Exam Papers Revisited

Posing Questions To Students

Abstract
Described is a module for the consistent production and maintenance of student examinations. It can typeset questions with long or short answers, yes/no questions and multiple choice. The questions are formulated as XML documents and access ConTEXt through a special interface with HTML-like syntax.

Keywords
exam, examination, collection, problem, question, multiple choice, ConTeXt, XML, HTML

Introduction

The ConTEXt `hvdm-xam` module aims at easy and consistent typesetting of student exams and maintaining collections of exam questions. It especially facilitates questions with short answers and multiple choice. Many aspects of the typesetting are configurable. The module depends on two other modules `hvdm-xml` and `hvdm-lua`. The former provides the bridge between HTML-style formatting expressions and the ConTEXt-world, the latter contains support functions that are better programmed in the programming language Lua than in TEX.

This module is a followup of the previous `hvdm-exm` module, which in turn was an upgrade to ConTEXt of an older LATEX package. After a period of relative inactivity in using my own module, it turned out that working in ConTEXt-MKII did not automatically meant working in ConTEXt-MKIV. Nothing to be surprised of, because after all ConTEXt is a fast moving target and incompatibilities are the price one has to pay for using it. Moreover, it was one of the incentives that made me move forward on the path of separating data and program code as much as possible.

In this latest overhaul of the exam producing module, the questions in the exam are no longer in TEX or ConTEXt, but in XML. A change that to a great extent decouples the problem description from the typesetting machinery. TEX code might sometimes be indispensible to achieve high quality math formulae and can be conveniently inserted. Using MathML is another option. In a sense, this article can be seen as an updated version of the earlier one published in the NTG-MAPS number 36, spring 2008.

Exam structure

Exams need to be typeset in different formats. First of course is the format in which the questions will be presented to the students taking an examination. Secondly, one can produce the same with the answers added to the questions. In my experience this is useful for scoring student results, especially when the questions are divided in several parts each meriting a reward for a correct answer. Finally one can produce a catalogue of all questions including the answers, annotations and scores. In between there is a lot of flexibility. The product can be customized through attributes on the nodes of the XML data descriptions. A small built in and simple to adapt and extend vocabulary of common language dependent terms eases localization of the product. Defining this vocabulary for a specific language allows one to typeset them automatically in ConTEXt's current active language.

The typesetting of exams is structured in three layers. The top layer is a CONTEXT file driving the typesetting. Do font setup and other such things here. A short example is given below.

```
% Load exam module
% includes [mathml][hvdm-lua][hvdm-xml]
\usemodule[hvdm-xam]
% papersize, layout, font, language, etc.
\def\prologue{..}% execute before exam
\def\epilogue{..}% execute after exam
\starttext
% Call root node <exam> in examfile.xml
\xmlprocessfile{exam}{examfile.xml}{}
\stoptext
```

The above ConTEXt program processes `examfile.xml`, which is the file harbouring the `<exam>` root. Within that root are nodes that call files for each problem. The example below clarifies the structure of this file. Nodes `<title>`, `<date>`, `<time>` and `<location>` define macros `\thetitle`, `\thedate`, `\thetime` and `\thelocation`, that can be used in the `\prologue` and `\epilogue` code figuring in the example above. How to have these called will be explained shortly. But, as can be seen in the example, it is also possible to use

these nodes directly in composing a header to the exam and even have them preceded by a language localized prefix. The bonus for having date, etc. in nodes like <date> is the possibility to process a bunch of exams within another XML structure with direct access to these data.

```
<?xml version="1.0" encoding="UTF-8"?>
<exam attributes>
  <p align="middle">
    <title>So and So Exam</title><br/>
    <date>2099-09-09</date>
    - <time>13:00-16:00</time><br/>
    <location Pre=": ">
      Main building lounge
    </location>
  </p>
  <file name="dirAAA/question-01.xml"/>
  ...
  <file name="dirZZZ/question-99.xml"/>
</exam>
```

So and So Exam
2099-09-09 – 13:00–16:00
Location: Main building lounge
"EXAM"

There are a few commands useful in the introductory text to the exam. These are:

1. <currentdate/> inserts the current date in the format yyyy-mm-dd, as in 12-07-2013

2. <date>content</date> for setting *content* as date and at the same time remembering it. The date may either be given as yyyymmdd in which case it will be formatted as above, or in completely free format. Use the attribute show="no" to set the stored value without displaying.

