IMS Question and Test Interoperability Migration Guide

Version 2.0 Final Specification

Document Name: IMS Question and Test Interoperability Migration Guide
Revision: 24 January 2005

Date Issued:	24 January 2005
Latest version:	http://www.imsglobal.org/question/qti_v2p0/imsqti_migrv2p0.html
Supersedes:	QTI Item v2.0 Public Draft specification, 07 June 2004, http://www.imsglobal.org/question/
Register comments or implementations:	http://www.imsglobal.org/developers/ims/imsforum/categories.cfm?catid=23

Table of Contents

1. Introduction

This document provides advice and guidelines on converting data conforming to the version 1 ASI data model into version 2. For readers who are already familiar with version 1 it can also be used as a quick introduction to the new concepts in version 2.

2. References

CSS
Cascading Style Sheets, level 2
http://www.w3.org/TR/CSS21/
IMS_CP
IMS Content Packaging Specification, Version 1.3.1

3. Assessments and Sections

Version 2 of this specification concentrates on providing an updated information model and associated bindings for individual items, their meta-data, usage-data and packaging, however, it does not update the *assessment*, *section* or *objectbank* concepts defined in version 1.

At the time of writing, no decision has been made by IMS as to whether or not to offer an updated model for an *assessment*. Tools that aggregate items using this structure are not believed to be widely implemented. There are three possible migration paths for such content.

1. Maintain it using version 1. This migration guide should still be used to check that the individual items could be converted into version 2 if required and that features that are now deprecated are not being relied upon. If the content relies heavily on extensions to implement features that are now possible directly then the custom markup could be migrated to more closely match version 2.
2. Migrate the individual items to version 2 and create a custom markup format for specifying how those items are aggregated to make an assessment in your system. Such a format could also be shared with other developers to help create a consensus on the appropriate development path for version 2 of the assessment object.
3. Migrate the individual items to version 2 and use a hybrid approach in which the assessment and section structures from version 1 are used along with *itemref* to refer to the (external) version 2 items.

4. Mapping Response/Render Types to Interactions

In QTI version 1, the interactions between the candidate and the delivery engine were described using a two-stage model. Firstly, the type of data required from the candidate was defined using one of the *response_lid*, *response_xy*, *response_str*, *response_num* or *response_grp* elements. The style or *rendering* of the item was then determined by

choosing one of the *render_choice*, *render_hotspot*, *render_slider* or *render_fib* elements.

In version 2, this model has been updated to specify more types of interaction. The data type is still defined separately (using a responseDeclaration) but this appears outside the presentation (now called the itemBody) so that it can be used more easily by systems that don't parse the item's content. The render types have been replaced by an expanded set of more tightly defined interactions that clearly state which data types they can and can't support. To help migration of version 1 content, the following table can be used to find the version 2 interaction that corresponds to each version 1 response/render combination.

response_/render_	choice	hotspot	slider	fib
lid	For single or multiple cardinality use choiceInteraction, for ordered cardinality use orderInteraction.	For single or multiple cardinality use hotspotInteraction, for ordered cardinality use graphicOrderInteraction. See Migrating render_hotspot below for more information.	For single cardinality use choiceInteraction, slider behavior must be communicated through an accompanying stylesheet and is now outside the scope of this specification. Multiple and ordered cardinality are undefined.	Undefined
xy	Undefined	For single or multiple cardinality use selectPointInteraction, ordered cardinality is no longer supported. See Migrating render_hotspot below for more information.	Undefined	Undefined
str	Undefined	Undefined	Undefined	Use extendedTextInteraction or textEntryInteraction. See Migrating render_fib below for more information.
num	Undefined	Undefined	For single cardinality use sliderInteraction. Multiple and ordered cardinality are undefined.	Use extendedTextInteraction or textEntryInteraction. See Migrating render_fib below for more information.
grp	For single or multiple cardinality use associateInteraction, for multiple cardinality: undefined.	Undefined	Undefined	Undefined

4.1. Migrating render_fib

In version 1, *render_fib* was used to describe all types of free text entry interaction. In version 2, this type of interaction is separated into two types: extendedTextInteraction and textEntryInteraction. The difference between them is the role that the interaction takes in the content model. A textEntryInteraction behaves like a simple run (span) of text and can be used for requesting the candidate to type a response that will appear in the context of the surrounding text, for example, completing missing words from a given sentence. In contrast, an extendedTextInteraction behaves more like a whole paragraph of text (with an optional prompt) and would typically be used for obtaining longer responses ranging from 'short' answers to complete essays.

