AIR Equation Scoring Engine Documentation

Release 0.9.2

American Institutes for Research

August 05, 2013

CONTENTS

Contents:

CHAPTER

ONE

QUICK START: REST INTERFACE ON APACHE 2.4 FOR WINDOWS

Contents

This document describes how to get up-and-running with the **AIR Equation Scoring Engine**'s RESTful web interface on a **Windows** machine running **Apache httpd** version 2.4.

The configuration described herein is meant for testing and development only. You will probably want to set up something more robust for a production system.

1.1 Get the Dependencies

1.1.1 Apache httpd

You will need to install the **Apache httpd** server. These instructions were written for version 2.4.4, but they should apply to any version of Apache 2.2 or 2.4. I used the Windows 64 binary build from Apache Lounge Choose an appropriate download for your architecture (32-bit or 64-bit)

After you have installed the Apache binary, you will want to set it up so that it runs as a Windows service. Open a command window in the Apache `bin` directory, and execute:

```
httpd.exe -k install
```

More information on running Apache on Windows can be found at http://httpd.apache.org/docs/current/platform/windows.html.

1.1.2 Python 2.7

You will need to have the **Python** programming language installed. The software was developed and tested on Python 2.7.4 for 64-bit Windows. The software should run fine on any version of 2.7. It has not been tested on Python 2.6 or

earlier. It will not run on any version of Python 3

A Windows installer for the latest version of Python may be downloaded from here: http://www.python.org/download/

I recommend installing Python's **setuptools** (https://pypi.python.org/pypi/setuptools) and **pip** (https://pypi.python.org/pypi/pip). packages. Once the setuptools are installed, you can install pip using:

```
c:\Python27\Scripts\easy-install pip
```

Then use pip to install the remaining Python dependencies:

```
c:\Python27\Scripts\pip install sympy
c:\Python27\Scripts\pip install django
c:\Python27\Scripts\pip install djangorestframework
```

1.1.3 mod_wsgi

An Apache plugin called **mod_wsgi** provides the bridge between Apache and Python. It is easiest to acquire a pre-compiled binary for this package. One is available from http://www.lfd.uci.edu/~gohlke/pythonlibs/#mod_wsgi Be sure to download the version that is appropriate for your Apache version, Python version and Windows architecture. After downloading the zip archive, extract the single file that it contains (should be called `mod_wsgi.so`) and drop it into the Apache modules directory (this is `C:\Program Files\Apache24\modules` on my machine)

1.1.4 Fiddler

Finally, in order to test the setup, you will need some way of sending a **POST** request to the Apache server. These instructions assume that you are using the **Fiddler** utility (http://fiddler2.com/get-fiddler). If you prefer a different utility, then modify them accordingly.

1.2 Download the Software

Download the latest version of the **AIR Equation Scoring Engine** from https://bitbucket.org/sbacoss/equationscorer/downloads

Unzip the downloaded archive into a directory of your choosing. For security reasons, this should *not* be a location under your server's document root. In the rest of this document, we will refer to this directory as *{Equation-Scorer-Root}*

1.3 Configure Apache

All of the remaining configuration is done in the main Apache configuration file, `httpd.conf` On my machine, this file is located at `C:\Program Files\Apache24\conf\httpd.conf`. Open this file in your favorite text editor.

First, we need to enable the **mod_wsgi**. Locate the section that has a whole bunch of lines that begin `LoadModule`, and add the following:

```
# MODULE mod_wsgi ADDED XX/XX/20XX
LoadModule wsgi_module modules/mod_wsgi.so
```

This line enables the link between Apache and Python, but Apache still doesn't know what Python code to invoke when it sees a particular HTTP request. To enable that, add the following lines at the end of `httpd.conf`

```
# Configuration for equation scorer app
WSGIPythonHome C:/Python27
WSGIPythonPath {Equation-Scorer-Root}/eqscorer_rest;{Equation-Scorer-Root}/lib
WSGIScriptAlias /eq-scorer-rest {Equation-Scorer-Root}/eqscorer_rest/eqscorer_rest/wsgi.py
```

Where you replace *{Equation-Scorer-Root}*, everywhere it appears, with the actual directory to which you unzipped the AIR Equation Scoring Engine. If your Python is not installed in the standard location, you will have to change `WSGIPythonHome` as well.