3. <currenttime/> inserts the current time in the format hh:mm in a 24 hour clock, as in 11:26.

4. <time> the same as the date equivalent.

5. <title> place and setup a title for the exam.

6. <location/> place and setup a location for the exam.

The nodes <date>, <time>, <title>, <location> can carry an attribute pre="" or Pre="". If present the language equivalent of the node name is typeset before the contents. The capitalization of the attribute determines the capitalization of the keyword. The value of the attribute may be empty, but if there is content then the value of the attribute will be placed before it. An example can be seen above in the location node.

Problem structure

Each problem description should get its own file, possibly in different directories. The name attribute of the <file> node locates the file, taking its origin in the directory that contains the caller. Alternatively, one may place the data in a ConTEXt buffer and call it up with a <buffer> node instead of a <file> node, as is done in the source of this article. See the heavily shortened example below.

```
\startbuffer[example]
  <?xml version="1.0" encoding="UTF-8"?>
  <problem>
    .. content of problem ..
  </problem>
\stopbuffer
\startbuffer[caller]
  <?xml version="1.0" encoding="UTF-8"?>
  <exam>
    <buffer name="example"/>
  </exam>
\stopbuffer
\xmlprocessbuffer{exam}{caller}{}
```

The problems themselves have the structure shown in the next example code. The root is a <problem> node and within it are nodes for specific information and one or more <question> nodes. If there are several questions in the problem, they can either be alternatives or they can comprise a series of questions all to be answered. Most nodes can be placed in any order. But of course the ordering matters for a series of successive questions.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem category="bachelor">
  <subject>Literature</subject>
  <description>About Shakespeare.</description>
  <history date="Earlier">First version</history>
  <history date="20120322">Was changed</history>
   <text>Text of question here.</text>
   <question score="1">
     <note>An example only.</note>
     ... <answer> nodes here ...
  </question>
  <note>Final note.</note>
</problem>
```

Buffer: example-0
Category: bachelor
Subject: Literature
Description: About Shakespeare
Maxscore: 1 point

History

Earlier: First version
22-03-2012: Was changed

Notes

1. An example only.
2. Final note.

Problem-1 Text of question here.

The purpose of the `<subject>` `<description>` and `<history>` nodes needs no further explanation. They are expected at the top of the node tree, i.e. directly inside the `<problem>` node. As can be seen, dates may be given either in free form or as `yyyymmdd`. Annotations given as `<note>` may occur everywhere, but for printing they are collected and output in one place. The `<question>` node has an attribute for the score that goes with the correct answer. Scores can either be shown or suppressed on the printout. In this article the printing of scores is suppressed, because they do not cope well with a multicolumn layout.

All information like `<subject>` etc. regarding the problem is placed inside a framed area having three sections.

Characteristic data, like the name of the data file, a short description, the subject matter, the maximum number of points allotted, etc.
The annotation part when there is at least one `<note>` present.
The version history of the problem.

Options are provided to determine what parts of this information should appear, if at all.

Question structure

There are five types of questions programmed:

1. `<shortanswer>`
2. `<altanswer>`
3. `<listanswer>`
4. `<blockanswer>`
5. `<choiceanswer>`

All will subsequently be described.

Short Answer

The first example shows the programming of a `<shortanswer>` question. Note that in this specific example the `<problem>` node carries the optional `category` attribute by which one can differentiate between various target groups. Specify on exam production a corresponding filter – for this example `<exam filter="bachelor">` – to filter out problems of this category and suppress all others.

The `<shortanswer>` example is presented here three times. First how it is given to students at the examination, then the same with answers filled in and finally how it will appear in a catalogue of problems. This example also demonstrates the use of consecutive questions preceded by an introductory `<text>` node. As already has been mentioned, score value are not shown but this problem will have 3 score points reported.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem category="bachelor">
  <text>Written by Shakespeare.</text>
  <question score="2">
    <text>
      What is the most famous question?
    </text>
    <shortanswer>
      To be or not to be.
    </shortanswer>
  </question>
  <question score="1">
    <text>Who said that?</text>
    <shortanswer>Hamlet.</shortanswer>
  </question>
  <note>Just an example.</note>
</problem>
```

Problem-1 Written by Shakespeare.
What is the most famous question?
Answer: .....