When migrating content from version 1, choosing between these two representations requires the application of some heuristics as the intention was not specified. A reasonable rule is to assume that *render_fib* elements that contain a mixture of material and *response_label(s)* should be translated into a series of *separate* textEntryInteractions, one for each *response_label*. In version 2, this mapping requires a new response variable to be declared for each textEntryInteraction. Otherwise, an extendedTextInteraction can be implied.

4.2. Migrating render_hotspot

In version 1, *render_hotspot* was used to describe interaction types that made use of areas defined relative to the coordinate system of a screen area containing an image. This coordinate-based relationship between the interaction and the material meant that hotspot questions could be created that interacted with several images simultaneously. Indeed, the diagram given in the version 1 information model encouraged this use. In addition, the location of the material objects that defined the images within the content-model was not constrained.

This definition of the coordinate system for hotspot interactions was impractical unless coordinates were also given for all of the material in the item (see Specifying Coordinates below for a related discussion). As a result, interoperability of hotspot questions was hampered and clarification requested.

In version 2, this issue has been resolved with a more restrictive model for the graphical interactions. As a result, when migrating content from version 1 only hotspot questions that made use of a single image can be converted directly. Items which located the *material* object containing the associated *matimage* as a child element of *render_hotspot* should be easier to migrate. Similarly, items that ignored coordinate positioning of material and used image-relative coordinates to describe hotspot areas instead of notional screen coordinates can be mapped to the new information model directly.

5. Specifying Coordinates

Version 1 of QTI defined the attributes *x0*, *y0*, *width* and *height* for specifying the exact position of images or other material relative to the screen. Version 2 inherits its material model from XHTML and therefore does not have concepts that correspond to *x0* and *y0* and specifying the dimensions of objects in the content model is limited to img and object.

Removing detailed information about the presentation (layout/styling) of the content from the QTI information model was a key requirement in version 2 to help support the exchange of items that can work on as wide a variety of systems as possible thereby improving accessibility. Specifying detailed information about layout is also a more general problem than the one QTI is trying is to solve and so it makes sense to look for a more general solution in a similar way to QTI's adoption of elements from XHTML to solve the problem of describing the structure of the material.

The general solution to providing presentation-orientated information for documents marked up in XML is to provide a stylesheet and this approach is encouraged in QTI through the stylesheet class. This specification does not mandate the use of one stylesheet language over another, however, because [CSS] is fairly well established as a language for controlling the presentation of HTML documents it is used in this document to illustrate how style can be applied to QTI items.

When migrating content from QTI version 1 that specifies coordinates it is recommended that the corresponding element in version 2 is given an id and that this id is used to associate *fixed* positioning rules in the associated

stylesheet. For example, the element from a version 1 item below:

```
<material>
    <matimage imagtype="image/jpeg" x0="300" y0="500" width="200" height="200" uri="image.jpg"/>
    <altmaterial>
        <mattext>A Pretty Picture</mattext>
    </altmaterial>
</material>
```

can be represented in version 2 as:

```
<img id="image01" src="image.jpg" width="200" height="200" alt="A Pretty Picture"/>
```

The following CSS rule can then be used to represent the missing coordinates:

```
#image01 { position: fixed; top: 500px; left: 300px }
```

Version 1 items that specify coordinates only for some of the material objects are harder to migrate because the relationship of the objects with fixed positions to those without is not described in the specification - custom rules will need to be applied based on additional knowledge of the original target delivery engine.

6. Reference

6.1. <and>

No change

6.2. <conditionvar>

In version 2 this element is no longer needed. Expressions are direct descendants of the responseIf or responseElseIf objects. If a conditionvar contained multiple test operators then it was an implicit *and*, in version 2, it must be explicitly replaced by an and.

6.3. <decvar>

In version 2, decvar has been replaced by outcomeDeclaration which appears in the information model as a direct descendant of assessmentItem and not as part of responseProcessing.

6.3.1. Attribute: varname

Outcome variable names are identifiers in version 2 and share a namespace with response variables *and* the identifiers given to choices (version 1 response_labels). See Attribute: identifier below for more information.

