Malformed JSON command output when non-ASCII strings are present

Ceph accepts non-ascii input when setting e.g. the name of a pool. Subsequently, when using human-formatted CLI output non-ASCII characters are converted to '?'. When using JSON-formatted CLI output, we get some encoding other than UTF-8 (possibly ISO-8859).

My LANG environment variable is en_GB.UTF-8, Ceph version is 0.72.2.

To reproduce:

```
# ceph osd pool create ? 1024
pool '?' created
# ceph osd lspools
9 data,10 metadata,11 rbd,14 test123,15 ?,
# ceph -f json-pretty osd dump -o out.json
# python -c "import json; json.load(open('out.json'))"
```

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python2.7/json/__init__.py", line 278, in load
    **kw)
  File "/usr/lib/python2.7/json/__init__.py", line 326, in loads
    return _default_decoder.decode(s)
  File "/usr/lib/python2.7/json/decoder.py", line 366, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
UnicodeDecodeError: 'utf8' codec can't decode byte 0xc4 in position 0: unexpected end of data
```

The printing as '?' is kind of annoying (good luck deleting a pool if you can't remember what character you used to create it!), but the output of non-UTF8 JSON is definitely a problem: anyone relying on parsing JSON output will have their world completely break if someone creates a pool with a non-ASCII character in the name.

Setting category to ceph cli because this would be the first place to start looking to see where we're breaking this, but it could be the Formatter stuff itself that's getting the output encoding wrong perhaps.

<table>
<thead>
<tr>
<th>Status:</th>
<th>Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>High</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Dan Mick</td>
</tr>
<tr>
<td>Category:</td>
<td>ceph cli</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
<tr>
<td>Source:</td>
<td>other</td>
</tr>
<tr>
<td>Tags:</td>
<td>hammer, firefly</td>
</tr>
<tr>
<td>Backport:</td>
<td>ceph-qa-suite:</td>
</tr>
<tr>
<td>Regression:</td>
<td>No</td>
</tr>
<tr>
<td>Severity:</td>
<td>3 - minor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
</table>
Ceph accepts non-ascii input when setting e.g. the name of a pool. Subsequently, when using human-formatted CLI output non-ASCII characters are converted to '?'. When using JSON-formatted CLI output, we get some encoding other than UTF-8 (possibly ISO-8859).

My LANG environment variable is en_GB.UTF-8, Ceph version is 0.72.2.

To reproduce:

```
# ceph osd pool create ? 1024
pool '?' created
# ceph osd lspools
9 data,10 metadata,11 rbd,14 test123,15 ?,
# ceph -f json-pretty osd dump -o out.json
# python -c "import json; json.load(open('out.json'))"
```

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python2.7/json/__init__.py", line 278, in load
    **kw)
  File "/usr/lib/python2.7/json/__init__.py", line 326, in loads
    return _default_decoder.decode(s)
  File "/usr/lib/python2.7/json/decoder.py", line 366, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
UnicodeDecodeError: 'utf8' codec can't decode byte 0xc4 in position 0: unexpected end of data
```

The printing as '?' is kind of annoying (good luck deleting a pool if you can't remember what character you used to create it!), but the output of non-UTF8 JSON is definitely a problem: anyone relying on parsing JSON output will have their world completely break if someone creates a pool with a non-ASCII character in the name.

Setting category to ceph cli because this would be the first place to start looking to see where we're breaking this, but it could be the Formatter stuff itself that's getting the output encoding wrong perhaps.

**Related issues:**
- Copied to Backport #11708: Malformed JSON command output when non-ASCII string... | Resolved | 02/10/2014
- Copied to Backport #11709: Malformed JSON command output when non-ASCII string... | Resolved | 02/10/2014
json_spirit: use utf8 internally when parsing \uHHHH

When the python CLI is given non-ASCII characters, it converts them to \uHHHH escapes in JSON. json_spirit parses these internally into 16 bit characters, which could only work if json_spirit were built to use std::wstring, which it isn't; it's using std::string, so the high byte ends up being zero'd, leaving the low byte which is effectively garbage.

