:group ref="instructionGroup"/>
  </xs:choice>
  <xs:element name="excludeAlign" type="xs:boolean" minOccurs="0" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>If true xslt will exclude from alignment process</xs:documentation>
    </xs:annotation>
  </xs:element>
 </xs:sequence>
 <!-- checks every instruction has stroke, rest and length defined
      any other element in an instruction doesnt have to be defined-->
 <xs:assert test="                (./ancestor-or-self::*/sw:stroke                and ./ancestor-
or-self::*/sw:length)                or ./sw:repetition                or ./sw:continue
     or ./sw:pyramid                or ./sw:segmentName            ">
   <xs:annotation>
     <xs:documentation>checks every instruction has stroke, rest and length defined any other
 element in an instruction doesnt have to be defined</xs:documentation>
   </xs:annotation>
 </xs:assert>
 <!-- checks every instruction doesnt have repetitions of elements defined and cannot be extended
 -->
 <xs:assert test="                if (not(./sw:repetition                or ./sw:continue
        or ./sw:pyramid                or ./sw:segmentName))                then
     (                    every $element in ./*                satisfies (
          every $match in ./ancestor::*[name() = 'instruction' or name() = 'repetition' or name()
 = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue or ./sw:pyramid or ./
sw:segmentName)]/*[name() = 'length' or name() = 'stroke' or name() = 'rest' or name() = 'intensity'
 or name() = 'breath' or name() = 'underwater']                satisfies
            not(name($element) = name($match))                )                else
               (true())                ">
   <xs:annotation>
     <xs:documentation>returns false if any instruction has a repeating element, e.g two different
 defined lengths</xs:documentation>
   </xs:annotation>
 </xs:assert>
 <xs:assert test="                if (not(./sw:repetition                or ./sw:continue
          or ./sw:pyramid                or ./sw:segmentName))                then
       (                    every $element in ./*[name() = 'equipment']
 satisfies (                    every $match in ./ancestor::*[name() = 'instruction' or name()
 = 'repetition' or name() = 'continue' or name() = 'pyramid'][not(./sw:repetition or ./sw:continue
 or ./sw:pyramid or ./sw:segmentName)]/*[name() = 'equipment']                satisfies
                        not($element/text() = $match/text())                ))
       else                (true())                ">
   <xs:annotation>
     <xs:documentation>returns false if any instruction has a repeating equipment, e.g two seperate
 equipment elements both containing fins</xs:documentation>
   </xs:annotation>
 </xs:assert>
 <xs:assert test="                if (count(./sw:segmentName) > 0)                then
         (count(./sw:segmentName/../../ancestor::*) = 0)                else
 (true())                ">
   <xs:annotation>
     <xs:documentation>returns false if any segmentName element is not at the top level e.g. it is
 nested within a repetition</xs:documentation>
   </xs:annotation>
 </xs:assert>
 <!-- check if this is just for standard stroke or for kicks and drills too -->
 <xs:assert test="                every $stroke in ./sw:stroke
 satisfies                if ($stroke/sw:standardStroke = 'individualMedleyOverlap'
 or $stroke/sw:standardStroke = 'individualMedleyOrder' or $stroke/sw:standardStroke =
 'reverseIndividualMedleyOrder'                or $stroke/sw:kicking/sw:standardKick =
 'individualMedleyOverlap' or $stroke/sw:kicking/sw:standardKick = 'individualMedleyOrder' or
 $stroke/sw:kicking/sw:standardKick = 'reverseIndividualMedleyOrder'                or
$stroke/sw:drill/sw:drillStroke = 'individualMedleyOverlap' or $stroke/sw:drill/sw:drillStroke =
 'individualMedleyOrder' or $stroke/sw:drill/sw:drillStroke = 'reverseIndividualMedleyOrder')
            then                ($stroke/ancestor::*/sw:repetition)
  or                ($stroke/ancestor::*/sw:continue/sw:continueLength)
 else                ($stroke/parent::*)
       ">
```

