Moodle 1.9 to 2.0 Restore

General Information

- Restore controller detects that the backup file is FORMAT_MOODLE1 (EG: Moodle 1.9), this fires off the conversion.
- The conversion API should probably follow the same API structure as Moodle 2.0 backup/restore API. So, we would need the following:
  - Convert plan
  - Convert steps
  - Convert tasks
- Would be nice if the conversion API could support different conversion formats. This would be handled by what Eloy dubbed, "conversion dispatcher" - sort of like backup/restore controllers.
  - Dispatchers would be responsible for the overall execution of the conversion, EG: loading the plan, steps and tasks that are specific to converting the dispatchers format.
  - Dispatchers are format specific - EG: we would have a dispatcher for Moodle 1.9 format.
  - Probably would be nice to allow dispatchers to identify if a backup matches their format.
- The conversion process will support plugins by allowing plugins to define their own steps and tasks just like backup/restore.
  - Plugins will store their files in /path/to/plugin/backup/FORMAT/
  - The files will be prefixed with "convert_"
  - Example: mod/quiz/backup/moodle1/convert_quiz_activity_task.class.php
- From the start, unit testing must be taken into consideration.
  - Ideas to assist, inside of /path/to/plugin/backup/FORMAT/simpletest/files we have a set of XML files and perhaps fake course files that need to be converted. Run tests against these.
  - I think that 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.
  - **Can we use backup classes to accomplish this?** Looks like the answer is **no** without major refactoring by abstracting the data source so it can be swapped from the database to anything. Basically the idea being writing a M2.0 backup but you are swapping out the data source from M2.0 database tables to a backup file format like M1.9.
  - At the very least, we should be able to provide writers that handle generic stuffs

for plugins 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/**

- controller/
  - restore_controller.class.php - modify this to support converting to different formats. Primary focus is the convert function.
  - convert_controller.class.php - would use *convert_plan_builder_FORMAT.class.php* type classes to generate a convert plan and execute it.
- moodle2/
  - convert_plan_builder_moodle1.class.php - this generates the convert plan for 1.9 backups to 2.0 backups. This would also handle factory type methods for converting core parts like courses, sections, etc.
  - *convert_plan_builder_FORMAT.class.php* - other format converters could created.
  - 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/
  - factories/
    - convert_factory.class.php - could use this class to load/setup convert dispatchers.
  - helper/
    - convert_helper.class.php - could use this for loading all convert dispatchers. 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 convert dispatchers.
    - 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 format name would be *moodle1* for Moodle 1.9 backups. A plugin would have the following files relative to **path/to/plugin/backup/FORMAT/**

- 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.