Who said that?
Answer: .....

Problem-1 Written by Shakespeare.
What is the most famous question?
Answer: To be or not to be.

Who said that?
Answer: Hamlet.

Buffer: example-1
Category: bachelor
Maxscore: 3 points
Alternatives: no

Note
Just an example.

Problem-1 Written by Shakespeare.
What is the most famous question?

Answer : To be or not to be.

Who said that?

Answer : Hamlet.

Alternate answers

The `<altanswer>` type of question is meant for simple alternatives, like yes/no questions. The example also demonstrates the use of different colors for good and the wrong answers. Understandably they show up in the answered version only.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
  <text>
    Was something rotten in the state of Denmark?
  </text>
  <question score="1">
    <altanswer>
      <item value="true">yes</item>
      <item value="false">no</item>
    </altanswer>
  </question>
</problem>
```

Problem-1 Was something rotten in the state of Denmark?

Markgood: yes / no

Problem-1 Was something rotten in the state of Denmark?

Markgood: yes / ~~no~~

List of answers

The `<listanswer>` is meant for a series of short questions. Each question has some small answer that should be filled in by the student.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
  <text>
    Finish the following three statements on
    quotation "To be or not to be":
  </text>
  <question score="1">
    <listanswer>
      <item>
        <text>this quote is from</text>
        <value>Hamlet</value>
      </item>
      <item> .... </item>
      <item> .... </item>
  </listanswer>
  </question>
</problem>
```

Problem-1 Finish these statements on "To be or not to be"

1. this quote is from .....
2. of the English writer .....
3. supposedly born at .....

Problem-1 Finish these statements on "To be or not to be"

1. this quote is from Hamlet
2. of the English writer Shakespeare
3. supposedly born at Stratford-on-Avon

Block answer

The `<blockanswer>` is meant for questions needing more space for the answer. Space can be reserved below the question or a separate answer sheet can be given. For the latter the answer block will be completely suppressed when answer and prompt are off both. That case is illustrated first, thereafter the prompt is kept and a given amount of space reserved for the answer.

Problem-1 Elaborate on the question "To be or not to be?"

Problem-1 Elaborate on the question "To be or not to be?"

Answer:

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
  <text>
    Elaborate on the question:
    "To be or not to be"?
  </text>
  <question score="1">
    <blockanswer frame="on" height="2cm">
      This famous quote is from <i>Hamlet</i>,
      a play of the English writer William
      Shakespeare. ....
    </blockanswer>
  </question>
</problem>
```

It is even possible to fill the block with a series of dotted lines. To accomplish this the height attribute on the `<blockanswer>` is set to zero and the `lines` attribute to the number of lines.

Problem-1 Elaborate on the question "To be or not to be?"

Answer: .....
.....
.....

Finally here follows the output for the case where `answer="on"`.

Problem-1 Elaborate on the question "To be or not to be?"

Answer: This famous quote is from *Hamlet*, a play of the English writer William Shakespeare. One really has to suspect that nowadays not many students have seen even one of Shakespeare's plays, let alone having read one. Most might not even know when or where William Shakespeare is supposed to have been born.

Multiple choice answers

Multiple choice questions are formulated in a `<choiceanswer>`. The items can be presented to the students in the given or in random order. In the first example below the order has been randomized; see the difference with the second one. The `random` attribute on the `<exam>` node governs this behaviour. Attribute `randomseed` on `<exam>` enables one to set the start of the random generator.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
  <text>
    Which person is found in <i>Hamlet</i>?
  </text>
  <question score="1">
    <choiceanswer>
      <item value="true">Ophelia</item>
      <item value="false">Rosalind</item>
      <item value="false">Desdemona</item>
      <item value="false">Juliet</item>
    </choiceanswer>
  </question>
</problem>
```

Problem-1 Which person is found in *Hamlet*?

- ☐ Juliet
- ☐ Desdemona
- ☐ Ophelia
- ☐ Rosalind

Problem-1 Which person is found in *Hamlet*?

- ☐ Rosalind
- ☐ Juliet
- ☐ Desdemona
- ☒ Ophelia

Other examples

Here follow some examples showing various possibilities of the package. A more systematic description of the nodes involved is presented after this section.