```
    <xs:annotation>
      <xs:documentation>checks all strokes to make sure medley order or overlaps are only used in a
 repetition</xs:documentation>
    </xs:annotation>
  </xs:assert>
</xs:complexType>
```

Complex Type `repetitionType`

Namespace	https://github.com/bartneck/swiML
Annotations	

Diagram	
Used by	Element instructionType/repetition
Model	repetitionCount{0,1} , simplify{0,1} , repetitionDescription{0,1} , length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1} , instruction+

Children	breath, equipment, instruction, instructionDescription, intensity, length, repetitionCount, repetitionDescription, rest, simplify, stroke, underwater

Asserts

Test	XPath default namespace
if(./sw:simplify = true())then((every $instruction in ./sw:instruction[not(./sw:pyramid or ./sw:segmentName)] satisfies((if($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) then(if(count($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) = 1) then(number(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance)) else(sum(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length \| ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance))) else(0)+(if($instruction/descendant-or-self::sw:continueLength) then(number($instruction/descendant-or-self::sw:continueLength)) else(0))) = number((((./descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance) | (./descendant-or-self::sw:continueLength))[1])))or(every $instruction in ./sw:instruction[not(./sw:pyramid or ./sw:segmentName)] satisfies((if($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) then(if(count($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) = 1) then(number(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps)) else(sum(($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length \| ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps))) else(0)+(if($instruction/descendant-or-self::sw:continueLength) then(number($instruction/descendant-or-self::sw:continueLength)) else(0))) = number((((./descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps) | (./descendant-or-self::sw:continueLength))[1])))else(true())	
checks all lengths within simplified repetition are the same Does this by using two seperate statments so that either all lengthAsDistances are the same or all lengthAsLaps checks that every instruction, repetition and continue has the same base length e.g. 100m , 4x100m and 50,50 swim as 100 ; these are all of length 100 so can be simplified in the same repetition has to choose whether to use continueLength or add up all instructions within a continue	
./sw:simplify[text() = 'true'] or ./sw:repetitionCount and not(./sw:simplify[text() = 'true'] and ./sw:repetitionCount)	

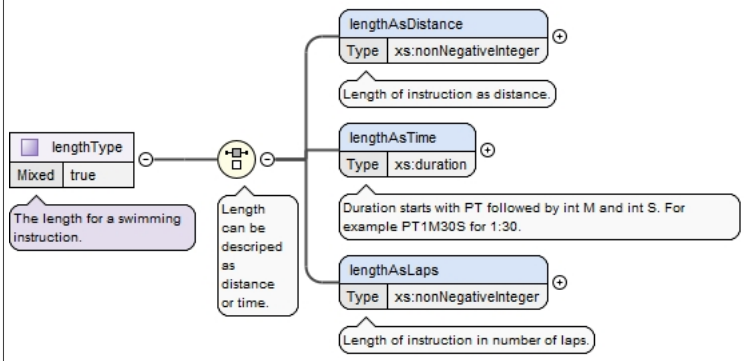
Source

```
<xs:complexType name="repetitionType">
  <xs:annotation>
    <xs:documentation/>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="repetitionCount" type="xs:nonNegativeInteger" minOccurs="0" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>Count of repetition.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="simplify" minOccurs="0" maxOccurs="1" type="xs:boolean">
```

```
      <xs:annotation>
        <xs:documentation>true -> repetition tag is simplifying a larger set of repetitions false ->
 normal repetition</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="repetitionDescription" minOccurs="0" maxOccurs="1"
 type="instructionDescriptionType">
      <xs:annotation>
        <xs:documentation>Additional information for the repetition. This can be used to customise
 the repetition.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <!-- Common elements for instructions -->
    <xs:group ref="instructionGroup"/>
    <!-- This is the main recursion statement. Every repetition contains instructions. -->
    <!-- Every instruction can contain a repetition -->
    <xs:element name="instruction" minOccurs="1" maxOccurs="unbounded" type="instructionType">
      <xs:annotation>
        <xs:documentation>A repetition can include any number of instructions, which in turn can
 contain repetitions or continues.</xs:documentation>
      </xs:annotation>
      <xs:unique name="repEquipmentUnique">
        <xs:annotation>
          <xs:documentation>Ensures all equipment values in an instruction are unique</
xs:documentation>
        </xs:annotation>
        <xs:selector xpath="./sw:equipment"/>
        <xs:field xpath="."/>
      </xs:unique>
    </xs:element>
  </xs:sequence>
  <!-- ================================================== -->
  <!-- Assertions -->
  <xs:assert test="            if(./sw:simplify = true())then(                (
                                every $instruction in ./sw:instruction[not(./
sw:pyramid or ./sw:segmentName)] satisfies(                        (if($instruction/
descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./
sw:continue/sw:continueLength) and not(./sw:repetition)]) then(
     if(count($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/
sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) = 1) then(
                              number(                                ($instruction/descendant-
or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/
sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-
self::*/sw:length)[last()]/sw:lengthAsDistance                                    )
                  ) else(                                sum(
  ($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and
 not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length |
 ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance                                    )
                       )                            ) else(                            0
             )+(                                if($instruction/descendant-or-self::sw:continueLength)
 then(                                number($instruction/descendant-or-self::sw:continueLength)
                           ) else(                                    0                            )
                        )) = number(                                (
  (                               (./descendant-or-self::sw:instruction[not(ancestor::sw:continue/
sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//
(preceding-sibling::sw:length | ancestor-or-self::*/sw:length)[last()]/sw:lengthAsDistance
                                ) | (                                        ./descendant-or-
self::sw:continueLength                                    )                            )[1]
                  )                     )                )or(                     every $instruction
 in ./sw:instruction[not(./sw:pyramid or ./sw:segmentName)] satisfies(
  (if($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength)
 and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) then(
          if(count($instruction/descendant-or-self::sw:instruction[not(ancestor::sw:continue/
sw:continueLength) and not(./sw:continue/sw:continueLength) and not(./sw:repetition)]) = 1) then(
                              number(                                ($instruction/descendant-
or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./sw:continue/
sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length | ancestor-or-
self::*/sw:length)[last()]/sw:lengthAsLaps                                    )
       ) else(                                    sum(                                ($instruction/
descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and not(./
sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length |
 ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps                                    )
                       )                            ) else(                            0
             )+(                                if($instruction/descendant-or-self::sw:continueLength)
 then(                                number(                                $instruction/
descendant-or-self::sw:continueLength                                )
   ) else(                                0                                )
  )) = number(                                (                                    (
            (./descendant-or-self::sw:instruction[not(ancestor::sw:continue/sw:continueLength) and
 not(./sw:continue/sw:continueLength) and not(./sw:repetition)][1])//(preceding-sibling::sw:length |
 ancestor-or-self::*/sw:length)[last()]/sw:lengthAsLaps                                ) | (
                                                        ./descendant-or-self::sw:continueLength
                                )                                )[1]                            )
```

