

Dutch Exam Profile v4.0

QTI Adaptations Reference

0. Table of Contents

1. Introduction	2
1.1. DEP Documentation Overview	2
1.2. Legend	2
1.3. Copyright Notice	2
2. Interaction Type Extensions	3
2.1. Matching Elements	3
2.1.1. QTI assessment structure	3
2.1.2. Extension document entry	5
3. Custom Operators Results Processing Extensions	6
4. Custom Interactions	7
4.1. CP manifest for custom interactions	7
4.2. The QTI customInteraction	7
4.3. The custom interaction JSON manifest	8
4.4. Custom interaction script API	8
4.5. The custom interaction's iframe	9
4.6. Additional remarks	9
5. Adaptive Testing	10

1. Introduction

This document is part of the technical documentation set for the *Dutch Exam Profile v4.0 (DEP)*. The DEP is a generic standard for specifying and transferring exams between parts of a computer based exam system (originally developed for the Dutch computer based exam system Facet). A common introduction and an overview of the available documentation and other source of information can be found in [DEP-FO]. The main goal of this technical documentation set is to supply content and software developers with enough information to implement the DEP.

This part of the documentation set describes the various adaptations done to the QTI standard.

1.1. DEP Documentation Overview

The DEP v4.0 June 2, 2015 documentation set consist of several documents. References to other documents are always in square brackets, e.g. [DEP-MER]. The following table provides an overview:

Document:	Title:
[DEP-DRG]	Dutch Exam Profile v4.0 - Documentation Reading Guide
[DEP-FO]	Dutch Exam Profile v4.0 - Functional Overview
[DEP-MER]	Dutch Exam Profile v4.0 - Manifest Extensions Reference
[DEP-EDR]	Dutch Exam Profile v4.0 - Extension Documents Reference
[DEP-QAR]	Dutch Exam Profile v4.0 - QTI Adaptations Reference

1.2. Legend

- Columns marked **Pg.** indicate on which page more information can be found about a particular element, attribute or type.
- Columns marked **#** hold information about the occurrences of the element/attribute described
- The following indicators for occurrences are used:

Indicator:	Description:
1	The element or attribute must occur exactly once (single mandatory).
?	The element or attribute can occur (single optional).
*	The element must occur one or more times (multiple mandatory).
+	The element can occur zero or more times (multiple optional).

1.3. Copyright Notice

This document contains extracts from or derivatives of the IMS Global Learning Consortium specifications listed below and are available from the IMS Global Learning Consortium at <http://www.imsglobal.org>:

- IMS Question and Test Interoperability Specification, Version 2.1
<http://www.imsglobal.org/question/>
Copyright © 2012
- IMS Content Packaging Specification, Version 1.2
<http://www.imsglobal.org/content/packaging/>
Copyright © 2007

Use of these specifications is subject to the IMS Global license agreement available here: <http://www.imsglobal.org/license.html>. The testing and granting of conformance certification to the specification is reserved by IMS Global.

Question and Test Interoperability® (QTI®) are trademarks of IMS Global Learning Consortium, Inc. in the United States and/or other countries. For more information: <http://www.imsglobal.org/copyright.html>.

2. Interaction Type Extensions

This chapter describes the extensions the DEP defines on top of the normal QTI interaction types.

2.1. Matching Elements

Matching Elements is a mechanism to combine QTI `<hottextInteraction>` with `<extendedTextInteraction>`. For instance: The student is asked to rectify the misspelled words in a sentence. He/she must click on the words he/she thinks are wrong and in another part of the screen a text box appears in which the word in the correct spelling can be entered. Here is an example (in Dutch):

Klik de foute woorden aan en verbeter ze:

1 Did is een 2 voorbeelt van het item.

Geef de juiste spelling aan:

1	Di
2	voorbeelt

2.1.1. QTI assessment structure

The item body of the QTI assessment that leads to the matching elements interaction looks like this (simplified, lay-out related stuff left out):

```
<itemBody>
  <!-- Section with the hottextInteraction. All words are marked as hottext: -->
  <div>
    <hottextInteraction responseIdentifier="RESPONSE" maxChoices="0" class="markCorrect">
      <p>Klik de foute woorden aan en verbeter ze:</p>
      <p>
        <span><hottext id="ID-HT_A" identifier="HT_A">Did</hottext></span>
        <span><hottext id="ID-HT_B" identifier="HT_B">is</hottext></span>
        <span><hottext id="ID-HT_C" identifier="HT_C">een</hottext></span>
        <span><hottext id="ID-HT_D" identifier="HT_D">voorbeelt</hottext></span>
        <span><hottext id="ID-HT_E" identifier="HT_E">van</hottext></span>
        <span><hottext id="ID-HT_F" identifier="HT_F">het</hottext></span>
        <span><hottext id="ID-HT_G" identifier="HT_G">item</hottext></span>.</p>
      </hottextInteraction>
    </div>
  <!-- Section with the extendedTextInteraction-s to allow entering the right answers: -->
  <p>Geef de juiste spelling aan:</p>
  <div>
    <extendedTextInteraction id="HT_A-ti" responseIdentifier="RESPONSE-A" expectedLength="140"
      expectedLines="2"/>
    <extendedTextInteraction id="HT_B-ti" responseIdentifier="RESPONSE-B" expectedLength="140"
      expectedLines="2"/>
    <extendedTextInteraction id="HT_C-ti" responseIdentifier="RESPONSE-C" expectedLength="140"
      expectedLines="2"/>
    <extendedTextInteraction id="HT_D-ti" responseIdentifier="RESPONSE-D" expectedLength="140"
      expectedLines="2"/>
    <extendedTextInteraction id="HT_E-ti" responseIdentifier="RESPONSE-E" expectedLength="140"
      expectedLines="2"/>
    <extendedTextInteraction id="HT_F-ti" responseIdentifier="RESPONSE-F" expectedLength="140"
      expectedLines="2"/>
    <extendedTextInteraction id="HT_G-ti" responseIdentifier="RESPONSE-G" expectedLength="140"
      expectedLines="2"/>
  </div>
</itemBody>
```

As you can see, the sentence to correct is in `<hottextInteraction>` and *all* words are marked up as `<hottext>`. For every word (every `<hottext>` entry) there is an `<extendedTextInteraction>`.

To make this work, the outcome and response declaration section of the assessment (which in the XML comes before the item's body) must look like this:

```
<!-- Main response declaration: -->
<responseDeclaration baseType="identifier" cardinality="multiple" identifier="RESPONSE">
  <correctResponse>
    <value>HT_A, HT_D</value>
  </correctResponse>
</responseDeclaration>
<!-- Response declarations for all extendedTextInteraction-s: -->
<responseDeclaration baseType="string" cardinality="single" identifier="RESPONSE-A">
  <defaultValue>
    <value>Did</value>
  </defaultValue>
  <correctResponse>
    <value>Dit</value>
  </correctResponse>
</responseDeclaration>
<responseDeclaration baseType="string" cardinality="single" identifier="RESPONSE-B"/>
<responseDeclaration baseType="string" cardinality="single" identifier="RESPONSE-C"/>
<responseDeclaration baseType="string" cardinality="single" identifier="RESPONSE-D">
  <defaultValue>
    <value>voorbeelt</value>
  </defaultValue>
  <correctResponse>
    <value>voorbeeld</value>
  </correctResponse>
</responseDeclaration>
<responseDeclaration baseType="string" cardinality="single" identifier="RESPONSE-E"/>
<responseDeclaration baseType="string" cardinality="single" identifier="RESPONSE-F"/>
<responseDeclaration baseType="string" cardinality="single" identifier="RESPONSE-G"/>
<!-- Main outcome declaration for passing the score: -->
<outcomeDeclaration cardinality="single" identifier="SCORE" baseType="integer">
  <defaultValue>
    <value>0</value>
  </defaultValue>
</outcomeDeclaration>
```

- There is a main response declaration (identifier="REPONSE") that defines the identifiers of the <hottext> entries that need correcting.
- There are response declarations for *all* <extendedTextInteraction> entries. The ones that need correcting have a defined correct response.
- And of course there is an outcome declaration called SCORE to pass the assessment's score.

