Agenda for callbacks / namespaces meeting

Links:
[MDL-44078: Proposal: API standard in Moodle that uses autoloading]
[Forum post: Interface to allow plugins interact with core (hooks)]
[Forum post: Plugins depending on other plugins]
[Draft of Namespaces documentation]

1. Do we allow plugins to implement API defined in another plugin?

Option 1: YES, all the time
Option 2: YES, but only for dependent plugins or subplugins
Option 3: YES, but for intercommunication of add-ons only (via callbacks or hooks, as agreed in #4 and #5 below), never in Moodle standard plugins
Option 4: NO, never

1-1 with dependency - ALLOWED in standard plugins, addons
1-1 with no dependency - NOT ALLOWED in standard plugins, ALLOWED addons
1-many (one plugin calling all installed plugins- no core api) - NOT ALLOWED in standard plugins, ALLOWED addons
many-1 (call the plugin that implements XX ie locking - no core api) - NOT ALLOWED in standard plugins, ALLOWED in addons
many-many (logging? - no core api) NOT ALLOWED in standard plugins, ALLOWED in addons

David's note: IIRC, it was also said that standard plugins can call another plugin's functionality if the dependency is explicitly declared in version.php

2. Decide on API names for existing or proposed plugin-plugin communication. If [#1=YES] some items may be left without core API

- tool_reportbuilder (reporting?)
- block_course_overview (asks modules to provide information for the block) (reporting?)
- block_recent_activity (asks modules to provide information for the block) (reporting?). This is combined with very similar callback in core for "pseudo-report" course/recent.php
- ~~report_participation (asks modules what log entries they consider to be "participating", deprecated in 2.7 with introduction of event->edulevel)~~
- report_outline (again, asks modules to provide info)
- tool_log (allows reports to notify if they require access to logs)
- qtype_cloze (allows other sub-qtypes to be added to cloze. https://tracker.moodle.org/browse/MDL-6371)

3. Where can plugins store classes implementing APIs defined by core or other plugins (even if [#1=NO], there are still exceptions)?

Option 1: core - <plugindir>/classes/<coreapinamewithoutunderscores>/xxx.php, plugins - locallib.php, lib.php, classes/local/xxx.php, classes/xxx.php and other existing implementations

Option 2: core - <plugindir>/classes/<coreapinamewithoutunderscores>/xxx.php, plugins - <plugindir>/classes/<another_plugin_name_with_underscores>/xxx.php (breaks current namespace rules).

Current rule is that they should go anywhere in /classes/local/, we can recommend that the go into /classes/local/<another_plugin_name>/xxx.php - but this is not enforced anywhere.

The whole point of the current rule is to stop name clashes. Plugins can do whatever they like in /local, and that will never clash with anyhing required by core. Therefore, following that logic, Option2 is better, I think. (Tim). In practice, however, it is pretty much the same. Frankensytle component names are unlikely to clash with other names.

4. How can core API or plugin call related function implemented in one or several plugins?

Option 1: total mess of callbacks implementation and hidden config settings as it is now

Option 2: callbacks but only through functions get_plugin_list_with_function() and component_callback() (or new similar functions in core_component). If we go this way we can improve by adding caching there and allow looking for function in multiple plugin types (missing now because of performance considerations)

Option 3: hooks for the new code, replacing couple of important existing callbacks

Option 4: hooks with the replacement of all existing callbacks

Option 5: Registration of functions implementing an API in db/XXX.php files (example for Events API: db/events.php). Hooks spec is an example of this.

Please can we stop using the word 'Hooks' until we have a short, clear definition. What is the differece between a 'hook' and "callbacks but only through functions get_plugin_list_with_function() and component_callback() (or new similar functions in core_component)"? I know there is MDL-44078 - long bug, which links to http://docs.moodle.org/dev/Hooks_spec - long wiki page. I am looking for the one-paragraph version. This is a key proposal that all developers will need ot undrestand. I you can explain the fundamental point in 1 paragraph, it is too complex.

5. How can core API or plugins search for the classes in plugins?

Option 1: core API / plugin looks for classes in an expected location (see #3). Probably will need some core functions that lookup for classes. See proposed solution in MDL-46155 (apparently it's very easy because the list of existing classes is already cached)

MDL-40457 is a recent example of a long-standing approach, we should decide on this option before it's integrated. Summary of MDL-40457 so you don't have to read the branch and all the comments: New undocumented secret config setting $CFG->quizquestionbankcolumns which defines the autoloaded class name of a class extending an abstract class provided by the question bank.

Option 2: Registration of classes implementing an API in db/XXX.php files (example for scheduled tasks API: db/tasks.php).

6. Nominate people to

- Publish Namespaces guidelines in dev docs (Damo will start and work with David)
- Announce the decisions of this meeting on Forum (Damo)