```
          )                )            )else(                true()            )
 ">
    <xs:annotation>
      <xs:documentation>checks all lengths within simplified repetition are the same Does this by
 using two seperate statments so that either all lengthAsDistances are the same or all lengthAsLaps
 checks that every instruction, repetition and continue has the same base length e.g. 100m , 4x100m
 and 50,50 swim as 100 ; these are all of length 100 so can be simplified in the same repetition
 has to choose whether to use continueLength or add up all instructions within a continue</
xs:documentation>
    </xs:annotation>
  </xs:assert>
  <!-- Explain what this assertion does -->
  <xs:assert test="./sw:simplify[text() = 'true'] or ./sw:repetitionCount and not(./
sw:simplify[text() = 'true'] and ./sw:repetitionCount)"/>
</xs:complexType>
```

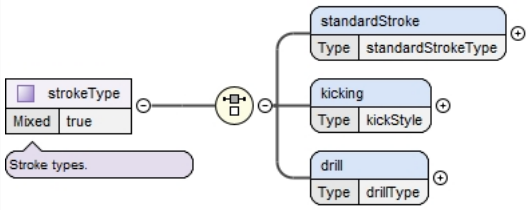
Complex Type `lengthType`

Namespace	https://github.com/bartneck/swiML
Annotations	The length for a swimming instruction.
Diagram	
Properties	mixed: true
Used by	Elements: continueType/continueLength, instructionGroup/length, pyramidType/startLength, pyramidType/stopLength
Model	lengthAsDistance | lengthAsTime | lengthAsLaps
Children	lengthAsDistance, lengthAsLaps, lengthAsTime
Source	

```
<xs:complexType name="lengthType" mixed="true">
  <xs:annotation>
    <xs:documentation>The length for a swimming instruction.</xs:documentation>
  </xs:annotation>
  <!-- Length as either distance, laps or time -->
  <xs:choice>
    <xs:annotation>
      <xs:documentation>Length can be descriped as distance or time.</xs:documentation>
    </xs:annotation>
    <xs:element name="lengthAsDistance" type="xs:nonNegativeInteger">
      <xs:annotation>
        <xs:documentation>Length of instruction as distance.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="lengthAsTime" type="xs:duration">
      <xs:annotation>
        <xs:documentation>Duration starts with PT followed by int M and int S. For example PT1M30S
 for 1:30.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="lengthAsLaps" type="xs:nonNegativeInteger">
      <xs:annotation>
        <xs:documentation>Length of instruction in number of laps.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

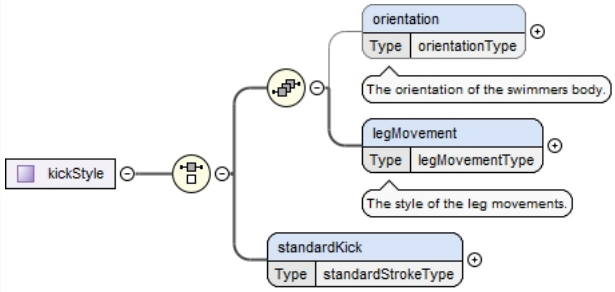
Complex Type `strokeType`

Namespace	https://github.com/bartneck/swiML
Annotations	Stroke types.

Diagram	
Properties	mixed: `true`
Used by	Element instructionGroup/stroke
Model	standardStroke | kicking | drill
Children	drill, kicking, standardStroke

Source	`<xs:complexType name="strokeType" mixed="true">`
	`  <xs:annotation>`
	`  <xs:documentation>Stroke types.</xs:documentation>`
	`  </xs:annotation>`
	`  <xs:choice>`
	`  <xs:element name="standardStroke" type="standardStrokeType"/>`
	`  <xs:element name="kicking" type="kickStyle"/>`
	`  <xs:element name="drill" type="drillType"/>`
	`  </xs:choice>`
	`</xs:complexType>`

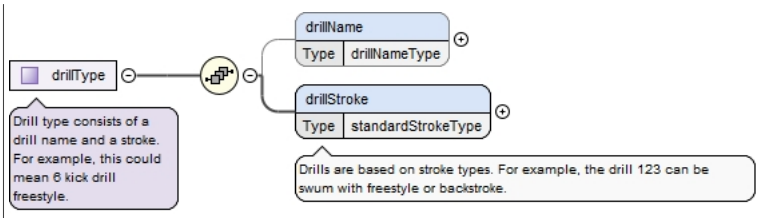
Complex Type `kickStyle`

Namespace	https://github.com/bartneck/swiML
Diagram	
Used by	Element strokeType/kicking
Model	(orientation{0,1} , legMovement) | standardKick
Children	legMovement, orientation, standardKick

