Documentation Guide

The following page is a guide on how to publish and troubleshoot documentation of MXNet's APIs. Each time you contribute to MXNet, it is highly likely that your contribution will require documentation. This guide will help you build the documentation sets, check for errors and troubleshoot them.

Setup for Writing Docs

You will need to be setup for building the website to see the docs output.

Building the website

Review: How to Build the Website
Copy the `docs/settings.ini` file to `settings`. Update `settings` to turn off (set to 0) all of the doc sets. Python runs by default. To run only Python, your settings would look like:

```
[document_sets_default]
clojure_docs = 0
doxygen_docs = 0
r_docs = 0
scala_docs = 0
```

Then copy `settings` over `settings.ini`. This way when you checkout again and lose your `settings.ini`, you can easily overwrite it again with your local `settings` file.

If you need to build a different doc set like Scala, you would set `scala_docs = 1`.

Tip for building and seeing Sphinx errors:

Sphinx caches the files between builds, so you will not see the error messages until you touch a page and it regenerates it. To get a full output, run:

```
cd docs # from mxnet root
make clean
make html USE_OPENMP=1
```

Writing Docs

Docs are written in a combination of markdown and restructured text (rst). When the docs are built Sphinx will convert markdown to an intermediate rst format or incorporate regular rst before converting everything to an html output for the website.

The following resources can help you with writing rst, using special Sphinx directives, and incorporating all of these things into documentation that is published on the MXNet website.

Sphinx info: http://openmdao.org/releases/0.0.9/docs/quick-ref/sphinx.html

Sphinx docs / rst primer: http://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

rst Quick reference: http://docutils.sourceforge.net/docs/user/rst/quickref.html

rst Cheatsheet: https://github.com/ralsina/rst-cheatsheet/blob/master/rst-cheatsheet.rst

Markdown and rst comparison and incompatibilities: https://gist.github.com/dupuy/1855764

pydocstyle - analysis tool for checking Python docstrings: https://github.com/PyCQA/pydocstyle

napoleon Sphinx plugin for Google and Numpy style docs: http://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html

Troubleshooting

Use tools to help convert doc strings like https://github.com/dadadel/pyment

Tip for formatting errors

The issues are related to validation of the reStructured Text (.rst) format. Utilizing a linter for rst might be a good way to go. Sphinx errors on rst typically don't report the right line number. It is relative to the rst code and not the python code that wraps it. Then it will be off due to imports of other modules, so line numbers in the error message are misleading.

Indentation error

`ERROR: Unexpected indentation.`

This can happen in a couple of ways:

1. Codes samples that are indented without letting Sphinx know that it is coming.
2. A new line was made for readability or lint and only indents were used to identify it. See new line fixes.

Code Sample Fixes

A simple fix is to use the double colon (::) in the preceding line. For example the following update that resolved the error switched out the comma for a the double colon.

~~a dictionary of Graphviz attribute names and values. For example,~~

```
        a dictionary of Graphviz attribute names and values. For example::
            ``node_attrs={"shape":"oval","fixedsize":"false"}``
```

Another approach to this that would work for multi-line example code blocks is to use the `code-block` directive. The `emphasize-lines` option is optional.

```
    .. code-block:: python
       :emphasize-lines: 3,5

       def some_function():
           interesting = False
           print 'This line is highlighted.'
           print 'This one is not...'
           print '...but this one is.'
```

New Line Fixes

This error is often preceded by an unexpected indentation:

`WARNING: Block quote ends without a blank line; unexpected unindent.`

You have options:

- End the previous line with a backslash and retain the indenting.

~~- for input shape = (10,5,4), shape = (-1,0), output shape would be~~

```
 - for input shape = (10,5,4), shape = (-1,0), output shape would be \
   (40,5).
```

- Adjust the indent to match the current block

```
    multi_precision: bool, optional
```

~~Flag to control the internal precision of the optimizer.~~
~~``False`` results in using the same precision as the weights (default),~~
~~``True`` makes internal 32-bit copy of the weights and applies gradients~~
~~in 32-bit precision even if actual weights used in the model have lower precision.`<~~
~~Turning this on can improve convergence and accuracy when training with float16.~~

```
        Flag to control the internal precision of the optimizer.
        ``False`` results in using the same precision as the weights (default),
        ``True`` makes internal 32-bit copy of the weights and applies gradients
        in 32-bit precision even if actual weights used in the model have lower precision.
        Turning this on can improve convergence and accuracy when training with float16.
```

Emphasis Warning

```
WARNING: Inline emphasis start-string without end-string.
```

1. Emphasis should be used like `*make this italics*`. Or like `**make this bold**`. Some people may have tried using `*_something` in the code's comment block, but this doesn't work.

