Coverage for **common/lib/xmodule/xmodule/modulestore/xml_importer** : 38%

264 statements | 100 run | 164 missing | 0 excluded

```
import logging
import os
import mimetypes
from path import path

from xblock.core import Scope

from .xml import XMLModuleStore, ImportSystem, ParentTracker
from xmodule.modulestore import Location
from xmodule.contentstore.content import StaticContent
from .inheritance import own_metadata
from xmodule.errortracker import make_error_tracker
from .store_utilities import rewrite_nonportable_content_links

log = logging.getLogger(__name__)


def import_static_content(modules, course_loc, course_data_path, static_content_store, target_location_namespace,
                          subpath='static', verbose=False):

    remap_dict = {}

    # now import all static assets
    static_dir = course_data_path / subpath

    verbose = True

    for dirname, dirnames, filenames in os.walk(static_dir):
        for filename in filenames:

            try:
                content_path = os.path.join(dirname, filename)
                if verbose:
                    log.debug('importing static content {0}...'.format(content_path))

                fullname_with_subpath = content_path.replace(static_dir, '')  # strip away leading path from the name
                if fullname_with_subpath.startswith('/'):
                    fullname_with_subpath = fullname_with_subpath[1:]
                content_loc = StaticContent.compute_location(target_location_namespace.org, target_location_namespace.course, fullname_with_subpath)
                mime_type = mimetypes.guess_type(filename)[0]

                with open(content_path, 'rb') as f:
                    data = f.read()

                content = StaticContent(content_loc, filename, mime_type, data, import_path=fullname_with_subpath)

                # first let's save a thumbnail so we can get back a thumbnail location
                (thumbnail_content, thumbnail_location) = static_content_store.generate_thumbnail(content)

                if thumbnail_content is not None:
                    content.thumbnail_location = thumbnail_location

                #then commit the content
                try:
                    static_content_store.save(content)
                except Exception as err:
                    log.exception('Error importing {0}, error={1}'.format(fullname_with_subpath, err))

                #store the remapping information which will be needed to subsitute in the module data
                remap_dict[fullname_with_subpath] = content_loc.name
            except:
                raise

    return remap_dict


def import_from_xml(store, data_dir, course_dirs=None,
                    default_class='xmodule.raw_module.RawDescriptor',
                    load_error_modules=True, static_content_store=None, target_location_namespace=None,
                    verbose=False, draft_store=None,
                    do_import_static=True):
    """
    Import the specified xml data_dir into the "store" modulestore,
    using org and course as the location org and course.

    course_dirs: If specified, the list of course_dirs to load. Otherwise, load
    all course dirs

    target_location_namespace is the namespace [passed as Location] (i.e. {tag},{org},{course}) that all modules in the should be remapped to
    after import off disk. We do this remapping as a post-processing step because there's logic in the importing which
    expects a 'url_name' as an identifier to where things are on disk e.g. ../policies/<url_name>/policy.json as well as metadata keys in
    the policy.json. so we need to keep the original url_name during import

    do_import_static: if False, then static files are not imported into the static content store.  This can be employed for courses which
                      have substantial unchanging static content, which is to inefficient to import every time the course is loaded.
```

```
                    Static content for some courses may also be served directly by nginx, instead of going through django.

    """

    xml_module_store = XMLModuleStore(
        data_dir,
        default_class=default_class,
        course_dirs=course_dirs,
        load_error_modules=load_error_modules
    )