Source	`<xs:complexType name="kickStyle">`
	`  <xs:choice>`
	`  <xs:sequence>`
	`  <xs:element name="orientation" type="orientationType" minOccurs="0" maxOccurs="1">`
	`  <xs:annotation>`
	`  <xs:documentation>The orientation of the swimmers body.</xs:documentation>`
	`  </xs:annotation>`
	`  </xs:element>`
	`  <xs:element name="legMovement" type="legMovementType" minOccurs="1" maxOccurs="1">`
	`  <xs:annotation>`
	`  <xs:documentation>The style of the leg movements.</xs:documentation>`
	`  </xs:annotation>`
	`  </xs:element>`
	`  </xs:sequence>`
	`  <xs:element name="standardKick" minOccurs="1" maxOccurs="1" type="standardStrokeType"/>`
	`  </xs:choice>`
	`</xs:complexType>`

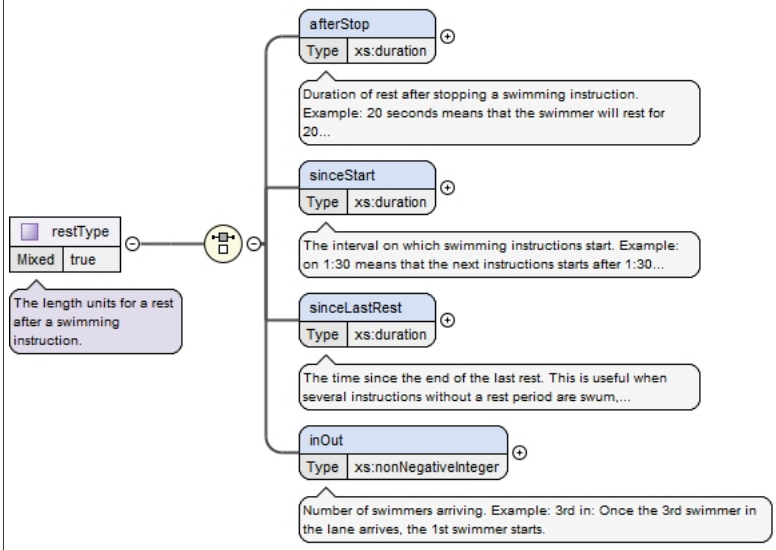
Complex Type `drillType`

Namespace	https://github.com/bartneck/swiML
Annotations	`Drill type consists of a drill name and a stroke. For example, this could mean 6 kick drill freestyle.`

Diagram	
Used by	Element strokeType/drill
Model	drillName{0,1} , drillStroke
Children	drillName, drillStroke

Source	`<xs:complexType name="drillType">`
	`  <xs:annotation>`
	`  <xs:documentation>Drill type consists of a drill name and a stroke. For example, this could mean 6 kick drill freestyle.</xs:documentation>`
	`  </xs:annotation>`
	`  <xs:sequence>`
	`  <xs:element name="drillName" minOccurs="0" maxOccurs="1" type="drillNameType"/>`
	`  <xs:element name="drillStroke" type="standardStrokeType" maxOccurs="1" minOccurs="1">`
	`  <xs:annotation>`
	`  <xs:documentation>Drills are based on stroke types. For example, the drill 123 can be swum with freestyle or backstroke.</xs:documentation>`
	`  </xs:annotation>`
	`  </xs:element>`
	`  </xs:sequence>`
	`</xs:complexType>`

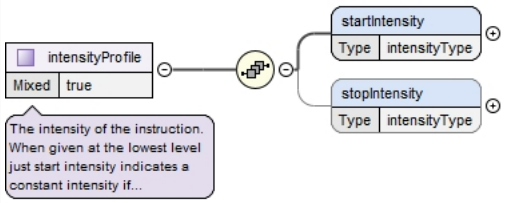
Complex Type `restType`

Namespace	https://github.com/bartneck/swiML
Annotations	The length units for a rest after a swimming instruction.
Diagram	
Properties	mixed: true
Used by	Element instructionGroup/rest
Model	afterStop | sinceStart | sinceLastRest | inOut
Children	afterStop, inOut, sinceLastRest, sinceStart