6.3.2. Attribute: vartype

Variables in version 2 are declared with a baseType. The version 1 types *integer*, *string* and *boolean* are the same. The two numeric types *Decimal* and *Scientific* both become simply float and the *Enumerated* type becomes identifier. An additional value, *set*, was referred to in the information model but was never defined.

6.3.3. Attribute: defaultval

This is now represented by defaultValue, an optional part of each variableDeclaration.

6.3.4. Attribute: minvalue, maxvalue

In early versions of the specification when (if at all) these constraints were applied to outcome variables caused some confusion. In version 2, it is not possible to specify constraints on variables when they are declared. In order to implement the version 1 behavior it is necessary to add additional responseConditions that check the value and set it appropriately at the end of the responseProcessing section.

The most common use for this type of bounds checking is to score multiple-response items where each individual response value has been assigned a score and these scores are summed to create a total. This allows incorrect responses to have a negative effect and correct ones to have a positive effect. This type of response processing is now enabled using a more direct method: the declaration of a mapping. The mapping forms part of the outcomeDeclaration and *does* have these constraints, through the two attributes lowerBound and upperBound.

6.3.5. Attribute: cutvalue

No longer supported in version 2. If an assessmentItem has a cutvalue it is recommended that a second outcome variable is declared as a *boolean*, for example "MASTERY", and set during response processing explicitly.

6.3.6. Attribute: members

This attribute supplied a set of identifiers to accompany declarations with vartype="Enumerated". In version 2, this vartype becomes identifier and it is not possible to restrict the outcome values in this way.

6.4. <displayfeedback>

In version 2, all feedback is controlled by the built-in outcome variable *feedback*. This variable is of multiple cardinality and is initially empty. The equivalent operation in version 2 is therefore to add an identifier which controls the desired feedback to this outcome variable. This is done using the multiple operator to combine the *current* value of the feedback outcome variable with the desired identifier.

6.4.1. Attribute: feedbacktype

This attribute was poorly understood in version 1 and is no longer supported.

6.4.2. Attribute: linkrefid

In version 1, this attribute was used to identify the feedback to be displayed. In version 2, this identifier should be added to the built-in feedback outcome variable as described above.

6.5. <duration>

Although an attribute of item in version 1 it has been placed outside the scope of assessmentItem in version 2.

6.6. <flow>

The flow element has been replaced along with the entire material model. In version 1, the flow element allowed the contents of the presentation to be grouped into sub-parts *and* structured hierarchically. In version 2, the contents of the itemBody are grouped by the familiar structural markup of HTML. In HTML, hierarchical structure is not strongly represented. Although multiple levels of headings are defined it is not easy to determine at which level within the document each paragraph is supposed to be. To solve this problem, the HTML *div* element is usually used. div is supported directly in QTI in the same way, however, when converting version 1 to version 2 some flow elements will correspond to *div* and some to other structural elements, such as *p*.

6.6.1. Attribute: class

The class attribute on flow was designed to support stylesheet rules. In version 2, all the components of the itemBody can take a class attribute for the same purpose.

6.7. <flow_label>

The flow_label element is a constrained form of flow that is used within the render family of objects. Not well supported, its meaning was often not clear, though it was used to *arrange* the choices in a simple multi-choice question. The new interactions do not allow this sort of control over the choices and so there is no support for it in version 2.

6.8. <flow_mat>

The flow_mat element is a constrained form of flow that is used in contexts where only non-interactive material is allowed. See <flow> for more details.

6.9. <interpretvar>

Replaced by the interpretation attribute on outcomeDeclaration. In version 1, any material could be used whereas in version 2 it is limited to plain text.

6.9.1. Attribute: view

Interpretations can no longer be restricted by view in version 2.

6.10. <item>

In version 2 the *item* element has been renamed *assessmentItem* to avoid confusion with the similarly named *item* element in IMS Content Packaging [IMS_CP]

6.10.1. Attribute: maxattempts

No longer supported directly in version 2. Control over the maximum number of attempts allowed at an item is no longer controlled by the author. Instead, it is controlled when the item is used. See the notes on Assessments and Sections for further details.

6.10.2.

The Integration Guide sets out a mechanism for packaging assessment items using the IMS Content Packaging specification [IMS_CP]. There are now potentially three places where items can be identified.