    # NOTE: the XmlModuleStore does not implement get_items() which would be a preferable means
    # to enumerate the entire collection of course modules. It will be left as a TBD to implement that
    # method on XmlModuleStore.
    course_items = []
    for course_id in xml_module_store.modules.keys():

        if target_location_namespace is not None:
            pseudo_course_id = '/'.join([target_location_namespace.org, target_location_namespace.course])
        else:
            course_id_components = course_id.split('/')
            pseudo_course_id = '/'.join([course_id_components[0], course_id_components[1]])

        try:
            # turn off all write signalling while importing as this is a high volume operation
            if pseudo_course_id not in store.ignore_write_events_on_courses:
                store.ignore_write_events_on_courses.append(pseudo_course_id)

            course_data_path = None
            course_location = None

            if verbose:
                log.debug("Scanning {0} for course module...".format(course_id))

            # Quick scan to get course module as we need some info from there. Also we need to make sure that the
            # course module is committed first into the store
            for module in xml_module_store.modules[course_id].itervalues():
                if module.category == 'course':
                    course_data_path = path(data_dir) / module.data_dir
                    course_location = module.location

                    log.debug('======> IMPORTING course to location {0}'.format(course_location))

                    module = remap_namespace(module, target_location_namespace)

                    if not do_import_static:
                        module.lms.static_asset_path = module.data_dir                  # for old-style xblock where this was actually linked to kvs
                        module._model_data['static_asset_path'] = module.data_dir
                        log.debug('course static_asset_path={0}'.format(module.lms.static_asset_path))

                    log.debug('course data_dir={0}'.format(module.data_dir))

                    # cdodge: more hacks (what else). Seems like we have a problem when importing a course (like 6.002) which
                    # does not have any tabs defined in the policy file. The import goes fine and then displays fine in LMS,
                    # but if someone tries to add a new tab in the CMS, then the LMS barfs because it expects that -
                    # if there is *any* tabs - then there at least needs to be some predefined ones
                    if module.tabs is None or len(module.tabs) == 0:
                        module.tabs = [{"type": "courseware"},
                                       {"type": "course_info", "name": "Course Info"},
                                       {"type": "discussion", "name": "Discussion"},
                                       {"type": "wiki", "name": "Wiki"}]  # note, add 'progress' when we can support it on Edge

                    import_module(module, store, course_data_path, static_content_store, course_location,
                                  target_location_namespace or course_location, do_import_static=do_import_static)

                    course_items.append(module)

            # then import all the static content
            if static_content_store is not None and do_import_static:
                _namespace_rename = target_location_namespace if target_location_namespace is not None else course_location

                # first pass to find everything in /static/
                import_static_content(xml_module_store.modules[course_id], course_location, course_data_path, static_content_store,
                                      _namespace_rename, subpath='static', verbose=verbose)

            elif verbose and not do_import_static:
                log.debug('Skipping import of static content, since do_import_static={0}'.format(do_import_static))

            # no matter what do_import_static is, import "static_import" directory

            # This is needed because the "about" pages (eg "overview") are loaded via load_extra_content, and
            # do not inherit the lms metadata from the course module, and thus do not get "static_content_store"
            # properly defined.   Static content referenced in those extra pages thus need to come through the
            # c4x:// contentstore, unfortunately.  Tell users to copy that content into the "static_import" subdir.

            simport = 'static_import'
            if os.path.exists(course_data_path / simport):
                _namespace_rename = target_location_namespace if target_location_namespace is not None else course_location

                import_static_content(xml_module_store.modules[course_id], course_location, course_data_path, static_content_store,
                                      _namespace_rename, subpath=simport, verbose=verbose)

```

```
178
179             # finally loop through all the modules
180             for module in xml_module_store.modules[course_id].itervalues():
181                 if module.category == 'course':
182                     # we've already saved the course module up at the top of the loop
183                     # so just skip over it in the inner loop
184                     continue
185
186                 # remap module to the new namespace
187                 if target_location_namespace is not None:
188                     module = remap_namespace(module, target_location_namespace)
189
190                 if verbose:
191                     log.debug('importing module location {0}'.format(module.location))
192