Source	`<xs:complexType name="restType" mixed="true">`
	`  <xs:annotation>`
	`  <xs:documentation>The length units for a rest after a swimming instruction.</xs:documentation>`
	`  </xs:annotation>`
	`  <xs:choice>`
	`  <xs:element name="afterStop" type="xs:duration">`
	`  <xs:annotation>`
	`  <xs:documentation>Duration of rest after stopping a swimming instruction. Example: 20 seconds means that the swimmer will rest for 20 seconds after stopping the current instructions.</xs:documentation>`
	`  </xs:annotation>`

```
    </xs:element>
    <xs:element name="sinceStart" type="xs:duration">
      <xs:annotation>
        <xs:documentation>The interval on which swimming instructions start. Example: on 1:30
 means that the next instructions starts after 1:30 from starting the current instruction.</
xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="sinceLastRest" type="xs:duration">
      <xs:annotation>
        <xs:documentation>The time since the end of the last rest. This is useful when
 several instructions without a rest period are swum, followed by a since start type rest.</
xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="inOut" type="xs:nonNegativeInteger">
      <xs:annotation>
        <xs:documentation>Number of swimmers arriving. Example: 3rd in: Once the 3rd swimmer in the
 lane arrives, the 1st swimmer starts.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

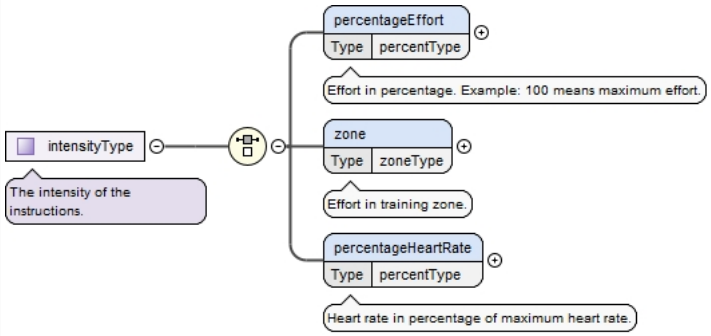
Complex Type `intensityProfile`

Namespace	https://github.com/bartneck/swiML
Annotations	The intensity of the instruction. When given at the lowest level just start intensity indicates a constant intensity if the stop intensity is given then it is a build within the instruction If the intensity is given at a higher level (repetition or continue) just start intensity is the same constant for all child instructions given a stop intensity then it is descending/ascending over the child instructions
Diagram	
Properties	mixed: `true`
Used by	Element instructionGroup/intensity
Model	startIntensity , stopIntensity{0,1}
Children	startIntensity, stopIntensity
Source	