1. The identifier attribute of the assessmentItem itself, corresponds to the old ident attribute.
2. The identifier, as specified in the associated meta-data for the item. In fact, multiple identifiers are supported by the meta-data but one of them must have an entry that corresponds to the value of the assessmentItem's identifier attribute.
3. The identifier of the associated resource in a content package. This could be the same as the assessmentItem's identifier, however, resource identifiers are subject to different restrictions so some translation may be necessary.

6.10.3. Attribute: title

In version 2, the length limit of 256 characters for item titles has been lifted.

6.10.4. Attributes: label, xml:lang

No change.

6.11. <itemcontrol>

Outside the scope of assessmentItem in version 2.

6.12. <itemfeedback>

This element has been replaced with modalFeedback.

6.12.1. Attribute: view

In version 2, modalFeedback cannot be made view dependent.

6.12.2. Attribute: ident

In version 1, each piece of feedback was given an identifier and the displayfeedback element was used to control whether or not the feedback was to be shown. In version 2, this mechanism has changed. Feedback is now controlled by a built-in outcome variable *feedback* which is a container of identifiers set in the same way as other outcome variables (using setOutcomeValue). The visibility of a particular piece of (modal) feedback is then controlled by an identifier and an associated showHide attribute which determines if the feedback is normally hidden and shown when its identifier is in the feedback outcome variable (the default) or vice-versa.

Although the effect is roughly the same, the new mechanism is more powerful as it allows integrated feedback to be controlled in the same way *and* it also enables several pieces of feedback to be activated by the *same* identifier.

6.12.3. Attribute: title

No change.

6.13. <mataudio>

Replaced by object.

6.14. <matbreak>

Replaced by br

6.15. <material>

In version 1, the material element provided a general purpose wrapper for runs of text, images and other multimedia objects. In version 2, this wrapping is largely unnecessary as these objects are now placed into the information model directly.

6.15.1. Attributes: label, xml:lang

If attributes of material have been used to assign properties to a specific group of material objects then these objects can be grouped in either a span or div as appropriate.

The material element could appear in many contexts in the version 1 model that are now more heavily constrained. The above mentioned mapping works for material contained directly in a presentation or flow but in other contexts use of div and/or span may not be allowed and no mapping will be possible, for example, when labeled material is used within <interpretvar>.

6.16. <matimage>

Replaced by img or object. This duplication of function is inherited from XHTML. When converting images used as part of interactions the object form is required.

6.16.1. Attribute: imagtype

Note that img, though the more usual element in HTML, does not have a corresponding type attribute so the object form might be preferred if this information is needed.

6.16.2. Attributes: label, height, width

Supported on both img and object.

6.16.3. Attributes: uri

Equivalent to the src attribute of img or the data attribute of object.

6.16.4. Attribute: embedded

In version 1, image data could be embedded directly into the QTI XML file using the embedded attribute to indicate the encoding used. This is no longer supported.

6.16.5. Attributes: x0, y0

No longer supported directly in QTI, see Specifying Coordinates for details.

6.16.6. Attribute: entityref

No longer supported in version 2.

6.17. <mattext>

Simple runs of text are now represented more directly and so there is no longer a need for the mattext element, the notional class textRun, which is simply bound to PCDATA in XML, will usually takes its place.

6.17.1. Attribute: texttype

mattext was also used to include text marked up in other languages, such as HTML or RTF. Material marked up in HTML should translate without difficulty into version 2 as most of the familiar HTML elements are represented directly in the information model. Some elements that define aspects of style and layout, such as the HTML *font* element are not supported though - this information must now be placed in an associated style sheet.

6.17.2. Attributes: label, xml:lang

If attributes of mattext have been used to assign properties to a specific run of text (or complex content) then either span or div will have to be used instead, as appropriate. See also the note under Attributes: label, xml:lang on material.

6.17.3. Attribute: xml:space

The xml:space attribute is no longer used, if space-preserved text is to be included in items then the pre element is

required.

6.17.4. Attribute: charset

No longer supported in version 2.

6.17.5. Attributes: x0, y0, width, height

No longer supported directly in QTI, see Specifying Coordinates for details.

6.17.6. Attributes: uri, entityref

Version 1 used mattext to incorporate externally defined material through a URI or external entity declaration and corresponding reference. The new content model, being based on XHTML, does not have a mechanism that corresponds to this. It was never completely clear what role this type of material was supposed to play. In version 2 it should either be included directly (marked up using the version 2 tags) or treated as an object.

6.18. <not>

No change.