Although Apache now knows how to run the engine, it will refuse to do so unless you tell it that it is allowed to. To do that, add the following lines to the end of `httpd.conf`

```
<Directory "{Equation-Scorer-Root}/eqscorer_rest/eqscorer_rest">
    <Files "wsgi.py">
      Order allow,deny
      Allow from all
      Require all granted
    </Files>
</Directory>
```

Again, replacing *{Equation-Scorer-Root}* with the correct value.

Now, from the Windows *Services* control panel, restart the Apache service. If Apache fails to start, then chances are you mistyped something in the `httpd.conf`

To test the service, fire up Fiddler, and find the *Composer* window. Select **POST** for the request method. For the address, use `http://127.0.0.1/eq-scorer-rest/isequivalent`, for the request headers, you should specify:

```
Content-Type: application/json; charset=utf-8
Accept: application/json
```

And for the request body, you should use:

```
{
  "answer":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo></mo><mn>3</mn
  "rubric":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo></mo><mn>3</mn
  "parameters":{}
}
```

Push the *Execute* button to submit the request. After a short delay (while Python starts and loads all of its libraries), you should get a response of `200 OK`. If you switch to Fiddler's *Inspectors* tab, you should see something like this:

```
HTTP/1.1 200 OK
Date: Fri, 14 Jun 2013 16:38:55 GMT
Server: Apache/2.4.4 (Win64) mod_wsgi/3.5-BRANCH Python/2.7.4
Vary: Accept,Cookie
Allow: POST, OPTIONS
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8

1f
{"correct": true, "reason": ""}
0
```

Now read the *API Documentation*, and have fun!

CHAPTER

TWO

AIR EQUATION SCORING ENGINE RESTFUL WEB INTERFACE

Contents

The REST interface has a single access point. A `POST` to `http://my_site/isequivalent/` with a JSON payload, returns a JSON object containing the results of the call.

2.1 POST /isequivalent/

Test whether an answer is mathematically equivalent to the rubric

2.1.1 Parameters

answer (String, required) The test answer

A JSON string containing a MathML expression.

Per the JSON specification, all quotation marks within the MathML must be preceded by backslashes. Any other "special" characters in the answer may be included in one of three ways:

- They may simply be embedded as unicode encoded using the charset of the request (e.g., ∀)
- They may be escaped as XML escape sequences (e.g., `&#x2264;`)
- They may be escaped as JSON escape sequences (e.g., `\u2264`)

rubric (String, required) The rubric against which the answer will be compared.

A JSON string. Depending on the **sympy_rubric** parameter, this will be a MathML expression (see *answer*) or an expression parsable by Sympy.

parameters (Dictionary, optional) A dictionary of parameters that modify how sympy tests for equivalency. If omitted, all of the parameters take on their default values. The keys permitted in **parameters** are the following:

sympy_rubric (`true` or `false`, optional) Default `false`. If `true`, the **rubric** argument will be treated as a Sympy string. If `false`, the **rubric** will be treated as MathML.

allow_change_of_variable (`true` or `false`, optional) Default `false`

allow_simplify (`true` or `false`, optional) Default `true`

trigIdentities (`true` or `false`, optional) Default `false`

logIdentities (`true` or `false`, optional) Default `false`

forceAssumptions (`true` or `false`) Default `false`

2.1.2 Normal Return Values

Status 200 OK

Body A JSON dictionary containing two entries

correct (`true` or `false`) Whether or not the answer was equivalent to the rubric

reason (`true` or `false`) If **correct** was `true`, then this will be an empty string. Otherwise it will describe the reason that the answer was rejected. This may be one of the following

`Answer is not equivalent to rubric` SymPy determined that the answer and the rubric were not equivalent

`Unable to parse answer as mathml` The answer was not valid mathml, or contains mathml constructs that are not currently supported by this package.

`Unable to compare answer with rubric` Sympy raised an error while attempting to test the answer and the rubric for equivalence

2.1.3 Return on Error

If the **answer** field on the request is not parsable MathML, then a "normal" return value will be reported, with a status of `200`, a **correct** value of `false`, and a **reason** value containing a message describing the reason for the failure. In all other cases, a non-`2xx` error status will be returned.

