Documentation

This is a page to record processes and procedures related to documentation, especially tips for developers who would like to contribute to the documentation.

The entire rendered Impala documentation set is now available on the Documentation tab of the Apache Impala web site. The rendered documentation is available in HTML and PDF. These links include all of the currently available Impala documentation: the release notes, SQL reference, installation, administration, and development guides.

Authoring Logistics

Source of the main Impala documentation (SQL Reference and such) is in XML, using the DITA XML format and buildable by an open source toolchain. For reference information about DITA tags and attributes, see the OASIS spec for the DITA XML standard.

Version control is through git. The doc source files live underneath the docs/ subdirectory, in the same repository as the Impala code.

XML Filename Conventions

Most Impala doc content is in files that live under the docs/topics/ subdirectory. (The exception is the set of reusable fragments that live in the docs/shared /impala_common.xml file.)

All the files under topics/ have a filename prefix of impala_ and a suffix of .xml.

Each SQL statement, query option, aggregate function, class of built-in scalar functions, and major areas such as "security" and "performance", has an associated XML file with a name starting with impala_, and then is the name of the thing with any spaces replaced by underscores. For example, impala_create_table.xml, impala_max.xml, impala_mem_limit.xml.

Building

For instructions to set up the doc build environment and produce HTML and PDF, see the docs/README.md file. Once the build environment is set up, you can do local builds of HTML, PDF, or both by issuing one of the following commands in the docs/ directory:

```
make html

make pdf

make all # Creates both HTML and PDF
```

Building either output format is a good way to check that all cross-references resolve properly, something that is not caught by file-at-a-time validation such as is done by `xmllint`. It is also useful to proofread content where the source is heavily tagged, or uses constructs like conref= that make it hard to reason about the exact text that appears in the output.

XML Source Conventions

Frequently Used Tags

The essential tags for documenting programming-type information are similar to the tags found in HTML. For example:

```
  <p>

    A paragraph with some text.

  </p>

  <ul>

    <li>Unordered list item.</li>

  </ul>

  <ol>

    <li>Ordered list item.</li>

  </ol>

<codeblock>

Sample code and/or output.

Spacing is preserved in output, so don't add extra indentation.

Left-justify the codeblock start and end tags.

</codeblock>
```

Internal and External Links

Reuse via conref= Attribute

Most reuse in DITA comes in the form of conref= attributes. These use link-style notation, similar to href= attributes in <xref> tags. The content of the linked-to element is substituted for whichever tag has the conref= attribute.

For example, the way to interpret this tag:

```
<p conref="../shared/impala_common.xml#common/views_vs_identifiers"/>
```

is like so:

- In the source file ../shared/impala_common.xml is a topic with the ID "common".
- Inside that topic is a paragraph tag with an ID "views_vs_identifiers".
- Whatever is inside that other paragraph will get substituted for the <p conref=...> one.

This makes the conref technique a good choice wherever there is some ambiguity about where to put some warning, notice, or tip. Often such wording involves using feature X along with feature Y. Do you put the advice in the topic for feature X, or the one for feature Y? Answer: put it both places, and be confident that the wording is identical in both spots.

Impala also uses this technique for short snippets of boilerplate wording, like "The default for this option is 0." or bolded pseudo-subheads like "Usage notes:". In addition to making sure the wording is identical in all locations, this lets us make future edits to the boilerplate by editing only a single spot. (For example, maybe someday we change "0" to "zero" in the preceding example.)

Almost all conref'ed elements have their canonical content in the file docs/shared/impala_common.xml, to make conref= attributes easy to construct (no guessing at filenames) and avoid broken conref= references if content is moved between or removed from files in the topics/ folder.

The most frequently conref'ed elements are <p> tags, where an entire explanation or instruction can be reused, even if it stretches across multiple sentences. There are also some <note> elements where a warning is reused, although often a reusable warning is structured as a regular <note> with a conref'ed <p> inside it. (That way, the text can appear as a warning in one place, and regular text in another place.) In and around the File Formats pages, rows from the introductory table of file format info are reused on the individual pages for each file format. The crucial step when reusing an element that requires additional inner or outer tags is to construct the minimal structure, with empty tags wherever needed to make the XML valid. For example, because a <ul> requires at least one <li> inside it, the way to reuse an entire unordered list is like so:

```
<ul conref="file_containing_real_list#id_of_concept_in_that_file/id_of_the_list_itself">

  <li/> <!-- The contents of this empty list element are replaced by however many list items are in the "real"
list -->

</ul>
```

XML Validation

Before checking in doc updates, make sure the XML markup is valid. To perform a basic check for mismatched tags, missing or extra or unescaped delimiters, and other mistakes that are straightforward to detect, run the command:

```
xmllint --noout <filename>.xml
```

You can run the command with multiple filenames, for example all the modified files that are about to be committed. (On a modern laptop, it only takes about 0.1 seconds to run xmllint --noout on all the XML files in the topics/ subdirectory, so you can be generous about validating frequently or for more files than strictly necessary.)

If everything is OK, the command produces no output. If there is any output, it will show the approximate line(s) where the XML error occurs.

By default, the xmllint command only checks the validity aspects that apply to any XML document, for example, that every start <tag> is matched by a closing </tag>. But not that the actual tag and attribute names are ones recognized by the DITA XML dialect. That extra level of checking requires additional setup. (Instructions pending.)

As mentioned previously, building either HTML or PDF output provides an extra level of validation, producing warnings or errors for any cross-file mismatches between IDs, filenames, and conref / topicref / mapref / href elements. HTML is typically the format that's faster and less memory-intensive to build.

Commit Messages

By convention, to allow for filtering in JIRA reports and email notifications, commits that are purely for documentation updates include the eyecatcher string DOCS inside square brackets. (Literal example with square brackets omitted for the moment because Confluence wants to turn it into a link.)

Often, an IMPALA- JIRA for a new feature or improvement has a subtask (with its own JIRA number) for documenting the new feature or the behavior changes. When including a JIRA number in the subject line of the commit message, prefer to use the JIRA number for the parent issue rather than the one for the doc-specific subtask.

gerrit Reviews