6.19. <objectives>

In version 2, item objectives are treated as part of the item's meta-data. In version 1, any material could be used whereas in version 2 they are limited to plain text. Also, in version 1, it was not clear whether objectives should have been displayed to the candidate when the item was presented or if they were just informational. As meta-data they are now *only* informational.

6.19.1. <view>

Objectives with a restricted view are not supported in version 2, if the restriction is important then it is recommended that a rubricBlock is used instead. In fact, use of this attribute is suggestive of objectives that should be displayed along with the item's presentation.

6.20. <or>

No change.

6.21. <other>

The other operator was used in version 1 to match conditions that were "otherwise undefined". In practice, what this meant was that it was used as if it were a test that always returned true in the last respcondition in a response processing section. This adhered to the spirit of the definition when each respcondition had the default value for the continue attribute ("No") as the last respcondition would effectively catch situations in which all the previous tests had failed or had been "otherwise" untested for.

In version 2, responseCondition (the replacement for respcondition) now uses an if/else model which does not require the use of always-true test operators. Literal conversion from version 1 can use baseValue with a baseType of boolean and the value "true" in place of other if required.

6.22. <outcomes>

The outcomes element served an organizational function in version 1, grouping together the outcome declarations at

the start of the response processing section. In version 2, outcomes are declared using outcomeDeclarations and appear as direct descendants of the assessmentItem eliminating the need for an equivalent definition.

6.23. <presentation>

This has been replaced by the itemBody in version 2, though the role is roughly equivalent. The term presentation is sometimes treated as synonymous with *layout* and was therefore changed to reduce the confusion concerning its role in QTI. QTI does not define information for laying out the contents of the itemBody, instead, it supports the class and id attributes on the itemBody, and all its component parts, to enable layout information to be specified by an associated stylesheet.

6.23.1. Attributes: x0, y0, width, height

No longer supported directly in QTI, see Specifying Coordinates for details.

6.23.2. Attributes: label, xml:lang

No change.

6.24. <questestinterop>

This element acts as a container for all version 1 QTI ASI objects. This role is now played by a content package. For more information, see the Integration Guide.

6.25. <render_choice>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.25.1. Attribute: shuffle

Now becomes an attribute of the individual interactions, for example, choiceInteraction.

6.25.2. Attribute: minnumber

No longer supported in version 2. It is no longer possible to insist on the minimum number of choices that a user must select.

6.25.3. Attribute: maxnumber

Now becomes an attribute of the individual interactions, renamed as appropriate. For example, the maxChoices attribute of choiceInteraction

6.26. <render_fib>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.26.1. Attributes: encoding, charset

No longer supported in version 2.

6.26.2. Attribute: fibtype

This attribute was largely redundant in version 1 as response_num had a similar Attribute: numtype attribute. In version 2, the response type is determined by the responseDeclaration only.

6.26.3. Attributes: rows, columns, maxchars

Replaced with a single attributed, expectedLength. Note that this is now guidance to the delivery system for the proper sizing of input boxes (where applicable), it is not possible to restrict the number of characters entered by the candidate.

6.26.4. Attribute: prompt

No longer supported in version 2. If control is required over the style of the input box used to enter a response the class attribute should be used to identify a description using an externally defined style-sheet.

6.26.5. Attribute: minnumber, maxnumber

These attribute were problematic in version 1. Support for requiring a minimum number of responses is no longer available in version 2. Maxnumber is replaced by the maxStrings attribute.

6.27. <render_hotspot>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.27.1. Attribute: minnumber

No longer supported in version 2. It is no longer possible to insist on the minimum number of points/hotspots that a user must select.

6.27.2. Attribute: maxnumber

Now becomes an attribute of the individual interactions, renamed as appropriate. For example, the maxChoices attribute of hotspotInteraction

6.27.3. Attribute: showdraw

In the version 1 examples, this attribute heralded a "connect the points" question, though these questions were always bound to a point type response variable with multiple cardinality so there was no information about exactly how the points had been connected, only which points had been connected. In version 2, this type of interaction is called a graphicAssociateInteraction and must be bound to a response with type pair. The showdraw attribute itself therefore has no direct equivalent in version 2.

6.28. <render_slider>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.28.1. Attribute: orientation

Equivalent to the orientation attribute of sliderInteraction.

6.28.2. Attributes: lowerbound, upperbound

Equivalent to the lowerBound and upperBound attributes of sliderInteraction.