The body of the error response will depend on where in the processing of the request the error occurred. If an error occurs in the Equation Scorer REST code, then the response code will be `400`, and a JSON structure will be returned identical to the one returned for a "normal" response.

2.1.4 Examples

Example 1:

```
POST /isequivalent/ HTTP/1.1
Host: example.com
Content-Type: application/json; charset=UTF-8
Accept: application/json

{
  "answer":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo></mo><mn>3</mn
  "rubric":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo></mo><mn>3</mn
}
```

Response 1:

```
HTTP/1.0 200 OK
Date: Thu, 13 Jun 2013 18:27:39 GMT
Server: WSGIServer/0.1 Python/2.7.4
Vary: Accept, Cookie
Content-Type: application/json; charset=utf-8
Allow: POST, OPTIONS

{"correct": true, "reason": ""}
```

Example 2 (Incorrect answer):

```
POST /isequivalent/ HTTP/1.1
Host: example.com
Content-Type: application/json; charset=UTF-8
Accept: application/json

{
  "answer":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo>=</mo><mn>3</m
  "rubric":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo><</mo><mn>3</mn
}
```

Response 2:

```
HTTP/1.0 200 OK
Date: Thu, 13 Jun 2013 18:27:39 GMT
Server: WSGIServer/0.1 Python/2.7.4
Vary: Accept, Cookie
Content-Type: application/json; charset=utf-8
Allow: POST, OPTIONS

{"correct": false, "reason": "Answer is not equivalent to rubric"}
```

Example 3 (Parameters for equivalence check):

```
POST /isequivalent/ HTTP/1.1
Host: example.com
Content-Type: application/json; charset=UTF-8
Accept: application/json

{
  "answer":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo><</mo><mn>3</mn
  "rubric":"<math xmlns=\"http://www.w3.org/1998/Math/MathML\"><mn>1</mn><mi>x</mi><mo><</mo><mn>3</mn
  "parameters":{ "allow_change_of_variable":true }
}
```

Response 3:

```
HTTP/1.0 200 OK
Date: Thu, 13 Jun 2013 18:27:39 GMT
Server: WSGIServer/0.1 Python/2.7.4
Vary: Accept, Cookie
Content-Type: application/json; charset=utf-8
Allow: POST, OPTIONS

{"correct": true, "reason": ""}
```

CHAPTER

THREE

AIR EQUATION SCORING ENGINE PYTHON API

Contents

3.1 Module airscore

The airscore module is the primary Python entry point for the **AIR Equation Scoring Engine**. It provides two functions. The function process_mathml_data() accepts inputs in the MathML format, and returns an equivalent expression in a form that can be parsed by the Sympy symbolic mathematics library. The function isEquivalent() Uses Sympy to determine if two sets of expressions, equations, or inequalities are mathematically equivalent.

In order to compare a test answer, expressed in MathML, with a rubric, also expressed in MathML, the two functions are used in conjunction, like this:

```
answer_txt = unicode( process_mathml_data( answer_mathml ) )
rubric_txt = unicode( process_mathml_data( rubric_mathml ) )
is_correct = isEquivalent( answer_txt, rubric_txt )
```

airscore.**isEquivalent**(*response*, *rubric*, *allowChangeOfVariable=False*, *allowSimplify=True*, *trigIdentities=False*, *logIdentities=False*, *forceAssumptions=False*)

True if Sympy is able to determine that the two expressions are equivalent.

This function requires two parameters: a test answer and a rubric. Each of these is a string. Each string defines an equality, an inequality, or an expression in a form that can be parsed by the Sympy symbolic mathematics library. Alternatively, the answer or rubric may be a list of such equations, inequalities or expressions, enclosed in square brackets and separated by commas.

Additional optional parameters control the manipulations that Sympy will make when attempting to determine the equivalence of the response and the rubric

Parameters

- **response** (*str*) – The test response
- **rubric** (*str*) – The rubric for the test questions
- **allowChangeOfVariables** (*bool*) –

- **allowSimplify** (*bool*) –
- **trigIdentities** (*bool*) –
- **logIdentities** (*bool*) –
- **forceAssumptions** (*bool*) –

airscore.**process_mathml_data**(*mathml_string*, *encoding=None*)
Convert MathML into a form that can be understood by Sympy

The provided string must either contain a <mathml:math> element as the root, or it must contain a <response> element (no namespace), which contains zero or more <mathml:math> elements as children.

