Addons Cleanup Original Requirements

This document is now out of date and is kept to refer to information which could still be useful during development. Please see Addons Cleanup instead.

Background

In preparation for Tiki20, there needs to be a cleanup of Addons (doc site: Addons). The goal is to merge all useful functionality in Addons into Themes and Profiles, and then remove the Addons feature itself. The rationale is to not provide too many ways of Extending Tiki, especially ways that are half-baked or potentially confusing. Mods are already deprecated. Making Themes the main way to customize Tiki makes conceptual sense and also much of Addons is really "Profiles 2.0" so any enhancements should be merged into one of these 2.

Addons cover a number of functionality and so it is unhelpful to consider it as one monolithic thing. It is best to break it down. Below is a list of functionality, where the code is, and what to do with each.

You can generally find Addons code by grepping for "TikiAddons". The core code is in lib/core/TikiAddons.

Ability to install a bundle of profiles altogether

This should simply belong in Profiles.

The way it works is to put your bundle of profiles in a Local Profiles. And then instead of having the current "php console.php addon:install" we have a replacement "php console.php profile:installbundle" which does the same thing.

The current Addons code looks within a certain specific path (/addons/addon_name) but the replacement needs to be refactored to accept any Local Profiles "repo" location.

Right now it is very prescriptive what the form of the package name has to be (see Anatomy of a Tiki Addon) and to preserve backward compatibility (e.g. with Organic Groups) and for the good reasons from before, we will continue to use these package naming rules for Profile Bundles. But unlike the Addons situation where the pathname matches the package name, in the replacement Profile Bundles the pathname can be different.

The documentation will hence forth refer to these as Profile Bundles instead of Addons. Instead of tikiaddon.json, there will be a replacement profilebundle.json which will be placed in the Local Profile "repo" folder to indicate that it is a Profile Bundle.

Ability to version Profile Bundles, manage installation and upgrading

Right now, most relevant configuration information is stored in tikiaddon.json. Upgrading configuration information is in upgrade.json. Uninstallation configuration information is in uninstall.json. Everytime an

Addon is installed, upgraded or uninstalled, it updates what is installed in the db table tiki_addon_profiles.

As mentioned, tikiaddon.json should instead be profilebundle.json and placed in the Local Profiles folder where the profiles are. upgrade.json and uninstall.json will similarly be moved there. Instead of "php console.php addon:upgrade" we will have a replacement "php console.php profile:upgradebundle" which does the same thing. As part of refactoring, the db table tiki_addon_profiles can be renamed tiki_profile_bundles and the field "addon" can be renamed "package".

Now that the Local Profiles can be in any "repo" path, the code will also need to be refactored to look in the profilebundle.json file for the canonical package name instead of deriving it from the path within the addons folder. As already mentioned above, the path of the repo can be anything.

The additional metadata in tikiaddon.json in the form of the URL (of the addon) can be left as optional and is harmless. It's not currently used by the system for any purpose afaik and is useful as human readable data.

The upgrading info in upgrade.json contains information what versions you are allowed to upgrade from to the new version, and records what versions you have installed. It also contains instructions to remove items if that is part of the upgrade.

Similarly, the uninstalling info in uninstall.json contains info what items to remove when uninstalling. Instead of "php console.php addon:remove", it should be replaced by "php console.php profile:uninstallbundle" which does the same thing.

Future UX enhancements (to be considered)

Not immediately urgent but:

- The profile installer should be improved to be able to handle Profile Bundles (this might be useful for the UX to install Organic Groups) see need for configuration UI below
- There should be a way to install Profile Bundles that are on a remote server vs only Local Profiles

Ability to lookup the object ID from the profile registry for use in custom smarty templates

The purpose is to avoid hardcoded IDs in custom smarty templates that refer to objects created via profiles, so that when the same profiles are installed on another Tiki the same templates will work without modification.