193                 import_module(module, store, course_data_path, static_content_store, course_location,
194                               target_location_namespace if target_location_namespace else course_location,
195                               do_import_static=do_import_static)
196
197             # now import any 'draft' items
198             if draft_store is not None:
199                 import_course_draft(xml_module_store, store, draft_store, course_data_path,
200                                     static_content_store, course_location, target_location_namespace if target_location_namespace
201                                     else course_location)
202
203         finally:
204             # turn back on all write signalling
205             if pseudo_course_id in store.ignore_write_events_on_courses:
206                 store.ignore_write_events_on_courses.remove(pseudo_course_id)
207                 store.refresh_cached_metadata_inheritance_tree(
208                     target_location_namespace if target_location_namespace is not None else course_location
209                 )
210
211     return xml_module_store, course_items
212
213
214 def import_module(module, store, course_data_path, static_content_store,
215                   source_course_location, dest_course_location, allow_not_found=False,
216                   do_import_static=True):
217
218     logging.debug('processing import of module {0}...'.format(module.location.url()))
219
220     content = {}
221     for field in module.fields:
222         if field.scope != Scope.content:
223             continue
224         try:
225             content[field.name] = module._model_data[field.name]
226         except KeyError:
227             # Ignore any missing keys in _model_data
228             pass
229
230     module_data = {}
231     if 'data' in content:
232         module_data = content['data']
233     else:
234         module_data = content
235
236     if isinstance(module_data, basestring) and do_import_static:
237         # we want to convert all 'non-portable' links in the module_data (if it is a string) to
238         # portable strings (e.g. /static/)
239         module_data = rewrite_nonportable_content_links(
240             source_course_location.course_id, dest_course_location.course_id, module_data)
241
242     if allow_not_found:
243         store.update_item(module.location, module_data, allow_not_found=allow_not_found)
244     else:
245         store.update_item(module.location, module_data)
246
247     if hasattr(module, 'children') and module.children != []:
248         store.update_children(module.location, module.children)
249
250     # NOTE: It's important to use own_metadata here to avoid writing
251     # inherited metadata everywhere.
252     store.update_metadata(module.location, dict(own_metadata(module)))
253
254
255 def import_course_draft(xml_module_store, store, draft_store, course_data_path, static_content_store, source_location_namespace, target_location_namespa
256     '''
257     This will import all the content inside of the 'drafts' folder, if it exists
258     NOTE: This is not a full course import, basically in our current application only verticals (and downwards)
259     can be in draft. Therefore, we need to use slightly different call points into the import process_xml
260     as we can't simply call XMLModuleStore() constructor (like we do for importing public content)
261     '''
262     draft_dir = course_data_path + "/drafts"
263     if not os.path.exists(draft_dir):
264         return
265
266     # create a new 'System' object which will manage the importing
267     errorlog = make_error_tracker()
268     system = ImportSystem(
269         xml_module_store,
```

```
        target_location_namespace.course_id,
        draft_dir,
        {},
        errorlog.tracker,
        ParentTracker(),
        None,
    )

    # now walk the /vertical directory where each file in there will be a draft copy of the Vertical
    for dirname, dirnames, filenames in os.walk(draft_dir + "/vertical"):
        for filename in filenames:
            module_path = os.path.join(dirname, filename)
            with open(module_path) as f:
                try:
                    xml = f.read().decode('utf-8')
                    descriptor = system.process_xml(xml)

                    def _import_module(module):
                        module.location = module.location._replace(revision='draft')
                        # make sure our parent has us in its list of children
                        # this is to make sure private only verticals show up in the list of children since
                        # they would have been filtered out from the non-draft store export
                        if module.location.category == 'vertical':
                            module.location = module.location._replace(revision=None)
                            sequential_url = module.xml_attributes['parent_sequential_url']
                            index = int(module.xml_attributes['index_in_children_list'])

                            seq_location = Location(sequential_url)

                            # IMPORTANT: Be sure to update the sequential in the NEW namespace
                            seq_location = seq_location._replace(org=target_location_namespace.org,
                                                                 course=target_location_namespace.course
                                                                 )
                            sequential = store.get_item(seq_location)

                            if module.location.url() not in sequential.children:
                                sequential.children.insert(index, module.location.url())