Parameters

- **mathml_string** (str() or unicode(). If the provided object is unicode(), it will be converted to a string by the specified encoding.) – A string containing a <mathml:math> or <response> element
- **encoding** (*str*) – Name of the encoding that will be used in parsing the mathml_string. Defaults to UTF-8

Returns A MathExpressionList object equivalent to the MathML original. This object's __unicode__() method returns a string that can be passed to Sympy

3.2 Module airscore.mathmlsympy.math_expression

Most of the classes and methods in airscore.mathmlsympy.math_expression are mainly of interested to those who intend to extend the MathML parsing engine to understand a wider selection of MathML elements. Two classes may be of interest to ordinary users of these libraries, however. These are the MathExpressionList class that is returned by airscore.process_mathml_data(), and the MathExpression objects that it contains.

class airscore.mathmlsympy.math_expression.**MathExpression**(*math_node*)
The representation of a MathML expression, equality, or inequality that has been returned by the parser.

math_node
airscore.mathmlsympy.mathml_containser.MathmlMath - The XML element tree (see xml.etree.ElementTree) that was returned by the parser. Elements within the tree will be represented by subclasses of airscore.mathmlsympy.base_mathml_element.BaseMathmlElement, which in turn subclasses xml.etree.ElementTree.Element

sympy_response
list of str - A list of strings representing equations, inequalities or expressions. This is the result of parsing the XML represented by math_node. The list will consist of more than one element if the math_node contains more than one equality or inequality operator. Specifically, if the MathML represents something like:

```
A = B = C < D
```

then the list will contain three entries, corresponding to:

```
A = B
B = C
C < D
```

class airscore.mathmlsympy.math_expression.**MathExpressionList**
A container for multiple MathExpression elements.

The `__str__()` and `__unicode__()` methods have been overridden to return strings that will be useful to Sympy.

CHAPTER

FOUR

EXTENDING THE AIR EQUATION SCORING ENGINE

Contents

The **AIR Equation Scoring Engine** supports a limited subset of the MathML standard. The most likely place that users will want to extend the engine is to implement support for MathML constructs that are not currently supported. This can be accomplished by writing subclasses of `airscore.mathmlsympy.base_mathml_element.BaseMathmlElement` or `airscore.mathmlsympy.mathml_containers.BaseMathmlContainer`, and registering the new classes with the parser using the `airscore.mathmlsympy.parser.mathml_element()` decorator. The interfaces for the relevant classes and methods are described below.

4.1 BaseMathmlElement

class airscore.mathmlsympy.base_mathml_element.**BaseMathmlElement**(*tag*, *attrib={}*, ***extra*)

This is the base class for all MathML elements in the XML tree

is_implicit_addend
bool() - This node may appear as the implicit addend of an integer (as the fractional part of a mixed number).

is_implicit_multiplicand
bool() - When a number or a symbol appears to the left of this node, an implicit multiplication should be performed.

is_inequality
bool() - This is an equal sign or inequality operator. A chained equation can be broken on this node.

is_number
bool() - This node contains a number (digits, decimal points, etc).

is_non_neg_integer
bool() - This node is a non-negative integer. This flag is used to detect when the numerator and denom-

inator of a fraction contain simple numbers, allowing us to use the fraction as part of a mixed number (see `is_implicit_addend`).

validate_max_children
`int()` - Maximum number of children permitted for this node.

validate_min_children
`int()` - Minimum number of children permitted for this node.

validate_no_text
`bool()` - If `True`, it is an error for this node to contain text directly. Text may still exist inside of nested nodes.

validate_required_attributes
`set` of `str()` - A set containing the names of required attributes. Not namespace aware.

decoded_text
`unicode` The text content

The default implementation decodes a limited dictionary of common unicode values to sympy equivalents.

get_sympy_text()
Get a string representing this node, including children

This is the main method that you will need to override in order to control how this node and its children are rendered in the sympy output.

The default implementation simply returns the `decode_text` attribute.