Define and use of mathml

This example shows how a MathML coded formula can be defined at the `<problem>` level, then used twice in the subsequent code.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
```

```
<define name="parabola">
<math xmlns='http://www.w3c.org/mathml'
    version='2.0'>
  <apply><eq/>
    <ci>y</ci>
    <apply><plus/>
      <apply><power/>
        <ci>ax</ci><cn>2</cn>
      </apply>
      <ci>bx</ci><ci>c</ci>
   </apply></apply></math>
</define>
<question score="1">
  <text>
    How is this function called?<br/>
    <define name="parabola"/>
  </text>
  <shortanswer>
    <define name="parabola"/> is a parabola
  </shortanswer>
</question>
</problem>
```

Problem-1 How is this function called?
$y = ax^2 + bx + c$

Answer: $y = ax^2 + bx + c$ is a parabola

Messages

Messages are typeset as XML nodes and can serve as reminders. To name two examples: (1) as a note for the corrector that special attention is needed here, (2) as a reminder that the problem is still under construction. Messages are not shown in case answers are suppressed, because the answer flag is checked and must be true in order to get the message through. Thus by default messages are absent from exams to be handed out, but visible on the correction sheet. Another flag can takeover the role of guardian, for example: `<message flag="series"..>` will typeset the message when the series flag has been set true. Messages can be colored and given another tag, as can be seen in the example.

Problem-1 Which person is found in *Hamlet*?

☒ Ophelia

TODO: 3 answers missing

The following example shows how an error is presented. Here caused by the absence of `<item>` nodes in the multiple choice.

Problem-1 Which person is found in *Hamlet*?
ERROR <choiceanswer> no items (input source = example-9)

Random selection

Multiple choice questions have their items shuffled at random by setting the `random` attribute on the `<exam>` to `on`. Other decisions can be randomized too. For example, a question can have two variants where a random choice is appropriate. The `<random>` node does just that: choosing at random from the nodes directly inside. But what if a second, related question in the same problem must be synchronized with that choice? In that case the last random choice can be stored and called up later on. The next example, albeit a little contrived, illustrates the principle. A first `<random save="yes">` generates the random selection and saves its value, the following `<random reuse="yes">` uses this value again. In more intricate cases, one can even remember several random choices by replacing the `yes` with a name.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
  <random save="yes">
  <text>First text</text>
  <text>Second text</text>
  <text>Third text</text>
  </random>
  <blockanswer>
    <random reuse="yes">
    <text>First text again</text>
    <text>Second text again</text>
    <text>Third text again</text>
    </random>
  </blockanswer>
</problem>
```