6.28.3. Attribute: startval

The starting value for a slider is now taken from the defaultValue in the associated responseDeclaration.

6.28.4. Attributes: step, steplabel

Equivalent to step and stepLabel.

6.28.5. Attribute: minnumber, maxnumber

These attributes were meaningless when applied to sliders in version 1 and are no longer supported.

6.29. <respcondition>

This element has been replaced with responseCondition which has additional functionality.

6.29.1. Attribute: continue

One of the most significant changes in version 2 is the removal of the continue attribute in favor of an if, else-if, else model within responseCondition. Continue was clearly documented but it was not well implemented, partly because some implementers found the default value unintuitive. If all respcondition elements had continue set to No (the default) then they can all be replaced with a single responseCondition containing a corresponding sequence of responseIf, responseElseIf, responseElseIf, etc. If all respcondition elements had continue set to Yes then the transformation is even simpler, each respcondition corresponding to a individual responseCondition containing a single responseIf. Response processing sections that contain mixtures of Yes and No on their continue attributes are a little more complicated to translate. A No followed by a Yes requires the addition of a responseElse to the responseCondition and the commencement of a nested set of rules. A Yes followed by a No is mercifully simpler as the strategy for the uniform Yes case can be switched to the strategy for the uniform No case at that point.

6.29.2. Attribute: title

No longer supported in version 2.

6.30. <response_group>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.30.1. Attributes: ident, cardinality, rtiming

See <response_lid> below for details.

6.31. <response_label>

In version 1, *response_label* was the powerhouse of QTI. It satisfied a different role depending on the combination of response/render elements that contained it. It therefore had many attributes, most of which were ignored in any given situation. In version 2, it has been replaced with an abstract class, choice from which a number of special purpose elements are derived.

In version 1.2.1 of the specification, the role of hotspot label was given to material contained within a response_label being used to define a hotspot. In version 2, this material must be a simple run of text (max 256 characters) and is specified using the hotspotLabel attribute.

6.31.1. Attribute: rshuffle

The sense of this attribute has been inverted and the name has changed in an attempt to reduce confusion with the shuffle attributes on the individual interactions. In version 2, rshuffle is replaced with the fixed attribute.

6.31.2. Attribute: rarea

This attribute has been replaced with the shape attribute of hotspot. Note that the value *Ellipse* is now deprecated. For compatibility, it has not been removed completely; converting an ellipse into a polygon through rasterization is a non-trivial operation. Circular ellipses should be converted to circles instead.

The coordinates that defined the area were previously supplied as PCDATA within the response_label itself. These are now specified through the coords attribute. Note that coordinates are now space-separated, instead of being comma separated, and that for rectangles they are now given as "xleft yleft xright yright" and for circles (and ellipses) the dimensions are give as radii *not* diameters. These changes are designed to bring area definitions inline with those used in HTML.

6.31.3. Attribute: rrange

rrange was not consistently documented in version 1 and is no longer supported in version 2. Determining whether or not a value entered by the candidate is acceptable is a job for response processing and the operators defined there should be used instead.

6.31.4. Attribute: labelrefid

This attribute was deprecated in version 1.2 and is no longer supported.

6.31.5. Attribute: ident

In version 2, the identifiers given to choices occupy the same namespace as the identifiers given to the variables themselves and must be unique within each assessmentItem. Therefore, care is needed when converting from version 1. See also the note Attribute: identifier below.

6.31.6. Attribute: match_group, match_max

In version 1, match_group was a comma-separated list of identifiers. In version 2 it follows the XML Schema convention of being a space-separated list of identifiers. Care will be needed when dealing with space padding around the commas, for ease of conversion it is probably safe to assume that the identifiers do not contain spaces. In version 2, they *must not* contain spaces.

In version 2, matchGroup is an attribute of associableChoice and matchMax is an attribute of those classes derived from it for which it makes sense, for example, simpleAssociableChoice.

6.32. <response_lid>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.32.1. Attribute: rcardinality

The cardinality of the response is now indicated by the cardinality attribute of the associated responseDeclaration.

6.32.2. Attribute: rtiming

In version 2, the interaction model for an assessmentItem does not support the concept of individual timing for each interaction. Therefore, the only duration that can be recorded is the amount of time taken for the whole item. For simple items, with only a single interaction, this distinction is not important.