This functionality need to be made more generic, i.e. work with any Local Profiles repo vs addons (they currently lookup within the addons/addon_name local "repo"). See lib/smarty_tiki/function.addonobjectid.php. Additionally, it would be good if this function can be made generic to be used with any profile and not just those installed via Local Profiles regardless of whether it is a Profile Bundle (i.e. those with a profilebundle.json file) or not. The name can be changed to profileobjectid from addonobjectid after refactoring.

Ability to manage dependencies on Tiki version and other addons when installing addons

This is currently configured in tikiaddon.json.

```
"tiki": ["15"],
"depends": [
],
```

This functionality can be removed. It is pretty basic (not sure if working properly at all) right now and dependencies can be taken care of manually and so it isn't valuable functionality at the moment. The idea was that in an environment where there is a kind of "addons marketplace" this kind of functionality might be useful but we are very far away from such a thing.

Ability to have templates, css, img within addons

This functionality can be removed. The alternative is simply to put all these in a Theme. See lib/setup/theme.php. The disadvantage of course is that you can only use one theme at any one time but you can have multiple addons running, so it would lose this capability but it might be possible to allow "theme addons" or something that would achieve the same thing but that is not urgent.

Ability to have lang files within addons

As far as I can tell this was not developed (I can't find code handling this) although the Anatomy of a Tiki Addon shows a folder for lang files. Anyway, this functionality can be removed. The alternative is simply to use custom.php but the problem there is that you can only have one custom.php but maybe in the future it might be possible to allow translations to be added to themes via "theme addons" or "lang addons" or something that would achieve the same result, but that is not urgent.

Ability to add custom prefs

This functionality can be moved to Themes. The disadvantage of course is that you can only use one theme at any one time but you can have multiple addons running, so it would lose this capability but it might be possible to allow "theme addons" or something that would achieve the same thing but that is not urgent.

The Addons control panel for the setting of these prefs can be moved to the Look and Feel control panel.

Ability to add custom PHP code libs

This functionality should be added to Themes. Basically move everything there and the information can be stored in something like a tikitheme.json info file instead of the tikiaddon.json file.

The disadvantage of course is that you can only use one theme at any one time but you can have multiple addons running, so it would lose this capability but it might be possible to allow "theme addons" or something that would achieve the same thing but that is not urgent.

Namespace handling within Addons

There is some PHP namespace (http://php.net/manual/en/language.namespaces.php) management functionality used so that when you add PHP code you can't clobber any Tiki core variables. Right now it is very prescriptive how the namespacing is done as the namespaces are based on the Addon name. Perhaps we can simply remove the code that looks up namespaces based on the Addon name and simply document it as a best practice to use PHP namespaces and allow any namespace to be used?

Ability to have own isolated Smarty instance

This is configured in the tikiaddon.json right now:

```
"smarty": true,
```

What this does is create a separate Smarty instance to be used for each addon so that custom code does not access the Smarty instance that is used by Tiki core. As Smarty does not support the equivalent of PHP namespaces (http://php.net/manual/en/language.namespaces.php) right now (afaik), it is possible for smarty variables assigned in addons to clobber Smarty variables in Tiki core in the absence of this feature. That is why this is introduced.

However, if we are removing Addons and moving the ability for custom code to be in Themes, do we retain this functionality? It is probably a good thing to create more security from custom code, but it should be default on and so a setting probably isn't needed. The instantiation of this Smarty object can be found in lib/core/Tiki/TikiAddons.php. You will see that it also picks up some common useful variables from the Tiki core Smarty instance into the isolated instance, such as $prefs.

Ability to disable Addons via a pref

Right now, addons can be disabled via a pref but if all the custom prefs/custom code functionality is moved to Themes then do we preserve this functionality? The reason for this functionality was more from a security perspective, to make it easy to disable the entire addon through the UI in case there are security vulnerabilities. Perhaps the functionality can be retained within themes but apply only to the code portion? i.e. add a checkbox pref to enable code within the currently installed theme which can be placed in the Look and Feel control panel instead?