Override `to_sympy()` instead if you need to control how this node relates to its neighbors.

Returns `unicode` - The string representing this node in a sympy expression

pick_subclass()
Change the leopard's spots to zebra stripes

The `xml.etree.ElementTree` parsing mechanism forces us to choose the class for new elements when the start tag has been parsed, but before any of the content has been read. There are a few MathML constructs for which we need different classes, but they are represented by the same start tag. Our parser calls this method after the end tag has been processed, in order to give the element a chance to make any changes it needs to make to finalize its class selection.

In most cases, this method does nothing (the default behavior). In a few cases, however, this method will assign a new value to the instance's `__class__` property in order to change the object into an instance of a subclass of its original class.

We are using an admittedly obscure Python "feature," and I can't recommend that you make a habit of altering the classes of existing objects. But for this limited purpose it seemed the cleanest solution.

Returns `None`

to_sympy(*tail=<airscore.mathmlsympy.partial_sympy_object.PartialSympyObject object at 0x3a94b10>*)
Return oneself as a "partial sympy object"

You will need to override this routine if you are changing the way this node combines with its neighbor nodes. The default implementation concatenates this node with its right-hand neighbor, adding a mutliplcation operator to the list first if the right-hand neighbor is a suitable implicit multiplicand.

Override `get_sympy_text()` instead if you need to change how this node and its children are represented in the output, but not how this node is related to its siblings.

Parameters **tail** (airscore.mathmlsympy.partial_sympy_object.PartialSympyObject) – The head of a linked list of sympy objects which will become the tail of the newly created object

Returns airscore.mathmlsympy.partial_sympy_object.PartialSympyObject

validate()

Validate the node content

This method is called during the processing of the XML end tag, immediately after the call to pick_subclass(). Subclasses should perform any required validation of the newly-created MathML object. An error should be raised for a validation failure (usually ValueError)

The default implementation performs the following steps:

- Validate the number of children against the validate_min_children and validate_max_children properties
- If validate_no_text is True, confirm that the element contains no text.
- Confirm that children (if any) are subclasses of BaseMathmlElement
- Confirm that any attributes listed in validate_required_attributes are present (but perform no validation on the attribute values)

Returns None

Raises ValueError

4.2 BaseMathmlContainer

class airscore.mathmlsympy.mathml_containers.**BaseMathmlContainer**(*tag*, *attrib={}*, ***extra*)

The base class for all MathML container elements.

The base class for all MathML elements that can contain an arbitrary list of other MathML elements. This includes elements like <math> and <row>, as well as elements listed in the MathML spec as containing an implicit <row> element.

get_sympy_text_list()

Return a list containing text representations of simple equations or inequalities.

This method is the main loop of the parser for most MathML expressions. Most containers will not need to override this method, but it is worth understanding how it works. One oddity worth noting is that the parser parses the expression from right to left, instead of the usual left-to-right.

If the expression is a "chained" equation, containing more than one equals or inequality operator, then this function will return multiple strings in its return list. Each return value will be a simple equation or inequality–i.e., one that contains only one equals or inequality operator.

If the expression is already a simple equation or inequality, then a list containing a single string represnting that equation or inequality will be returned.

If the expression contains no equality or inequality operators, then a list containing a single string representing that expression will be returned.

Returns list() of str()

4.3 mathml_element()

airscore.mathmlsympy.parser.**mathml_element**(**args*)

A decorator which registers the decorated class as a mathml element class

This decorator registers the association between a class and an element name for the MathMLBuilder. In order to have the MathMLBuilder use a particular class instead of the default xml.etree.ElementTree.Element class to represent a given XML element, decorate your class definition with this decorator.

There are three permitted calling conventions. You can use the decorator without arguments, like so:

```
@mathml_element
class bob( BaseMathmlElement ):
    ...
```

in which case the new class will be used for elements named <bob> in the MathML namespace. This usage is not ideal, as it requires your Python class to have the same name (including case) as the MathML element.