Finally there is the response processing to consider. For the example it looks like this:

```
<responseProcessing>
  <responseCondition>
    <responseIf>
      <and>
        <member>
          <baseValue baseType="identifier">HT_A</baseValue>
          <variable identifier="RESPONSE"/>
        </member>
        <not>
          <member>
            <baseValue baseType="identifier">HT_B</baseValue>
            <variable identifier="RESPONSE"/>
          </member>
        </not>
        <not>
          <member>
            <baseValue baseType="identifier">HT_C</baseValue>
            <variable identifier="RESPONSE"/>
          </member>
        </not>
        <member>
          <baseValue baseType="identifier">HT_D</baseValue>
          <variable identifier="RESPONSE"/>
        </member>
        <not>
          <member>
            <baseValue baseType="identifier">HT_E</baseValue>
            <variable identifier="RESPONSE"/>
          </member>
        </not>
        <not>
          <member>
            <baseValue baseType="identifier">HT_F</baseValue>
            <variable identifier="RESPONSE"/>
          </member>
        </not>
        <not>
          <member>
            <baseValue baseType="identifier">HT_G</baseValue>
            <variable identifier="RESPONSE"/>
          </member>
        </not>
        <stringMatch caseSensitive="true">
          <variable identifier="RESPONSE-A"/>
          <baseValue baseType="string">Dit</baseValue>
        </stringMatch>
        <stringMatch caseSensitive="true">
          <variable identifier="RESPONSE-D"/>
          <baseValue baseType="string">voorbeeld</baseValue>
        </stringMatch>
      </and>
      <setOutcomeValue identifier="SCORE">
        <sum>
          <baseValue baseType="integer">1</baseValue>
          <variable identifier="SCORE"/>
        </sum>
      </setOutcomeValue>
    </responseIf>
  </responseCondition>
</responseProcessing>
```

The result of all this is that the score is set to the value 1 when:

- The words to correct are selected (the <hottext> entries with identifiers HT_A and HT_D)
- The other words are *not* selected
- The entries in the <extendedTextInteraction> entries are correct

2.1.2. Extension document entry

Finally, the <hottext> and the <extendedTextInteraction> entries in the QTI document described above must be linked. This is done by adding a <matchingElements> section to the extension document belonging to the assessment (for more information about extension documents please refer to [DEP-EDR]). For the example above it looks like this:

```
<matchingElements>
  <matchingElement selectedElementId="ID-HT_A" matchingElementId="HT_A-ti"/>
  <matchingElement selectedElementId="ID-HT_B" matchingElementId="HT_B-ti"/>
  <matchingElement selectedElementId="ID-HT_C" matchingElementId="HT_C-ti"/>
  <matchingElement selectedElementId="ID-HT_D" matchingElementId="HT_D-ti"/>
  <matchingElement selectedElementId="ID-HT_E" matchingElementId="HT_E-ti"/>
  <matchingElement selectedElementId="ID-HT_F" matchingElementId="HT_F-ti"/>
  <matchingElement selectedElementId="ID-HT_G" matchingElementId="HT_G-ti"/>
</matchingElements>
```

3. Custom Operators Results Processing Extensions

The DEP has a number of so called "custom operators" that can be used inside a QTI item's results processing. These allow for comparing results in a specific way that is not possible in straight QTI.

Here is an example of using the custom operator `depcp:ParseCommaDecimal` for comparing real numbers. It allows the use of the Dutch comma decimal separator (instead of the dot):

```
<responseCondition>
  <responseIf>
    <and>
      <and>
        <gte>
          <customOperator definition="depcp:ParseCommaDecimal">
            <variable identifier="RESPONSE" />
          </customOperator>
          <baseValue baseType="float">4.20</baseValue>
        </gte>
        <lte>
          <customOperator definition="depcp:ParseCommaDecimal">
            <variable identifier="RESPONSE" />
          </customOperator>
          <baseValue baseType="float">4.30</baseValue>
        </lte>
      </and>
    </and>
    <setOutcomeValue identifier="SCORERESPONSE">
      <sum>
        <baseValue baseType="integer">1</baseValue>
      </sum>
    </setOutcomeValue>
  </responseIf>
  <responseElse>
    <setOutcomeValue identifier="SCORERESPONSE">
      <baseValue baseType="integer">0</baseValue>
    </setOutcomeValue>
  </responseElse>
</responseCondition>
```