Ability to execute that custom code through a wikiplugin

See PluginAddon and lib/smarty_tiki/block.addonview.php. The idea was originally to provide a "standard" way to access the code that was added via Addons for reasons of security management. This standard way is called Addon Views.

If Themes can already have wikiplugins added (to confirm if true) then perhaps it is not necessary to have this "standard" way to access custom code? I mean, it is always possible to write a custom wikiplugin and just put it in a theme (to be confirmed). If so, then can consider removing PluginAddon and lib/smarty_tiki/block.addonview.php? But if it is considered a good thing to have a standard interface, then it needs to be refactored to lookup code in themes instead of addons; and also made to be the only way to access PHP code added through a theme otherwise the security weakness would still exist and it would not achieve much. If there is a new standard way to access custom PHP code, then we should also try and think of a better name than "Views".

Ability to add custom Events to trigger on certain actions

For example, if a certain tracker is saved you can run your own custom code. This functionality should remain possible via Themes. See lib/setup/events.php for where the trigger is. The core code is in lib/core/TikiAddons/Api/Events.php.

The configuration is currently stored in tikiaddon.json. This can be moved to tikitheme.json.
Example:

```
"api": {
    "eventmap": [
                {
                  "event" : "tiki.trackeritem.save",
                  "lib" : "tikitheme.themename.libname",
                  "function" : "sendInternalSystemMessage",
                  "params" : {
                   }

                }
    ]
}
```

Ability to add custom Search sources

Basically you can add your own custom code that adds stuff to be indexed for searching as a new "Content Source". This functionality should remain possible via Themes. The alternative is to manually add files to lib/core/Search/ContentSource but it is better not to mess in the Tiki core files tree.

The code that handles this is in lib/core/TikiAddons/Api/Search.php. The call to it is in lib/core/Search/Indexer.php. Instead of the the configuration being in tikiaddon.json it can now be in tikitheme.json. For example:

```
"api": {
    "search": {
          "addonSources": [{
                "location": "lib/CustomSource.php",
                "class": "\\customnamespace\\CustomSource"
          }]
        }
}
```

Ability to create Groups that have a token name and store the information in a tracker

Groups can be created through Profiles, and also PluginDataChannel is used to enable Organic Groups functionality.

The reasons why we want to use a token name instead of actual names of groups are because:

1. To prevent clobbering of Group names already existing on the system when you install profiles.
2. To allow Organic Group type functionality.

For 1 above, it is something we can ignore for now, as once we remove Addons where the Group token name is the addon name, anyone who is installing profiles would just have to make sure they don't use Group names that are the same (i.e. manually avoid group name collision).

But for 2 above, it is tied to the Organic Group functionality which is something that we want to have.

In the context of Addons, the solution was to make the names of groups token based on the name of the addon, e.g. groups are addonname_1, addonname_2 and so on, and the group name (and any additional info if necessary) is in fact stored in tracker (the number after the token is the item ID of the tracker item that corresponds to that group).

To get back the actual group name or the tracker item ID of the group you are referring to (from the token name), there are lib/smarty_tiki/modifier.addongroupname.php and lib/smarty_tiki/modifier.addonitemid.php. After refactoring, these modifiers could be renamed "namefromtoken" or "itemidfromtoken" or something like that to more accurately represent what they do.

Organic Group functionality

With regards to Organic Group functionality, it is best to simply put in in core. We can commit the profiles that install the Organic Group functionality into a folder "tikiprofiles/organicgrp". The token can be derived in a similar way as the current code works, but instead from "addonname_", it should be derived from the package name that is set inside profilebundle.json. So if we set it to "tiki/organicgrp", then the token will be "tiki_organicgrp_".

Because it is in core, if people want to customize or extend Organic Groups, they can do it via Themes through templates and CSS, or if they want to modify the profiles that install the functionality, then they have to make a copy of "tikiprofiles/organicgrp" to say "profiles/organicgrp" and then modify and use that new one. This is basically what they would have had to do if it was an Addon anyway.

Impact of Roles? Even though it will probably need major refactoring when Roles are implemented and the migration path might involve some kind of migration script, it seems better to simply add it now now and maybe mark as experimental?