                                store.update_children(sequential.location, sequential.children)

                            del module.xml_attributes['parent_sequential_url']
                            del module.xml_attributes['index_in_children_list']

                        import_module(module, draft_store, course_data_path, static_content_store,
                                      source_location_namespace, target_location_namespace, allow_not_found=True)
                        for child in module.get_children():
                            _import_module(child)

                    # HACK: since we are doing partial imports of drafts
                    # the vertical doesn't have the 'url-name' set in the attributes (they are normally in the parent
                    # object, aka sequential), so we have to replace the location.name with the XML filename
                    # that is part of the pack
                    fn, fileExtension = os.path.splitext(filename)
                    descriptor.location = descriptor.location._replace(name=fn)

                    _import_module(descriptor)

                except Exception, e:
                    logging.exception('There was an error. {0}'.format(unicode(e)))
                    pass


def remap_namespace(module, target_location_namespace):
    if target_location_namespace is None:
        return module

    # This looks a bit wonky as we need to also change the 'name' of the imported course to be what
    # the caller passed in
    if module.location.category != 'course':
        module.location = module.location._replace(tag=target_location_namespace.tag, org=target_location_namespace.org,
                                                   course=target_location_namespace.course)
    else:
        module.location = module.location._replace(tag=target_location_namespace.tag, org=target_location_namespace.org,
                                                   course=target_location_namespace.course, name=target_location_namespace.name)

    # then remap children pointers since they too will be re-namespaced
    if hasattr(module, 'children'):
        children_locs = module.children
        if children_locs is not None and children_locs != []:
            new_locs = []
            for child in children_locs:
                child_loc = Location(child)
                new_child_loc = child_loc._replace(tag=target_location_namespace.tag, org=target_location_namespace.org,
                                                   course=target_location_namespace.course)

                new_locs.append(new_child_loc.url())

            module.children = new_locs

    return module


def allowed_metadata_by_category(category):
```

```
    # should this be in the descriptors?!?
    return {
        'vertical': [],
        'chapter': ['start'],
        'sequential': ['due', 'format', 'start', 'graded']
    }.get(category, ['*'])


def check_module_metadata_editability(module):
    '''
    Assert that there is no metadata within a particular module that we can't support editing
    However we always allow 'display_name' and 'xml_attribtues'
    '''
    allowed = allowed_metadata_by_category(module.location.category)
    if '*' in allowed:
        # everything is allowed
        return 0

    allowed = allowed + ['xml_attributes', 'display_name']
    err_cnt = 0
    illegal_keys = set(own_metadata(module).keys()) - set(allowed)

    if len(illegal_keys) > 0:
        err_cnt = err_cnt + 1
        print ': found non-editable metadata on {0}. These metadata keys are not supported = {1}'. format(module.location.url(), illegal_keys)

    return err_cnt


def validate_no_non_editable_metadata(module_store, course_id, category):
    err_cnt = 0
    for module_loc in module_store.modules[course_id]:
        module = module_store.modules[course_id][module_loc]
        if module.location.category == category:
            err_cnt = err_cnt + check_module_metadata_editability(module)

    return err_cnt


def validate_category_hierarchy(module_store, course_id, parent_category, expected_child_category):
    err_cnt = 0

    parents = []
    # get all modules of parent_category
    for module in module_store.modules[course_id].itervalues():
        if module.location.category == parent_category:
            parents.append(module)

    for parent in parents:
        for child_loc in [Location(child) for child in parent.children]:
            if child_loc.category != expected_child_category:
                err_cnt += 1
                print 'ERROR: child {0} of parent {1} was expected to be category of {2} but was {3}'.format(
                    child_loc, parent.location, expected_child_category, child_loc.category)

    return err_cnt


def validate_data_source_path_existence(path, is_err=True, extra_msg=None):
    _cnt = 0
    if not os.path.exists(path):
        print ("{0}: Expected folder at {1}. {2}".format('ERROR' if is_err == True else 'WARNING', path, extra_msg if
               extra_msg is not None else ''))
        _cnt = 1
    return _cnt


def validate_data_source_paths(data_dir, course_dir):
    # check that there is a '/static/' directory
    course_path = data_dir / course_dir
    err_cnt = 0
    warn_cnt = 0
    err_cnt += validate_data_source_path_existence(course_path / 'static')
    warn_cnt += validate_data_source_path_existence(course_path / 'static/subs', is_err=False,
                                                    extra_msg='Video captions (if they are used) will not work unless they are static/subs.')
    return err_cnt, warn_cnt


def validate_course_policy(module_store, course_id):
    """
    Validate that the course explicitly sets values for any fields whose defaults may have changed between
    the export and the import.

