Moodle 1.9 to 2.0 Restore

General Information

- Restore controller detects that the backup file is not Moodle 2.0 format, this fires off the conversion. The conversion process will transform the unknown format into a Moodle 2.0 format that can then be restored.
- From the start, unit testing must be taken into consideration. Having solid unit testing in place will make it possible for core developers to maintain as they change DB schema, etc.
- In the end, everything needs to write standard backup code once the information is extracted from the format what we are converting from. At the very least, we should be able to provide writers that handle generic services with existing API.
- Can we use progressive_parser class to parse M1.9 XML?

Base Convert API

The overarching API design is based off of the already two existing systems: backup and restore. This new API would introduce a third system called convert to handle converting various sources into Moodle 2.0 backups that can then be restored into Moodle courses. The following is a list of files that will most likely need to be generated or modified. All paths are relative to **backup/**


Unit testing still needs to be added where ever possible.

- converter/ - holds converter plugin files. Should be 100% pluggable (EG: no other code modifications to get them up and running).
  - moodle1.php - would extend plan_converter class. This generates the convert plan for 1.9 backups to 2.0 backups.
- controller/
  - restore_controller.class.php - modify this to support converting from different formats. Primary focus is the convert function. Would setup a class found in backup/converter/ and execute it.
- moodle2/
  - convert_activity_task.class.php - base class for converting activities.
  - convert_block_task.class.php - base class for converting blocks.
  - convert_plugin_task.class.php - base class for converting other plugins.

- convert_subplugin_task.class.php - base class for converting sub-plugins.
- convert_qtype_task.class.php - base class for converting questions.
- convert_course_task.class.php - base class for converting course information.
- convert_section_task.class.php - base class for converting course section information.
- convert_final_task.class.php - final conversion tasks. Probably things like writing out the final files.xml and etc.
- convert_stepslib.php - library of convert steps. These could potentially be used for generating fake contexts and handling of converting files instead of using a convert helper.

- util/
  - converter/
    - base_converter.php - abstract class that has the very minimum needs of a converter class. This is used to create plugins by dropping in a class file into backup/converter/. If a converter were to extend this, it could basically do whatever it wanted to get the conversion process done.
    - plan_converter.php - abstract class focused on plan/task/step based conversion.
  - factories/
    - convert_factory.class.php - could use this class to load/setup converter plugins found in backup/converter/.
  - helper/
    - convert_helper.class.php - could use this for loading all converters. Maybe also to see if the current restore format matches one of the converters.
    - *convert_X_helper.class.php* - throughout the conversion, we will need to do things like create fake contexts, convert files, etc. These might be best handled by a helper. An alternative would be to add steps in the moodle2/convert_stepslib.php.
  - includes/
    - restore_includes.php - modify this to include any classes we need in order to bootstrap conversion.
    - convert_includes.php - rest of the includes needed to perform conversions.
  - plan/
    - convert_plan.class.php - used by converters.
    - convert_step.class.php - used in stepslib type files.
    - convert_task.class.php - Extended by convert task classes.
  - structure/
    - convert_nested_element.class.php - basically this is like backup_nested_element.class.php but instead of working with tables, we are going to work with some other data source, probably arrays. This will be used by the actual convert steps.
    - *unknown.class.php* - may need to make other ones like

convert_nested_element.class.php.

- xml/
  - parser/
    - progressive_parser.class.php - *might be able to use this to read 1.9 XML*

Plugin Convert API

Each plugin that supports a particular conversion format would have to implement its own tasks and steps. An example converter name would be *moodle1* for Moodle 1.9 backups. A plugin would have the following files relative to **path/to/plugin/backup/CONVERTER/**

> Not all converters need to require this plugin structure as the converter may handle everything "in house" by setting up all of it's own plans/tasks/steps or by a completely different method.

- convert_PLUGINNAME_PLUGINTYPE_task.class.php - should define the conversion steps to take for the plugin. This would extend backup_PLUGINTYPE_task class.
- convert_PLUGINNAME_stepslib.php - define the actual steps. This would extend backup_PLUGINTYPE_structure_step class. This will make use of things like convert_nested_element.class.php.
- simpletest/
  - test_convert_PLUGINNAME_PLUGINTYPE_task.php
  - test_convert_PLUGINNAME_stepslib.php
  - files/
    - *unkown.unknown* - perhaps store files in here to help assist with unit testing. Maybe need XML files or fake course files. Might be able to use something more general like a fake 1.9 backup located somewhere under backup/ directory.

File Convert API

There are three sources for files that we must deal with: course files, moddata and files referenced in text. Course files should be added to the "legacy" files area, by doing so, this should take care of the files referenced in text fields. The moddata directory has to be handled on a per plugin basis. The following API is needed to accomplish this and all paths are relative to **backup/**

backup_ids_temp is created via backup_controller_dbops::create_backup_ids_temp_table() which copies the structure found in backup_files_template and sets the backupid to default to the current backupid (very helpful!). The table will be created/dropped by the converter class.

- util/helper/convert_helper.class.php - houses general convert helper methods. Responsibilities include:
  1. A method to set a inforef for a specific context
     - All possible inforef names are in backup_helper::get_inforef_itemnames()
     - Probably backup_ids_temp insert is like: itemname = 'file/user/etc', itemid = ID of the file/user/etc
  2. A method to query inforefs for a specific context
  3. Context ID generation
     - Probably backup_ids_temp insert like: itemname = 'context, itemid = CMID/BLOCK_INSTANCEID/ETC, info = component. The context ID is the ID generated from the insert into backup_ids_temp.
- util/helper/convert_file_manager.class.php - this would provide methods for converting files. Responsibilities include:
  1. There would be a method for converting all course files, these will be stored in the "legacy" files area (sort of mimic upgrade_migrate_files_course).
  2. There should be a general function that operates on a single file and it would do the following: copy the file properly to files/ directory (would hash the file name), create content hash, serialize all data and insert it into backup_ids_temp table and register the file inforef. Modules will make use of the general function to convert their files.
     - Probably backup_ids_temp insert is like: itemname = 'file', itemid = FILE_ID, info = all file information. FILE_ID is probably artificially generated.
  3. Provide a method to query information to generate files.xml
- moodle2/convert_stepslib.php - contains step classes. The following should be added:
  1. Create a step to convert course files by using convert_file_manager.class.php.
  2. Create a step that reads from the backup_ids_temp table to generate the files.xml (probably have to extend convert_structured_step and make use of convert_file_manager.class.php to pull the data down).
  3. Create a step to generate a inforef.xml file for a specific context (probably have to extend convert_structured_step and make use of convert_helper.class.php to pull the data down). After the generation, call step move_inforef_annotations_to_final or something equivalent.