So we will need to add into core a minimal Organic Groups feature that creates groups with:

1. A forum per group
2. A file gallery per group
3. A members listing
4. A group home page
5. A group management page (for adding/removing/approving members)

Need for configuration UI

So all the existing capability to enable Organic Groups should be retained but instead of looking for the configuration information in the addons folder (in the tikiaddon.json file), we simply have it look in the replacement profilebundle.json file.

> Note that we could consider a Generic Profiles Bundle installation capability since we need this for Organic Groups anyway

However, there also needs to be a UI to be able to configure Organic Groups otherwise the feature will never be used. So maybe some info needs to be stored in the database (for configurability). Right now the info we have in tikiaddon.json is:

```
"api": {
          "group": {
                "tracker": "trk_og",
                "public_catroot": "cat_public_grp",
                "private_catroot": "cat_private_grp",
                "managementpage": "GroupManage",
                "homepage": "GroupHome"
          },
}
```

In the most basic version, the tracker, categories etc. will be created by the Local Profiles that are in Tiki core so the defaults for the above can be left in profilerepo.json as the code is already setup to look in the configuration json file. However, it would be necessary to allow changing of:

- the managementpage and homepage settings as these are wiki page names.
- location of Profile Bundle repo for the setup of the Organic Groups feature (default to the one included in Tiki core as a Local Profile e.g. tikiprofiles/organicgrp, but you can point to your own customized one)

There should also be a button to activate Organic groups (i.e. execute the equivalent of "php console.php profiles:installbundle"). Once activated, the configuration is frozen, and there should be an upgrade button if the version has changed in the event of a Tiki upgrade (i.e. execute the equivalent of "php console.php profiles:upgradebundle"), and a uninstall button ((i.e. execute the equivalent of "php console.php profiles:uninstallbundle") with warnings that data will be lost etc.

Is it possible to have more than one instance of

Organic Groups in a Tiki?

Additionally, it would be nice to be able to setup multiple organic group instances on the same Tiki for different use cases. Theoretically this already works but you need the configuration UI to be able to support creation of multiple configurations, and for each configuration you will need to specify a different Local Profiles repo (you'd need to make a clone of the one provided by Tiki and modify if needed).

The configuration UI will also need the additional parameter configuration name (for reference).
You'd also need to modify the profilebundle.json in your copy to change the Package Name.

Although this should not be too difficult, as the code already allows different instances with different token names (it is part of the Addons design), to keep things simple nonetheless, it is recommended that the 1st UI version should not allow multiple configurations, i.e. you can only have one instance of Organic Groups on each Tiki unless if you know enough you can manually clone a profile, change the json and and run it.

Ability to have File Galleries as part of Organic Groups

i.e. one file gallery per group above. In the code it is a separate API in lib/core/TikIAddons/Api/FileGallery which stores additional code specific to the File Gallery per group functionality of Organic Groups.

Ability to add a navbar for each organic group

This is a secondary navbar that sits within the content frame and provides links to the different Organic Group features (e.g. Forum, Files).

The challenge with such a navbar is that the links need to be dynamic (have the correct organic group ID set) depending on which organic group you are viewing. E.g. if the Group homepage is "GroupHome", the homepage for the group "tiki_organicgrp_1" is "GroupHome?organicgroup=1". Therefore the logic to handle this needs to be somewhere and now it is in lib/core/TikiAddons/Api/NavBar.php. See lib/smarty_tiki/modifier.addonnavbar.php.

Currently, It loads a custom navbar template which the addon developer is to set in tikiaddon.json. Since Organic Groups will be in core we will have to refactor it to load a template e.g. tiki-ognavbar.tpl (need to be committed in core) instead.

Configuration in tikiaddon.json which we can just change in the new profilebundle.json that is committed in Tiki core tikiprofiles/organicgrp.

```
"api": {
    "navbar": {
                "tpl": "tiki-ognavbar.tpl"
          },
}
```