Problem-1 Third text

Answer: Third text again

Hide and show

The following example illustrates how parts can be selectively shown and hidden. The `<show>` and `<hide>` nodes take the name of a flag (default: `answer`) as an attribute triggering appearance and disappearance, respectively. In the example the flag is the otherwise unused `user` flag. A `<show>` node will present its content when the `flag=on` and suppresses it for `flag=off`. In contrast `<hide>` hides for `flag=on` and shows for `flag=off`.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem>
  <blockanswer user="on">
    <hide flag="user">
      <text>&lt;hide&gt;: flag=on</text>
    </hide>
    <show flag="user">
      <text>&lt;show&gt;: flag=on</text>
    </show>
  </blockanswer>
  <blockanswer user="off">
    <hide flag="user">
      <text>&lt;hide&gt;: flag=off</text>
    </hide>
    <show flag="user">
      <text>&lt;show&gt;: flag=off</text>
    </show>
  </blockanswer>
</problem>
```

Problem-1

Answer: <show>: flag=on

Answer: <hide>: flag=off

Nodes at the <exam> level

This and the next section describe the attributes that various nodes can have. The presentation takes the form <node attributes>, where attributes is a list of one or more attribute-value pairs.

<exam attributes>

The <exam> node has quite a number of attributes. Many of these provide an opportunity to customize the type, specific content and formatting style of the product. If the attribute has a default, that one is presented with the name of the attribute.

1. init="exam"
   This attribute determines the product type. Also switches on by default some flags. Attribute values are:
   - exam examination paper without answers
   - answer examination paper with answers
   - series catalogue of questions in all variants, with answers and full data

2. base="./"
   This attribute states the directory from where the search for included problem and other data files begins. By default the home of the calling tex-file is taken as point of departure. Macro \currentdirectory will be defined to the value of the attribute. Note that the value must end with a directory separator /. Besides affecting file reading and include processing, it is also doing a setup for the figures directory with
   ```
   \setupexternalfigures[
     location=global,
     directory={\currentdirectory}]
   ```

3. prologue="" epilogue=""
   Code executed before and after processing of the problem files. The value of these attributes should be the name of a macro. For example, to call macro \startup on the prologue one has to use <exam prologue="startup">. Empty by default.
   - prologue="" If present executes its value before processing the content.
   - epilogue="" If present executes its value after processing the content.

4. randomseed="0"
   Seeding the random generator. Give it a value different from 0 (the default) as the start of the random series generator.

5. filter="all"
   When this attribute is not empty, <problem>'s having the same category are allowed to pass while others are rejected. The <problem> of the first example in this paper has a category="bachelor", which means that <exam filter="bachelor"> will select that one.

6. <flag="off">
   Here the attribute flag stands for one of the flag names defined below. These flags are provided in order to influence type and content of the exams produced. They assume the value on or off. For example, set random="on" to turn on the random ordering of multiple choice questions. For convenience true/false and yes/no act as synonyms for on/off. Flags are initialized to off. However, depending on the init attribute some are turned on by default.
   - init="exam" on:
     prompt, random, score
   - init="answer" on:
     prompt, random, answer, subscore
   - init="series" on:
     prompt, answer, subscore, info, history, note

   This is the complete list of flags:
   - answer

Show answers.
- break
  Generate linebreak after prompt.
- frame
  By default frame answer blocks.
- history
  Show history data.
- info
  Show information block as a whole.
- newpage
  Start each problem on a new page.
- note
  Show notes.
- series
  Show all problems in full.
- source
  Show problem source.
- prompt
  Precede answer space with a prompt.
- random
  Activate random generator and by default randomize multiple choice item order.
- score
  Show total score for problems; can be set to `left` for putting the score on the left or `right` for right placement (default).
- source
  Typeset the sourcecode of the problem.
- subscore
  Show (sub)score for all questions within problem; values are as for `score`.
- user
  Flag to be freely used.
- verbose
  Report activities in the logfile.

In order to provide as much flexibility as possible, the presence of flag attributes is checked and processed on entrance of the following nodes <exam> <file> <buffer> <problem> <question> and all the <...answer> nodes. Thus, if there is a reason to permanently inhibit random item order for a specific multiple choice question, one can do so by giving <problem random="off"> on its definition. To inhibite randomization in just this specific exam, put random="off" on the <file> node calling this problem.

A flag whose value was changed on a <file> or anywhere within, will revert back to the value that was set on the <exam> node before the next file is processed. A word of advice might be in order. Use flags sparingly below the <exam> level, because it is all too easy to create bewildering behaviour by indiscriminately sprinkling flag changes all over the place.

7. infostyle=""
   Font setting for the text in the information block, empty by default. A nonempty value triggers the execution of \switchtobodyfont[infostyle] at the start of the information block.

8. sourcestyle=""
   Font setting for source code. Source is typeset verbatim, the default therefore uses the current monotype font. A nonempty value triggers the execution of \switchtobodyfont[sourcestyle] at the start of typesetting the source.

9. Color settings.
   - errorcolor="black"
     Color of error messages.
   - messagecolor="errorcolor"
     Color of <message>, default is errorcolor.
   - infocolor="black"
     Color of items in information block.
   - goodcolor="black"
     Color of good answers.
   - wrongcolor="black"
     Color of wrong answers.
   - backgroundcolor="white"
     Color of background in framed blocks.
   - infobackgroundcolor="white"
     Color of background in information block.
   - colors="yes"
     Use set of colors built in, black and white otherwise.

<file attributes>

During the processing of a file or buffer the \currentsource macro gives access to its name. This node has the following set of attributes.

1. name=""
   This attribute gives the path to the XML file containing the problem code. The path must originate in \currentdirectory and lead to the location of the file. Note that the origin is dependent on the base attribute on the <exam> node. The alternative src is also accepted.

2. selection="1"
   A <select="on"> attribute on a <problem> node means the problem has alternative questions. For problems having alternatives this attribute provides a means to select a specific one. The attribute value 1...n determines which one will be chosen. That value must be a number greater than zero and less than or equal to the number of <question> nodes in the problem. By default the first of the alternatives is taken.

If this attribute is something other than a number, then the list of <question> nodes is searched for a matching selection attribute. If found, that question is chosen otherwise the first question is taken by default, An attribute of this type has precedence over a numeric one.

3. theselector="value"
   The attribute theselector denotes a selector defined inside the problem. See the description of the <content> node below for its usage.

<buffer attributes> Same as the <file> node but pertaining to a ConTEXt buffer. The name attribute or its alternative src gives the name of the buffer. Of course \currentdirectory has no meaning here.

<page option="yes"/> Generates a ConTEXt \page statement at this point. The default value is yes.

<vocabulary file="" buffer=""/> The terms appearing in the information block or for example in the answer prompt, are all localized through the value of ConTEXt's \currentlanguage macro. This is set by macro call \language[..]. In the program these terms are referred to in English and internally translated when an equivalent one is available in the current language. Currently built in are translations for English, Dutch and German. With <vocabulary> one can add other languages or supersede existing translations. The structure of a <vocabulary> can be seen in the following example. Here the standard term *file* is replaced by another Dutch (nl) equivalent: the word *bestand*. The English word must be present too, because all translations originate from that language.