The following custom operators are defined:

Operator	Description
<code>depcp:FacetMathMLEqual</code>	Converts a formula response into MathML. This custom operator is deprecated, do not use.
<code>depcp:ParseCommaDecimal</code>	Converts a value containing decimal number in string format into a float, using the comma as decimal separator instead of the dot.
<code>depcp:Trim</code>	Removes all whitespace from a value.
<code>depcp:ToAscii</code>	Converts a value into straight ASCII by changing all characters with diacritics into there non-diacritical counterparts (e.g. change <i>één</i> into <i>een</i>).

Remark: The use of the `depcp:` prefix in every custom operator creates the impression that this is a namespace prefix. This is however not the case: The name of every custom operator must simply always begin with `depcp:`. No namespace binding (`xmlns:depcp="..."`) for this prefix needs to exist.

4. Custom Interactions

The DEP allows the use of so called *custom interactions*. These are interactions with functionality that cannot be expressed in QTI directly, for instance something graphical like directly manipulating a curve. Custom interactions are expressed in a combination of (X)HTML and JavaScript.

For custom interactions to be able to communicate with the player and its environment, the DEP defines a small scripting API. This can be used to, for instance, pass the final result of the interaction back to the environment for further processing.

4.1. CP manifest for custom interactions

Defining and using a custom interaction starts in the CP manifest (see [DEP-MER for more information]). All files that comprise the custom interaction *must* be defined as a manifest resource (a `<resource>` element with `type="webcontent"`). Here is an example:

```
<resources>

  <!-- Main QTI custom interaction item: -->
  <resource identifier="ITM-ci" type="imsqti_depitem" href="depitems/ci.xml" depcp:version="1"
    depcp:guid="e75f2bcb-66f8-449c-a6db-332816b37dfd">
    <metadata>
      <depitemMetadata xmlns:imsqti="http://www.imsglobal.org/xsd/imsqti_v2p1"
        xmlns:depcp="http://www.imsglobal.org/xsd/imsqcp_ext_vlp2">
        <depcp:maxScore>0</depcp:maxScore>
      </depitemMetadata>
    </metadata>
    <file href="depitems/ci.xml" depcp:role="depItem"/>
    <dependency identifierref="RES-1087_WiKB_knikkers_manifest_json"/>
    <dependency identifierref="RES-1087_WiKB_knikkers_js"/>
    <dependency identifierref="RES-1087_WiKB_knikkers_css"/>
    <dependency identifierref="RES-body_html"/>
    <dependency identifierref="RES-cito_generated_css"/>
  </resource>

  <!-- Resources for the files that belong to the custom interaction: -->
  <resource identifier="RES-1087_WiKB_knikkers_manifest_json" type="webcontent"
    href="ref/json/1087_WiKB_knikkers.manifest.json" depcp:version="1"
    depcp:guid="a6d2a65-c70e-4670-b4bf-5d8e960596cb">
    <file href="ref/json/1087_WiKB_knikkers.manifest.json"/>
  </resource>
  <resource identifier="RES-1087_WiKB_knikkers_js" type="webcontent" href="ref/script/1087_WiKB_knikkers.js"
    depcp:version="1" depcp:guid="ea6afd67-49c4-4dea-83a7-d106d1a850e9">
    <file href="ref/script/1087_WiKB_knikkers.js"/>
  </resource>
  <resource identifier="RES-1087_WiKB_knikkers_css" type="webcontent"
    href="ref/style/1087_WiKB_knikkers.css" depcp:version="1"
    depcp:guid="16f20cdb-a280-43ec-9881-413c01a23c03">
    <file href="ref/style/1087_WiKB_knikkers.css"/>
  </resource>
  <resource identifier="RES-body_html" type="webcontent" href="ref/html/body.html" depcp:version="1"
    depcp:guid="1666eed0-ea1-4c5b-9c0b-f7b66414464d">
    <file href="ref/html/body.html"/>
  </resource>

  <!-- ... -->

</resources>
```

As you can see, the actual QTI custom interaction item (the first resource, with `identifier="ITM-ci"`) mentions *all* its constituent files as a dependency on their resources (with `<dependency>` elements).