2. There are places where docstrings try to showcase python definitions taking in variable args as follows:

```
"""
A function that conforms to "def function(F, data, *args)".
"""
```

Here *args can be misconstrued to be the beginning of emphasis. To solve this, we could do the following:

```
r"""
A function that conforms to "def function(F, data, \*args)".
"""
```

You can allow use a literal string with the double back ticks. This can be a bit more readable in the code. For example:

```
r"""
A function that conforms to "def function(F, data, ``*args``)".
"""
```

Unexpected section title - severe warning

```
SEVERE: Unexpected section title.
```

There are a set of expected section titles. You can find the list of section headings in the docs for the Napoleon extension for Sphinx.

This often can be a typo in the section title name. For example a fix for this:

~~Parameter~~
~~---------~~
```
    Parameters
    ----------
```

Or someone may have used this in their docs:

```
Example Usage:
```

You would change this to `Examples` to comply with the expected section headings.

Undefined substitution reference error

```
ERROR: Undefined substitution referenced: "x".
```

In some math examples, people will use |var| to express the absolute value of something. The usage of pipes around a name is special to Sphinx and triggers a substitution of that value for something that is defined in the doc later. Switching to actual code or escaping the math can fix this.

```
            absolute value str((|x|/size(x)).asscalar()).
            absolute value str((abs(x)/size(x)).asscalar()).
```

An inline literal would work too:

```
            absolute value str((|x|/size(x)).asscalar()).
            ``absolute value str((|x|/size(x)).asscalar())``
```

Or a regular code block would work:

```
"""
Some description ::

    absolute value str((|x|/size(x)).asscalar())

Where you made sure there were line breaks before and after the indented code block
```

Finally, you could use the styles mentioned in the code example fixes.

Failed to import warning

```
/home/ubuntu/incubator-mxnet/docs/api/python/symbol/symbol.md:1: WARNING: failed to import broadcast_logical_not
```

Correct the function name in the .md file. For example, in case of the above warning, there is not operator called broadcast_logical_not. Changed it to logical_not to fix the warning.

Incorrect cross-linking Warning

```
/home/ubuntu/incubator-mxnet/python/mxnet/rnn/rnn.py:docstring of mxnet.rnn.save_rnn_checkpoint:None: WARNING:
more than one target found for cross-reference u'RNNCell': mxnet.rnn.RNNCell, mxnet.gluon.rnn.RNNCell
```

Specify the correct class's reference. For example, for the above error, the fix was to specify mxnet.rnn.RNNCell

Tip for unknown or ambiguous references

Sphinx gets confused when there are more than one function or variable with the same name. Things like name, type, shape, or reshape, save, or load are generic and appear often in different modules. References to these should be specific.

When referencing a local class, prefix it with a dot as seen in the following example:

~~"""Retrieves a :py:class:`Constant` with name ``self.prefix+name``. If not found,~~

```
"""Retrieves a :py:class:`.Constant` with name ``self.prefix+name``. If not found,
```

If the target class is elsewhere call it out specifically. For example, consider the following docstring:

```
     Parameters
        ----------
        shapes : a tuple of (name, shape)
        types : a tuple of  (name, type)
```

Sphinx can easily get confused by the generic usage of “name” or “shape”. Sphinx won't know what this refers to. It will link up to whatever it saw last and throw an error or warning about the issue. This can be specified by the following:

```
        Parameters
        ----------
        shapes : a tuple of (mxnet.symbol.Symbol.name, mxnet.ndarray.NDArray.shape)
        types : a tuple of  (mxnet.symbol.Symbol.name, type)
```

After a Returns line is used, Sphinx expects the next line to contain an entity and it will attempt to link it up. If it is ambiguous, you will get an error there as well. For example, this will cause an error:

```
Returns

--------

Constant
```

Fix this as follows:

```
Returns

--------

:py:class:`.Constant`
```

Tip for using Aliases

You also have the option using an alias.
1) Create an alias. rst uses an underscore after the name to trigger an alias. For example:

```
        Parameters
        ----------
        shapes : a tuple of (name_, shape_)
        types : a tuple of  (name_, type)
```

Then you define the alias in the rst where you invoke the generation of the docs. This usually goes at the bottom of the page. Any Sphinx automodule on that page that uses those aliases will be updated to use the link you define. Note that defining the alias puts the underscore at the beginning, whereas calling it puts the underscore at the end.

~~~
```eval_rst
.. _name: mxnet.symbol.Symbol.name
.. _shape: mxnet.ndarray.NDArray.shape

 ```
~~~

Note the line break after the Sphinx directives in the rst block. This is important to include. Here is a code example is for the alias and for using it in the Python docstring.