```
<vocabulary>
  <word>
    <language name="en">file</language>
    <language name="nl">bestand</language>
  </word>
</vocabulary>
```

The vocabulary to add can be placed in a file or in a buffer whose name must be given as the appropriate attribute on the <vocabulary>. In case one needs to add or replace a few terms only, the <vocabulary> node can conveniently be placed in its entirety inside the <exam> node.

Counter values

Nodes that can come in handy, are those giving access to the counters for the number of problems and the scores.

<counter value="problem"> sequence number of current problem.

<counter value="problemscore"> score for the current problem.

<counter value="totalscore"> total value of scores sofar.

<message attributes>
The <message> node without attributes has its content typeset surrounded by <MESSAGE> and has the same color as error messages. This node can be used to draw attention to special situations as for example an unfinished part in the problem. Its appearance can be made conditional on a specific flag. A message ends the preceding paragraph.

1. color="messagecolor"
   The color used for the message.

2. text="MESSAGE"
   The text inside the brackets appearing before and after the message content.

3. flag=""
   If the name of an existing flag is given here, then that flag must be true in order to typeset the message; an unknown flag suppresses the message.

Nodes at the <problem> level

<problem attributes>
A <problem> node can contain a <subject> and <description> node and any number of <note>, <history>, <include> and <define> nodes.

1. category="all"
   Specifies a category for this problem on which it can be filtered.

2. select="off"
   Having select set to off signifies a problem consisting of several parts, all to be used. Setting this flag to on makes this a problem with alternative questions. Then the one selected on an examination is determined by the value of the selection attribute on the <file> node. To produce a catalogue of all alternatives choose the series option on <exam>.

<question score="0">
This node carries the score attribute. It specifies

the number of points to be earned by answering the question correctly. In the information block one will see the maximum score for the total of all independent questions contained in the problem. An absent score attribute defaults to zero.

A problem with a number of questions all to be answered, can be typeset as an itemized list. See the example below.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem select="off">
  <text>Itemized questions:</text>
  <ol columns="2">
    <li><question>Question-1</question></li>
    <li><question>Question-2</question></li>
    <li><question>Question-3</question></li>
    <li><question>Question-4</question></li>
  </ol>
</problem>
```

Problem-1 Itemized questions:

1. Question-1
2. Question-2
3. Question-3
4. Question-4

Answer nodes

`<shortanswer attributes>`

1. `fill="dots"`
   Type of line fill after the answer prompt. Other possibilities are `none` and `line`. After definition of macro `\myfill` the use of `fill="myfill"` is allowed.
2. `lines="1"`
   The number of filled lines typeset after the answer prompt. The default is one line. It is allowed to set the number of lines to 0.

`<listanswer attributes>`

1. `break="off"`
   Set this attribute to `on` in order to break the line after the descriptive text.
2. `sym="n"`
   Set the symbol used for the items as defined by ConTEXt's `\startitemize`. Numbers are the default.
3. `fill` and `lines`: See `<shortanswer>`

The attributes `fill` and `line` are valid also on the individual items of the list, covering the case where one needs to vary them individually.