4.2. The QTI customInteraction

A DEP custom interaction must use an XHTML `<object>` element inside a QTI `<customInteraction>` element to point to a JSON manifest. This JSON manifest defines the actual custom interaction. Here is an example:

```
<itemBody>
  <customInteraction responseIdentifier="RESPONSE">
    <html:object xmlns:html="http://www.w3.org/1999/xhtml"
      type="application/javascript"
      data="../ref/json/1087_WiKB_knikkers.manifest.json"
      height="800"
      width="500"
    >
      <param name="responseLength" value="5" valueType="DATA"/>
    </html:object>
  </customInteraction>
</itemBody>
```

The values for the attributes of this `<object>` element are:

Attribute:	Description:
type	The type of the main script that provides the interaction's functionality. <div> <div>application/ecmascript</div> <div>application/javascript</div> </div>
data	Relative reference to the JSON manifest for this interaction.
width	The width of the iframe the custom interaction will be displayed in.

Attribute:	Description:
height	The height of the iframe the custom interaction will be displayed in.

If the required number of response fields is greater than 1, it can be specified with an optional `<param>` element with the following attributes:

Attribute:	Description:
name	Value always <code>responseLength</code> .
value	The number of required response fields (integer).

4.3. The custom interaction JSON manifest

Central to a custom interaction is its JSON manifest. Here is an example:

```
{
  "script": [
    "ref/script/1087_WiKB_knikkers.js"
  ],
  "style": [
    "ref/style/1087_WiKB_knikkers.css"
  ],
  "media": [
    "ref/html/body.html"
  ]
}
```

The JSON manifest consists of a single object with the following properties:

Property:	#	Type:	Description:
script	1	array	References to the script files that together comprise the custom interaction. These references must be ordered according to their respective cascade and dependency requirements.
style	?	array	Optional references to the style sheets for custom interaction. These references must be ordered according to their respective cascade and dependency requirements.
media	1	array	Optional references to the media files that together comprise the custom interaction.

Important additional remarks for use with Facet:

1. Due to limitations set by the Facet CP importer, the filename of the JSON manifest *must* always ends with `manifest.json`
2. The references from the JSON manifest to the other files must be *relative to the location of the CP manifest* (and not relative to the location of the JSON manifest itself as might have been expected).

4.4. Custom interaction script API

The scripts for a custom interaction can communicate with the surrounding player environment through the custom interaction script API. The name of the object the API is defined on is `CES`.

Function:	Description:
<code>setResponse(String data)</code>	Sets the candidate response "as is" (the format of the response and its state is the responsibility of the implementation).
<code>getResponse()</code>	Gets the previously saved candidate response (if any).
<code>getMedia()</code>	Gets an array of resolved manifest media URLs.
<code>setStageHeight(Number verticalMargin)</code>	Convenience function to adjust the height of the generated iframe to the current height of the custom interaction's content height. If the content's visual size exceeds its layout box (e.g. by using <code>box-shadow</code>), an optional vertical margin can be specified.