```
<xs:complexType name="intensityProfile" mixed="true">
  <xs:annotation>
    <xs:documentation>The intensity of the instruction. When given at the lowest level just start
 intensity indicates a constant intensity if the stop intensity is given then it is a build within
 the instruction If the intensity is given at a higher level (repetition or continue) just start
 intensity is the same constant for all child instructions given a stop intensity then it is
 descending/ascending over the child instructions</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="startIntensity" minOccurs="1" maxOccurs="1" type="intensityType"/>
    <xs:element name="stopIntensity" minOccurs="0" maxOccurs="1" type="intensityType"/>
  </xs:sequence>
</xs:complexType>
```

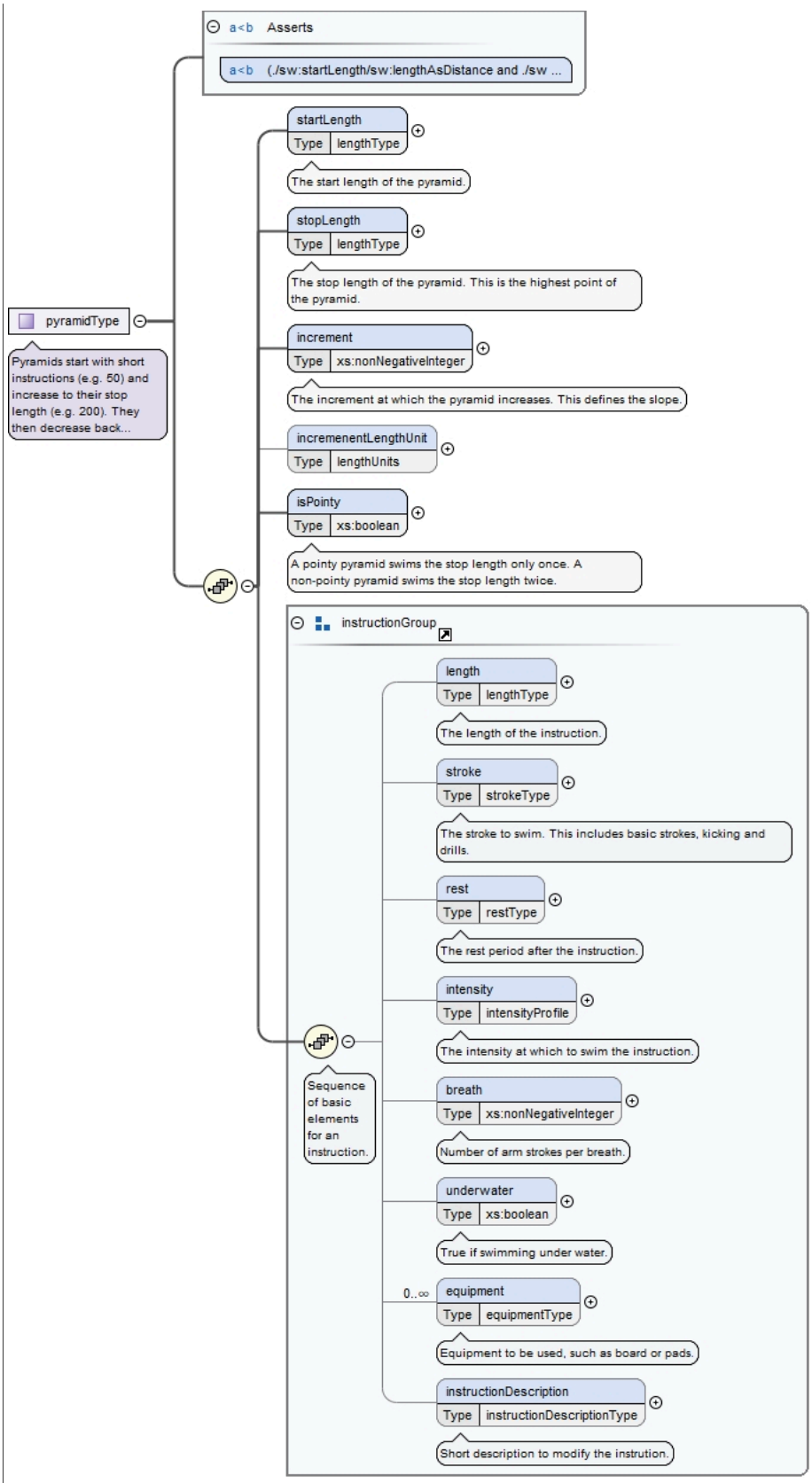
Complex Type `intensityType`

Namespace	https://github.com/bartneck/swiML
Annotations	The intensity of the instructions.

Diagram	
Used by	Elements intensityProfile/startIntensity, intensityProfile/stopIntensity
Model	percentageEffort | zone | percentageHeartRate
Children	percentageEffort, percentageHeartRate, zone
Source	<xs:complexType name="intensityType"> <xs:annotation> <xs:documentation>The intensity of the instructions.</xs:documentation> </xs:annotation> <xs:choice> <xs:element name="percentageEffort" type="percentType"> <xs:annotation> <xs:documentation>Effort in percentage. Example: 100 means maximum effort.</ xs:documentation> </xs:annotation> </xs:element> <xs:element name="zone" type="zoneType"> <xs:annotation> <xs:documentation>Effort in training zone.</xs:documentation> </xs:annotation> </xs:element> <xs:element name="percentageHeartRate" type="percentType"> <xs:annotation> <xs:documentation>Heart rate in percentage of maximum heart rate.</xs:documentation> </xs:annotation> </xs:element> </xs:choice> </xs:complexType>

Complex Type `pyramidType`

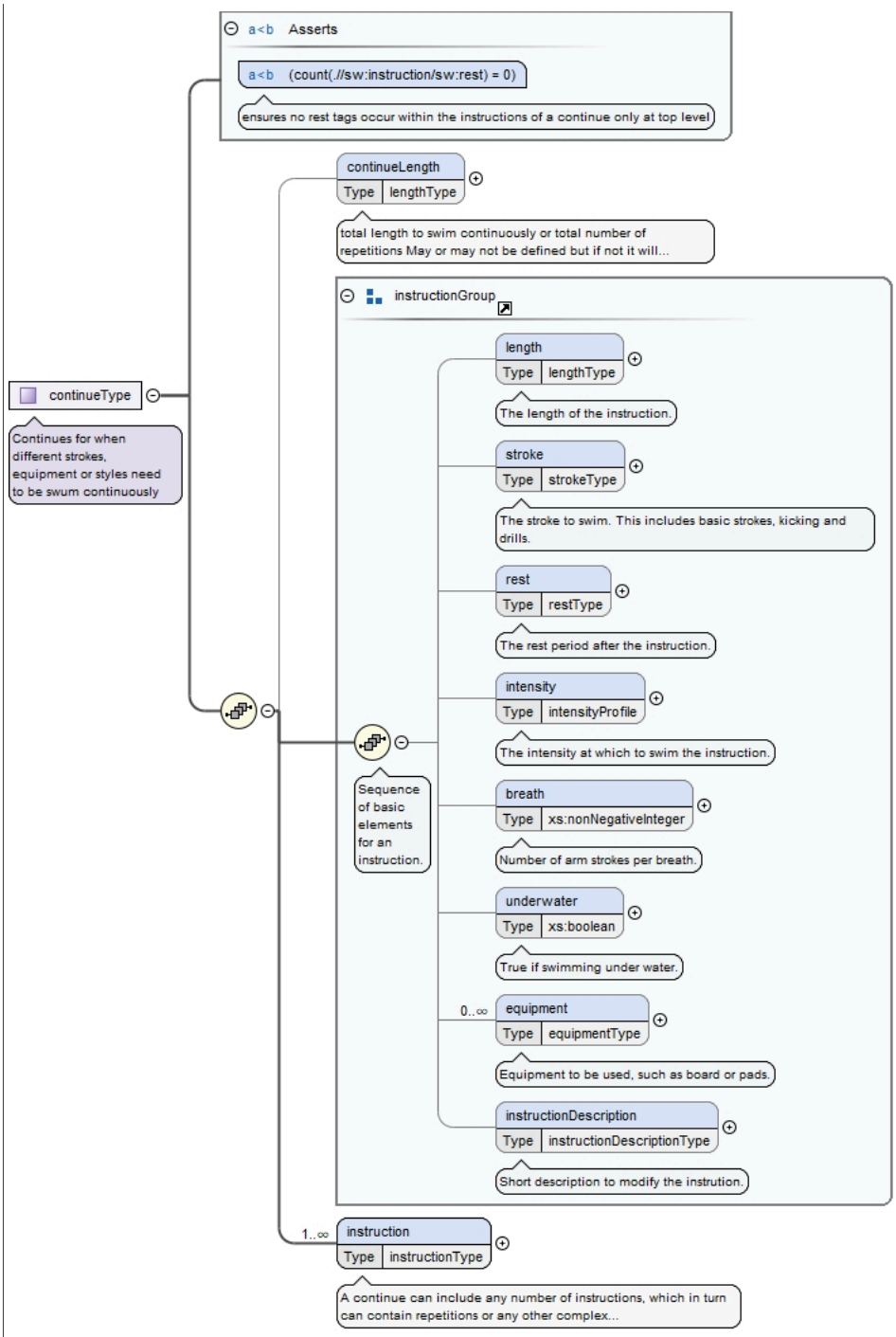
Namespace	https://github.com/bartneck/swiML
Annotations	Pyramids start with short instructions (e.g. 50) and increase to their stop length (e.g. 200). They then decrease back to the start length.

Diagram	
Used by	Element instructionType/pyramid
Model	startLength , stopLength , increment , incremenentLengthUnit{0,1} , isPointy , length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1}
Children	breath, equipment, incremenentLengthUnit, increment, instructionDescription, intensity, isPointy, length, rest, startLength, stopLength, stroke, underwater

<table>
<tr><td>Asserts</td><td><table>
<tr><th>Test</th><th>XPath default namespace</th></tr>
<tr><td>(./sw:startLength/sw:lengthAsDistance and ./sw:stopLength/sw:lengthAsDistance) or (./sw:startLength/sw:lengthAsLaps and ./sw:stopLength/sw:lengthAsLaps) or (./sw:startLength/sw:lengthAsTime and ./sw:stopLength/sw:lengthAsTime)</td><td></td></tr>
</table></td></tr>
<tr><td>Source</td><td><pre>
<xs:complexType name="pyramidType">
  <xs:annotation>
    <xs:documentation>Pyramids start with short instructions (e.g. 50) and increase to their stop length (e.g. 200). They then decrease back to the start length.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="startLength" minOccurs="1" maxOccurs="1" type="lengthType">
      <xs:annotation>
        <xs:documentation>The start length of the pyramid.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="stopLength" minOccurs="1" maxOccurs="1" type="lengthType">
      <xs:annotation>
        <xs:documentation>The stop length of the pyramid. This is the highest point of the pyramid.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="increment" minOccurs="1" maxOccurs="1" type="xs:nonNegativeInteger">
      <xs:annotation>
        <xs:documentation>The increment at which the pyramid increases. This defines the slope.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="incremenentLengthUnit" type="lengthUnits" minOccurs="0" maxOccurs="1"/>
    <xs:element name="isPointy" minOccurs="1" maxOccurs="1" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>A pointy pyramid swims the stop length only once. A non-pointy pyramid swims the stop length twice.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:group ref="instructionGroup"/>
  </xs:sequence>
  <xs:assert test="   (./sw:startLength/sw:lengthAsDistance and ./sw:stopLength/sw:lengthAsDistance)   or (./sw:startLength/sw:lengthAsLaps and ./sw:stopLength/sw:lengthAsLaps)   or (./sw:startLength/sw:lengthAsTime and ./sw:stopLength/sw:lengthAsTime)   "/>
</xs:complexType>
</pre></td></tr>
</table>

Complex Type `continueType`

<table>
<tr><td>Namespace</td><td>https://github.com/bartneck/swiML</td></tr>
<tr><td>Annotations</td><td>Continues for when different strokes, equipment or styles need to be swum continuously</td></tr>
</table>

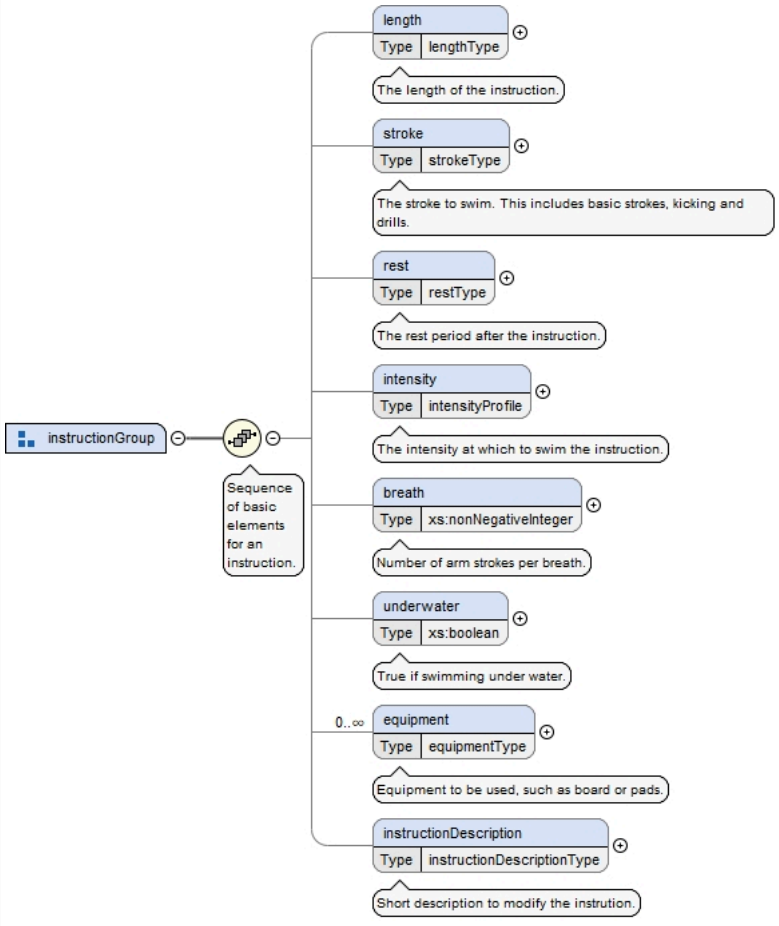
Diagram	
Used by	Element instructionType/continue
Model	continueLength{0,1} , length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1} , instruction+
Children	breath, continueLength, equipment, instruction, instructionDescription, intensity, length, rest, stroke, underwater
Asserts	**Test** | **XPath default namespace** (count(.//sw:instruction/sw:rest) = 0) | `ensures no rest tags occur within the instructions of a continue only at top level`

Source	`<xs:complexType name="continueType">`
	`<xs:annotation>`
	`<xs:documentation>Continues for when different strokes, equipment or styles need to be swum continuously</xs:documentation>`
	`</xs:annotation>`
	`<xs:sequence>`