This hack^H^H^H^H change makes json_spirit convert to utf8 internally instead, which can be stored just fine inside a std::string.

Note that this implementation still assumes \uHHHH escapes are four hex digits, so it'll only cope with characters in the Basic Multilingual Plane. Still, that's rather a lot more characters than it could cope with before ;)

(For characters outside the BMP, Python seems to generate escapes in the form \uHHHHHHHH, i.e. 8 hex digits, which the current implementation doesn't expect to see)

Fixes: #7387

Signed-off-by: Tim Serong <tserong@suse.com>

Revision 84b00f18 - 05/10/2015 09:40 AM - Tim Serong
json_spirit: use utf8 internally when parsing \uHHHH

When the python CLI is given non-ASCII characters, it converts them to \uHHHH escapes in JSON. json_spirit parses these internally into 16 bit characters, which could only work if json_spirit were built to use std::wstring, which it isn't; it's using std::string, so the high byte ends up being zero'd, leaving the low byte which is effectively garbage.

This hack^H^H^H^H change makes json_spirit convert to utf8 internally instead, which can be stored just fine inside a std::string.

Note that this implementation still assumes \uHHHH escapes are four hex digits, so it'll only cope with characters in the Basic Multilingual Plane. Still, that's rather a lot more characters than it could cope with before ;)

(For characters outside the BMP, Python seems to generate escapes in the form \uHHHHHHHH, i.e. 8 hex digits, which the current implementation doesn't expect to see)

Fixes: #7387

Signed-off-by: Tim Serong <tserong@suse.com>
(cherry picked from commit 8add15b86e7aaef41397ab8fa9e77ee7957eb607)

11/19/2015
Conflicts:
src/test/mon/osd-pool-create.sh

Changed $CEPH_MON to 127.0.0.1 -- the CEPH_MON was introduced after firefly to allow tests to run in parallel. Back in firefly all tests use the same port because 127.0.0.1 was hardcoded. We can't conveniently backport all that's necessary for tests to run in parallel, therefore we keep the 127.0.0.1 hardcoded.

Revision 678b3e60 - 05/15/2015 10:24 AM - Tim Serong

json_spirit: use utf8 internally when parsing \uHHHH

When the python CLI is given non-ASCII characters, it converts them to \uHHHH escapes in JSON. json_spirit parses these internally into 16 bit characters, which could only work if json_spirit were built to use std::wstring, which it isn't; it's using std::string, so the high byte ends up being zero'd, leaving the low byte which is effectively garbage.

This hack^H^H^H^H change makes json_spirit convert to utf8 internally instead, which can be stored just fine inside a std::string.

Note that this implementation still assumes \uHHHH escapes are four hex digits, so it'll only cope with characters in the Basic Multilingual Plane. Still, that's rather a lot more characters than it could cope with before ;)

(For characters outside the BMP, Python seems to generate escapes in the form \uHHHHHHHH, i.e. 8 hex digits, which the current implementation doesn't expect to see)

Fixes: #7387

Signed-off-by: Tim Serong <tserong@suse.com>
(cherry picked from commit 8add15b86e7aaef41397ab8fa9e77ee7957eb607)

History

#1 - 02/10/2014 03:46 PM - John Spray
Urgh, redmine apparently can't cope with unicode either! Here's what the snipped should look like http://pastebin.com/KzWab33X

#2 - 02/11/2014 09:40 AM - Ian Colle
IMO this is a big can of worms having to do with Ceph itself; I'm willing to bet that none of the internal routines handle wide characters or even try to.

Yeah -- I don't really want to open that can of worms either, and we'll add some extra hygiene here in Calamari.

Before engaging with the larger problem of funny strings internally, it would be a good thing to scrub this stuff at the entry and exit points, i.e. the command interface that is receiving these arguments. It's draconian, but given that the unicode handling is known-broken, it might make sense to force ASCII both on the way in and on the way out (do the same '?' nastiness in the JSON output that we do in the other output).