6.32.3. Attribute: identifier

Responses are identified by a unique (within the scope of an assessmentItem) identifier in version 2. This identifier is declared in the responseDeclaration and used to bind an interaction to it. It is also referred to during responseProcessing in access the value of the response in a similar way to version 1.

In version 1, no guidance was given as to what constituted a legal identifier, though a length restriction of 32 characters was specified in the information model. In version 2, identifiers must satisfy the XML NMTOKEN production *and* must not contain the colon character. Use of the period is deprecated to enable its use as an access character in the future.

The use of standard response processors, new in version 2, also encourages some uniformity in the naming of response variables. For simple items (with only one response) the identifier of the response should be "RESPONSE".

Version 1 was silent on the case sensitivity of identifiers. Version 2 assumes *case-sensitivity*. It is recommended that identifiers from version 1 be lower-cased when converting to version 2 unless they already match one of the standard variable names.

6.33. <response_num>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.33.1. Attribute: numtype

In version 2, *decimal* and *scientific* are both treated as being of base-type float. See also the comment concerning Attribute: fibtype above.

6.33.2. Attributes: ident, cardinality, rtiming

See <response_lid> below for details.

6.34. <response_str>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.34.1. Attributes: ident, cardinality, rtiming

See <response_lid> below for details.

6.35. <response_str>

The response/render model has been replaced in version 2 by responseDeclarations and interactions. For detailed information, see Mapping Response/Render Types to Interactions.

6.35.1. Attributes: ident, cardinality, rtiming

See <response_lid> below for details.

6.36. <resprocessing>

This element is now called responseProcessing. Version 2 allows richer processing rules through the use of the new QTI expressions and conditional structures. In version 1, multiple resprocessing sections could be specified, in version 2 only one responseProcessing section can be specified.

6.36.1. Attribute: scoremodel

Originally introduced as a way to differentiate multiple resprocessing sections this attribute was deprecated in version 1.2 and is not needed in version 2.

6.37. <rubric>

In version 1, whether or not the item's rubric should be displayed when the item was presented required clarifying. In version 2, rubric has been replaced by rubricBlock which is integrated with the itemBody to make it clearer that it must be.

6.37.1. Attribute: view

The view attribute is unchanged, however, in version 2 there are a smaller number of permitted values: *author, candidate, proctor, scorer, tutor*. The version 1 values *administrator, adminauthority* and *invigilator* are treated in version 2 as *proctor*. Similarly, *assessor* and *psychometrician* are treated in version 2 as *scorer*. Although in many scenarios it may be possible to distinguish these roles more clearly they do not represent unique roles when controlling access to restricted parts of the itemBody.

6.38. <setvar>

The setvar element has been replaced with the setOutcomeValue response rule.

6.38.1. Attribute: varname

The name of the variable is now identified by the identifier attribute of setOutcomeValue.

6.38.2.

In version 2, setOutcomeValue does not support integrated arithmetic. Instead, it sets the outcome variable to the value of an expression. The *Add, Subtract, Multiply* and *Divide* actions are therefore replaced with the sum, subtract, product and divide operators respectively. The version 1 behavior can be obtained by using these operators to combine the variable operator (to obtain the *current* value of the outcome variable) with an appropriate baseValue.

6.39. <solution>

No longer supported in version 2. Material contained in a solution element (which itself was within <itemfeedback>) is simply treated as part of modalFeedback.

6.39.1. <Attribute: feedbackstyle

The style used to present solution material within feedback can be controlled using div and a suitable value for the class attribute, though a vocabulary is beyond the scope of this specification.

6.40. <solution_material>

This was a simple grouping element in version 1 and is no longer needed in version 2. See <solution> above for details.

6.41. <varequal>

In version 2, the elements used for testing the values of variables have been replaced by a new, more general, expression model. varequal is replaced by a comparison operator and two sub-expressions: a variable operator for accessing the value of the variable and a baseValue operator for specifying the value it should be compared with.

When testing identifiers and integers the match comparison operator is appropriate - it returns true only for exact matches between base values. When testing strings, stringMatch may be more appropriate as it supports case-insensitive matching. For real number comparisons, the equal operator is required.

When the variable being tested has multiple cardinality or ordered cardinality (with no index specified) varequal returns *true* if the container contains at least one matching value. This functionality is now achieved using the member operator. Note that this operator uses the match form of comparison only.

6.41.1. Attribute: case

When comparing strings, the case-sensitivity of the comparison can be controlled with the caseSensitive attribute.