```
    <xs:element name="continueLength" minOccurs="0" maxOccurs="1" type="lengthType">
      <xs:annotation>
        <xs:documentation>total length to swim continuously or total number of repetitions May
 or may not be defined but if not it will automatically calculated from given instructions</
xs:documentation>
      </xs:annotation>
    </xs:element>
    <!-- Common elements for instructions -->
    <xs:group ref="instructionGroup"/>
    <xs:element name="instruction" minOccurs="1" maxOccurs="unbounded" type="instructionType">
      <xs:annotation>
        <xs:documentation>A continue can include any number of instructions, which in turn can
 contain repetitions or any other complex instruction type.</xs:documentation>
      </xs:annotation>
      <xs:unique name="contEquipmentUnique">
        <xs:annotation>
          <xs:documentation>Ensures all equipment values in an instruction are unique</
xs:documentation>
        </xs:annotation>
        <xs:selector xpath="./sw:equipment"/>
        <xs:field xpath="."/>
      </xs:unique>
    </xs:element>
  </xs:sequence>
  <!-- ======================================================== -->
  <!-- Assertions -->
  <!-- Explain what this assertion does -->
  <xs:assert test="(count(.//sw:instruction/sw:rest) = 0)">
    <xs:annotation>
      <xs:documentation>ensures no rest tags occur within the instructions of a continue only at top
 level</xs:documentation>
    </xs:annotation>
  </xs:assert>
</xs:complexType>
```

Element Group(s)

Element Group `instructionGroup`

Namespace	https://github.com/bartneck/swiML
Diagram	

Used by	Complex Types continueType, instructionType, pyramidType, repetitionType
Model	length{0,1} , stroke{0,1} , rest{0,1} , intensity{0,1} , breath{0,1} , underwater{0,1} , equipment* , instructionDescription{0,1}
Children	breath, equipment, instructionDescription, intensity, length, rest, stroke, underwater
Source	