    Does not add to error count as these are just warnings.
    """
    # is there a reliable way to get the module location just given the course_id?
    warn_cnt = 0
    for module in module_store.modules[course_id].itervalues():
        if module.location.category == 'course':
            if not 'rerandomize' in module._model_data:
                warn_cnt += 1
                print 'WARN: course policy does not specify value for "rerandomize" whose default is now "never". The behavior of your course may change
```

```
454             if not 'showanswer' in module._model_data:
455                 warn_cnt += 1
456                 print 'WARN: course policy does not specify value for "showanswer" whose default is now "finished". The behavior of your course may char
457     return warn_cnt
458
459
460 def perform_xlint(data_dir, course_dirs,
461                   default_class='xmodule.raw_module.RawDescriptor',
462                   load_error_modules=True):
463     err_cnt = 0
464     warn_cnt = 0
465
466     module_store = XMLModuleStore(
467         data_dir,
468         default_class=default_class,
469         course_dirs=course_dirs,
470         load_error_modules=load_error_modules
471     )
472
473     # check all data source path information
474     for course_dir in course_dirs:
475         _err_cnt, _warn_cnt = validate_data_source_paths(path(data_dir), course_dir)
476         err_cnt += _err_cnt
477         warn_cnt += _warn_cnt
478
479     # first count all errors and warnings as part of the XMLModuleStore import
480     for err_log in module_store._location_errors.itervalues():
481         for err_log_entry in err_log.errors:
482             msg = err_log_entry[0]
483             if msg.startswith('ERROR:'):
484                 err_cnt += 1
485             else:
486                 warn_cnt += 1
487
488     # then count outright all courses that failed to load at all
489     for err_log in module_store.errored_courses.itervalues():
490         for err_log_entry in err_log.errors:
491             msg = err_log_entry[0]
492             print msg
493             if msg.startswith('ERROR:'):
494                 err_cnt += 1
495             else:
496                 warn_cnt += 1
497
498     for course_id in module_store.modules.keys():
499         # constrain that courses only have 'chapter' children
500         err_cnt += validate_category_hierarchy(module_store, course_id, "course", "chapter")
501         # constrain that chapters only have 'sequentials'
502         err_cnt += validate_category_hierarchy(module_store, course_id, "chapter", "sequential")
503         # constrain that sequentials only have 'verticals'
504         err_cnt += validate_category_hierarchy(module_store, course_id, "sequential", "vertical")
505         # validate the course policy overrides any defaults which have changed over time
506         warn_cnt += validate_course_policy(module_store, course_id)
507         # don't allow metadata on verticals, since we can't edit them in studio
508         err_cnt += validate_no_non_editable_metadata(module_store, course_id, "vertical")
509         # don't allow metadata on chapters, since we can't edit them in studio
510         err_cnt += validate_no_non_editable_metadata(module_store, course_id, "chapter")
511         # don't allow metadata on sequences that we can't edit
512         err_cnt += validate_no_non_editable_metadata(module_store, course_id, "sequential")
513
514         # check for a presence of a course marketing video
515         location_elements = course_id.split('/')
516         if Location(['i4x', location_elements[0], location_elements[1], 'about', 'video', None]) not in module_store.modules[course_id]:
517             print "WARN: Missing course marketing video. It is recommended that every course have a marketing video."
518             warn_cnt += 1
519
520     print "\n\n----------------------------------------\nVALIDATION SUMMARY: {0} Errors   {1} Warnings\n".format(err_cnt, warn_cnt)
521
522     if err_cnt > 0:
523         print "This course is not suitable for importing. Please fix courseware according to specifications before importing."
524     elif warn_cnt > 0:
525         print "This course can be imported, but some errors may occur during the run of the course. It is recommend that you fix your courseware before
526     else:
527         print "This course can be imported successfully."
528
529     return err_cnt
```