You avoid this problem by specifying an element name like this:

```
@mathml_element( 'bob' )
class MathMLBob( BaseMathmlElement ):
    ...
```

Finally, if you have a class that should be associated with multiple MathML tag names, you can specify all of the names as arguments to the mathml_element decorator()

```
@mathml_element( 'bob', 'jim', 'joe' )
class MathMLBob( BaseMathmlElement ):
    ...
```

In every case, the classes will only be used for elements in the MathML namespace (*http://www.w3.org/1998/Math/MathML*)

No special effort beyond the use of this decorator is required to register new classes for handling MathML elements. However, you must be certain that the modules containing your classes have been imported before attempting to process the XML data.

4.4 PartialSympyObject

class airscore.mathmlsympy.partial_sympy_object.**PartialSympyObject**(*el*, *tail*)

An element in a linked list that represents a Sympy expression

Parameters

- **el** (BaseMathmlElement) – The mathml element from which this object is derived
- **tail** (PartialSympyObject) – The next item in the list

next

PartialSympyObject - The next rightward neighbor in the list.

is_closed

bool() - Used in balancing absolute value bars. True if the parser has encountered an odd number of absolute value bars to the right of this point.

is_implicit_mutliplicand
bool() - The *is_implicit_multiplicand* attribute of the airscore.mathmlsympy.base_mathml_element.BaseM that generated this object.

is_implicit_addend
bool() - The *is_implicit_addend* attribute of the airscore.mathmlsympy.base_mathml_element.BaseMathm that generated this object.

is_number
bool() - The *is_number* attribute of the airscore.mathmlsympy.base_mathml_element.BaseMathmlElem that generated this object.

text
unicode() - The result of the BaseMathmlElement.get_sympy_text() method of the airscore.mathmlsympy.base_mathml_element.BaseMathmlElement object that generated this object.

get_sympy_text()
The concatenated text attributes of this node and all of the nodes to its right.

Returns unicode

itertext()
Iterate through the linked list, returning the text attribute of each node.

Returns iterator of unicode

CHAPTER

FIVE

AIR EQUATION SCORING ENGINE VER. 0.9.2 – READ ME FIRST!

5.1 COPYRIGHT

Equation Scoring Engine Copyright (c) 2013 American Institutes for Research

Distributed under the AIR Open Source License, Version 1.0 See accompanying file AIR-License-1_0.txt or at https://bitbucket.org/sbacoss/equationscorer/wiki/AIR_Open_Source_License_1.0

The AIR Equation Scoring Engine is a Python application for evaluating answers to tests in mathematics. The answers are submitted in MathML, and are evaluated by comparing to a MathML rubric, using the Sympy symbolic mathematics package.

The AIR Equation Scoring Engine consists of two assemblies. The main engine has a Python API. It depends only on Sympy. This is located in the package "airscore"

The second assembly provides a RESTful web interface to the main engine. This assembly depends on Django and the Django REST framework, in addition to the dependencies for the main assembly.

5.2 DEPENDENCIES

The AIR Equation Scoring Engine was developed on Python 2.7.5.

All of the required libraries can be installed from PyPi using "pip install <package-name>"

Sympy: http://sympy.org/en/index.html Django: https://www.djangoproject.com/ Django REST framework: http://django-rest-framework.org/

5.3 INSTALLATION

After downloading the project, add the directory "lib" to your Python path, or else copy or symlink the contents of "lib" into a directory that is on your Python path.

If you are using the REST interface, you must also add the directory "eqscorer_rest" to your Python path, or else copy or symlink the contents of "eqscorer_rest" into a directory that is on your Python path.

5.4 NOTES FOR ECLIPSE USERS

The project directory may be opened as a PyDev-Django project in eclipse. However Eclipse makes default assumptions about the location of the settings module that are wrong for this project. Eclipse users should open the project properties window and select the "PyDev - Django" page. On this page, set the value for "Django settings module" to "eqscorer_rest.settings"

If you do not perform this step, you will receive errors when attempting to run the test server via "Run As -> PyDev: Django"

CHAPTER

SIX

AIR EQUATION SCORING ENGINE RELEASE NOTES

6.1 Version 0.9.2

- Documentation changes

6.2 Version 0.9.1

- Updates to logic for "is-equivalent" assessment
- Support for <mathml:mtext> element
- Added `\Box` as alias for `\u25fb` (the box character)
- Protect against stack overflow on long expressions
- Permit rubric to be submitted in sympy syntax or as mathml

6.3 Version 0.9.0

- Initial release

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

a