`<altanswer break="off">`
Set the attribute `break` to `on` in order to break the line after the descriptive text.

`<blockanswer attributes>`

1. In case both the `answer` and `prompt` flags are off, the block is completely suppressed thus enabling questions to be answered elsewhere.
2. `frame="off"`
   Draws a frame around the block when set to `on`.
3. `height="0pt"`
   Set the height of the block. Setting `height` to a value greater than zero forces the answerblock to take that height. Otherwise the `lines` setting is honored.
4. `width="0pt"`
   Set the width of the block.
5. `alignblock="middle"`
   Force the position of the answer block to either `left`, `middle`, `right` or `none`.
6. `align="right"`
   Force the alignment of the content of the block. Values are one of the ConTEXt options `left`, `middle`, `right`.
7. `break="off"`
   Set this attribute to value `on` in order to insert a linebreak directly after the answer prompt, whenever the `prompt` option is selected. Useful when one needs the full textwidth, for example for a multicolumn list to follow.
8. `fill` and `lines`: See `<shortanswer>`

`<choiceanswer attributes>`

1. `distance=""`
   Offset between the item marker and the text.
2. `skip="none"`
   Extra blank space between items; either a dimension or one of small, medium, etc.

Other nodes

`<text attributes>`

The `<text>` node is meant for regular text and usually typesets at least one paragraph. By default a bare `<text>` node ends with a `\par`. This seems the most natural behaviour, because the objective of this node is the placement of a block of text. However, sometimes it might be better to suppress the trailing `\par`, most notably when the content is a small fragment of text. In that case, give the `par` attribute the value `no`.

1. `par="yes"`
   A `no` will suppress the `\par` at the end of the node.

2. `align=""`
   Presence of an attribute value `left`, `middle`, `right` will place the node content inside respectively a `\leftaligned`, `\midaligned`, `\rightaligned` macro.

3. `color="black"`
   Color the whole text node.

Be aware that in XML the ampersand & and similar reserved characters must be typed as `&` `<` `>` etc. TEX's nonbreaking space ~ must to be given as ` `

`<content attributes>`
This node enables one to select from alternatives. These alternatives are defined by the user with a (`selector`,`value`)-pair of freely chosen names. Within each group a member should be designated as the default for that group. On a `<file>` node the attribute pair `selector="value"` will select that specific member.

1. `selector="selection"`
   The `selector` attribute designates a group of alternatives. Through this designator one chooses a member from that group. Its value becomes an attribute on the `<file>` node having as its legal values those of its member `value` attributes.

2. `value=""`
   This `value` distinguishes this alternative from the other members of its `selector` group. On the `<file>` node a specific alternative is chosen by specifying its value as the attribute value of the group selector.

3. `default="off"`
   Setting the `default` attribute to `on` makes this one the default member within its group, to be chosen whenever there is no selection made on the `<file>` node. If none of the group members has been designated as the default, then none of them will be typeset.

Thus in the following example the first `<file>` will typeset the default and the second will typeset the content explicitely chosen.

```
<question>
  <content
    selector="mysel"
    value="first"
    default="yes">
    ..
  </content>
  <content
    selector="mysel"
    value="second">
    ..
  </content>
</question>
```

```
<file name=".."/>
<file name=".." mysel="second"/>
```

Inclusion and exclusion of code

`<include file=""/>`
Instead of repeatedly copying the same code into several problems, one can define these common parts separately (`includes.xml` in the example below) in a file. This file must contain a `<root>` node in which the definitions of the common code parts are put. The particular name of this node is not significant, it is merely a container for the `<define>` nodes inside. Each of the definitions contained should be given a name whereby they can be called up for insertion in the problem. This inclusion mechanism greatly enhances the maintainability of the problem database, because common code corrections and alterations can be concentrated in one place. Loading the file with definitions is done with the `<include>` node carrying the path of the file to read.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <define name="mypart">
    .. code to include ..
  </define>
</root>
<?xml version="1.0" encoding="UTF-8"?>
<exam>
  <include file="includes.xml"/>
  .. other nodes ..
</exam>
```

It is imperative that `<include>` nodes are placed directly inside the `<exam>` node, otherwise they are ignored. In case the use of a specific definition is local to a single problem, one has the option of putting the definition directly inside the `<problem>` itself.

`<define name="">`
A `<define>` node carrying content is a definition node. These nodes may be placed either in a separate file to be included in the `<exam>` node, in the `<exam>` node itself, or put directly inside the `<problem>` node. Definitions in the `<problem>` have priority above those in the `<exam>`. Those in `<include>` files come last, later inclusions having priority above those loaded earlier. The first match of a definition breaks off the search.

A `<define name="name"/>` node not carrying content implies retrieval of the definition. Thus the presence or absence of the node's content determines its role. In the following example code the definition `mypart` is put inside the `<problem>` and afterwards substituted for `<define name="mypart"/>`.