6.41.2. Attribute: respident

To test the value of a specified variable in version 2, use the variable operator, the respident corresponds to its identifier attribute.

6.41.3. Attribute: index

When testing a response variable of ordered cardinality, the optional index attribute allowed varequal to single out a specific response for the comparison. In version 2, this is achieved with the index operator.

6.42. <vargt> and <vargte>

In version 2, the elements used for testing the values of variables have been replaced by a new, more general, expression model. vargt(e) is replaced by a comparison operator and two sub-expressions: a variable operator for accessing the value of the variable and a baseValue operator for specifying the value it should be compared with.

6.42.1. Attributes: respident, index

See <varequal> for details.

6.43. <varinside>

In version 1, varinside was used to test a point against an area. This was done during response processing in order to ensure that the hotspots were hidden from the candidate's view during the interaction. In most simple cases the use of varinside can be replaced by declaring an areaMapping in the response declaration. The area mapping allows the (hidden) hot areas of an image to be scored using a standard response processing template. Determining if this scoring method is appropriate, however, is largely a matter of judgment and cannot (easily) be determined automatically. Therefore, varinside does have a directly corresponding operator in version 2: inside.

6.43.1. Attribute: areatype

See Attribute: rarea above for information about the transformation required between the version 1 area model and the version 2 shape model.

6.43.2. Attributes: respident, index

See <varequal> for details.

6.44. <varlt> and <varlte>

In version 2, the elements used for testing the values of variables have been replaced by a new, more general, expression model. varlt(e) is replaced by a comparison operator and two sub-expressions: a variable operator for accessing the value of the variable and a baseValue operator for specifying the value it should be compared with.

6.44.1. Attributes: respident, index

See <varequal> for details.

6.45. <varsubset>

The role of varsubset was clarified in version 1.2.1 and the examples updated to demonstrate its use in comparing associated pairs in items that used response_grp. Unfortunately, there were no render types that could be unambiguously used in association with response_grp so the use of varsubset was limited. In addition, the semantics post-clarification were identical to varequal excepting the change from simple type to the *grp*.

In version 2, associations between choices are represented by pair or directedPair and values can be matched using the match operator in exactly the same way as the simple types. (See <varequal> above for more information.)

It is possible that varsubset may have been interpreted as a more general purpose operator for testing that a given set of values is a subset of the response provided by the candidate. In version 2, this is represented by the contains operator.

6.45.1. Attribute: setmatch

The setmatch attribute was not illustrated in the examples distributed with version 1 and may have been used to differentiate between the two cases described above. In version 2, the match operator works just as well for comparing containers (i.e., variables with multiple or ordered cardinality) as it does for simple types. Therefore, there is no need to use an attribute (of contains) to distinguish between exact and partial comparison of containers.

About This Document

Title	IMS Question and Test Interoperability Migration Guide
Editor	Steve Lay (University of Cambridge)
Version	2.0
Version Date	24 January 2005
Status	**Final Specification**
Summary	This document describes the QTI Migration Guide specification.

Revision Information	24 January 2005
Purpose	This document has been approved by the IMS Technical Board and is made available for adoption.
Document Location	http://www.imsglobal.org/question/qti_v2p0/imsqti_migrv2p0.html

To register any comments or questions about this specification please visit:
http://www.imsglobal.org/developers/ims/imsforum/categories.cfm?catid=23

List of Contributors

The following individuals contributed to the development of this document:

Name	Organization	Name	Organization
Niall Barr	CETIS	Joshua Marks	McGraw-Hill
Sam Easterby-Smith	Canvas Learning	David Poor	McGraw-Hill
Jeanne Ferrante	ETS	Greg Quirus	ETS
Pierre Gorissen	SURF	Niall Sclater	CETIS
Regina Hoag	ETS	Colin Smythe	IMS
Christian Kaefer	McGraw-Hill	GT Springer	Texas Instruments
John Kleeman	Question Mark	Colin Tattersall	OUNL
Steve Lay	UCLES	Rowin Young	CETIS
Jez Lord	Canvas Learning		

Revision History

Version No.	Release Date	Comments
Base Document 2.0	09 March 2004	The first version of the QTI Item v2.0 specification.
Public Draft 2.0	07 June 2004	The Public Draft version 2.0 of the QTI Item Specification.
Final 2.0	24 January 2005	The Final version 2.0 of the QTI specification.