Here is a skeleton example script that uses this API:

```
// use a resolved media object
var media = CES.getMedia();
var image = document.createElement('IMG');
image.src = media[0];
// image.getAttribute('src')
// => resolved/path/to/resources/widget.png
// ...
// save response and state...
function serialize() {
  // return serialized response and state as string, e.g. JSON, XML
}
CES.setResponse(serialize());
// restore response and state...
function unserialize(string) {
  // return unserialized response and state from string
}
var response =
unserialize(CES.getResponse());
```

4.5. The custom interaction's iframe

The interaction is rendered to an iframe element whose `contentDocument`:

- `compatMode` is `CSS1Compat` (standards compliant)
- The `<head>` element contains the following elements (in order):
 - A copy of the `<base>` element of the iframe's `ownerDocument`
 - An empty `<title>` element
 - Copies of all linked style sheets of the iframe's `ownerDocument`
 - One `<link>` element for each style sheet specified in the JSON manifest file
 - a `<script>` element that assigns the public custom JavaScript interaction API to a global identifier in the iframe's `contentWindow`
 - one `<script>` element for each script file specified in the JSON manifest file
- The `<body>` element is empty
- All other HTML must be generated by the script, using the `load` and `DOMContentLoaded` events (as you would expect in any normal document).

Here is an example of such an iframe's document:

```
<!DOCTYPE html>
<html>
  <head>
    <base href="/base/path/of/parent/document/">
    <title></title>
    <link href="parent/document/stylesheets" rel="stylesheet">
    <link href="resolved/path/to/resources/widget.css" rel="stylesheet">
    <script>var CES = window.parent.customInteraction["RESPONSE"];</script>
    <script src="resolved/path/to/resources/library.js">
    <script src="resolved/path/to/resources/widget.js">
  </head>
  <body></body>
</html>
```

4.6. Additional remarks

If you need to save media references as part of the state, save their media array indexes, not the resolved values. The resolved value can vary, depending on the application's context your custom interaction is loaded in.

Since the custom interaction is rendered in an iframe, several of its `ownerDocument`'s tools cannot interact with it, notably:

- magnifier
- text marker
- spell checker
- text to speech

While the iframe provides a separate environment that should prevent most accidental scripting conflicts (e.g. global variables, augmented native or host objects), it should not be considered a sandbox in the security sense.

5. Adaptive Testing

Adaptive testing is going through the assessments in a `testSection` in a test based on the results of the answers given. It allows for instance to skip assessments about subjects the student seems to know enough about or vice versa.

The selections within the `testSection` in a test, necessary for adaptive testing are done by so called *adaptive test modules*. How to create these modules and their functionality is outside the scope of this documentation. There is a separate document (in Dutch) called "API Adaptive Module" describing the API for adaptive test modules.

Handled here is how to insert these modules into a DEP CP.

An adaptive test driver consists of two documents:

- The adaptive test *module*. This is a binary file that contains the code for the adaptations.
- An adaptive test *driver*. This is an XML file with definitions for the module. Its structure and contents depends on the functionality of the module and is therefore not defined here.

Both the driver and the module must be defined in the CP manifest as a resource with `type="imsqti_depmodule"` and `type="imsqti_depdriver"` respectively:

```
<resource identifier="adaptivemodule.so" type="imsqti_depmodule" href="adaptive/adaptivemodule.so"
  depcp:version="1" depcp:guid="guid-of-module">
  <file href="adaptive/adaptivemodule.so"/>
</resource>
<resource identifier="adaptivedriver.xml" type="imsqti_depdriver" href="adaptive/adaptivedriver.xml"
  depcp:version="1" depcp:guid="guid-of-driver">
  <file href="adaptive/adaptivedriver.xml"/>
</resource>
```

To link an adaptive test module to a `<testSection>` part of the QTI test that needs to become adaptive, you need to add an `<adaptiveSelection>` element to the test's extension document (for more information about extension documents, please refer to [DEP-EDR]). Here is an example:

```
<depTest xmlns="http://www.imsglobal.org/xsd/imsqti_ext_v2p1">
  <!-- ... -->
  <testPart qtiTestPartIdentifierRef="RES-6b359e65-8d4c-4948-bf8b-417d11b7f454" title="ToetsDeel_01">
    <testSection qtiAssessmentSectionIdentifierRef="RES-7cc8b148-170b-4e30-9d34-2fa4b88a2a43">
      <!-- ... -->
      <adaptiveSelection>
        <moduleIdRef>guid-of-module</moduleIdRef>
        <driverIdRef>guid-of-driver</driverIdRef>
      </adaptiveSelection>
    </testSection>
  </testPart>
</depTest>
```