```
<problem>
  <define name="mypart">
    .. code of mypart ..
  </define>
  ..
  <text><define name="mypart"/></text>
</problem>
```

The <define> can have a type attribute when it is a definition node. In that case when called some special action depending on its value is taken. The preprogrammed actions are:

1. type="image" When the <define> resolves to the location of a file, this is put into the document with an \externalfigure call.

2. type="mpgraphic" The <define> must resolve to code for a graphic. For example a definition can be

   ```
   <define type="mpgraphic" name="example"
     parameters="color=black,variant=0">
   \startuseMPgraphic{example}{color,variant}
   if \MPvar{variant} = 0:
     draw (0,0) -- (10,10)
       withcolor \MPcolor{color};
   else:
     draw (0,0) .. (10,10);
   fi
   </define>
   ```

   then called with <define name="example" parameters="color=orange"/>. Note that the name on the <define> and MPgraphic definition must be the same. Other attributes on these nodes are height, width, scale and rotation, their usage speaks for itself.

For more details and parameter usage see the paper on the hvdm-xml module.

Beware-1. The inclusion of <question> nodes inside a <define> is explicitely forbidden and raises an error, because it interferes with the intended processing of questions within <problem>. The radical solution chosen here is to delete the offending <define> from the nodetree. As a consequence its usage later in the input will produce another error message, enabling one to pinpoint the cause.

Beware-2. The inclusion of a <define> is not the same as macro substitution in a programming language as for instance C. As a result its replacement is not always bringing what is expected and its usage is somewhat limited. However, for low level replacements in <text> and <value> nodes or inclusion as list items it works reasonably well. Complicated formulas in MathML are good candidates too. On higher level nodes its usage turned out somewhat erratic and is therefore not supported everywhere. Feel free to experiment and don't be too disappointed if it doesn't work as you hoped for. After all, it is nothing more than a convenient extra.

Beware-3. The inclusion through XML can have consequences for special characters. Be especially on guard for trouble with &, a character that may need replacement by & more proper for XML data.

<hide flag="answer">

The content of the <hide> node will appear depending on the value of its flag attribute, by default the answer flag. For value on of the targeted flag the content of the <hide> node will be suppressed. Otherwise the content will appear. This node is especially useful in case one wants to put something inside a <blockanswer> having its <force> flag set. Then for example <hide flag="prompt"> allows suppression of content based on the value of the prompt flag.

<show flag="answer">

The alter ego of <hide>. When the flag targeted by <show> is on then the content of the <show> will be shown, otherwise it will be suppressed. Using a <show> and <hide> pair one can have alternating content depending on the attribute value of the flag. It is therefore possible to have one text for answer=off and a different one for answer=on.

<random save="" reuse="">

Chooses at random one of the nodes directly under it. Does nothing if there aren't nodes inside and always takes the first node if random selection is off. Intervening <!-- comment -->'s are ignored.

Attribute save="yes" enables the user to reuse the last value acquired by <random>. That last value is internally stored and can be reused (nondestructively) with <random reuse="yes">.

For more elaborate constructions it is possible to tie the random value to a chosen identification. For example, the value of the random generator produced with the call <random save="mysave"> is reused with <random reuse="mysave">. Mistyping the identification to a nonexistent save generates an error, as is to be expected. Furthermore the strings yes, no, on, off, true, false cannot be used as identifier.

I wish to thank Martin Althoff for stimulating discussions contributing greatly to the rapid finalization of the main features of both the module code and this article.

Hans van der Meer
H.vanderMeer@uva.nl