```
<xs:group name="instructionGroup">
  <xs:sequence>
    <xs:annotation>
      <xs:documentation>Sequence of basic elements for an instruction.</xs:documentation>
    </xs:annotation>
    <xs:element name="length" minOccurs="0" maxOccurs="1" type="lengthType">
      <xs:annotation>
        <xs:documentation>The length of the instruction.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="stroke" minOccurs="0" maxOccurs="1" type="strokeType">
      <xs:annotation>
        <xs:documentation>The stroke to swim. This includes basic strokes, kicking and drills.</
xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="rest" minOccurs="0" maxOccurs="1" type="restType">
      <xs:annotation>
        <xs:documentation>The rest period after the instruction.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="intensity" minOccurs="0" maxOccurs="1" type="intensityProfile">
      <xs:annotation>
        <xs:documentation>The intensity at which to swim the instruction.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="breath" minOccurs="0" maxOccurs="1" type="xs:nonNegativeInteger">
      <xs:annotation>
        <xs:documentation>Number of arm strokes per breath.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="underwater" minOccurs="0" maxOccurs="1" type="xs:boolean">
      <xs:annotation>
        <xs:documentation>True if swimming under water.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="equipment" minOccurs="0" maxOccurs="unbounded" type="equipmentType">
      <xs:annotation>
        <xs:documentation>Equipment to be used, such as board or pads.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="instructionDescription" type="instructionDescriptionType" minOccurs="0"
 maxOccurs="1">
      <xs:annotation>
        <xs:documentation>Short description to modify the instrution.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:group>
```