---+ Package `TWiki::Users` This package provides services for the lookup and manipulation of login and wiki names of users, and their authentication.

It is a Facade that presents a common interface to the User Mapping and Password modules. The rest of the core should **only** use the methods of this package, and should **never** call the mapping or password managers directly.

TWiki uses the concept of a *login name* which is used to authenticate a user. A login name maps to a *wiki name* that is used to identify the user for display. Each login name is unique to a single user, though several login names may map to the same wiki name.

Using this module (and the associated plug-in user mapper) TWiki supports the concept of *groups*. Groups are sets of login names that are treated equally for the purposes of access control. Group names do not have to be wiki names, though it is helpful for display if they are.

Internally in the code TWiki uses something referred to as a _canonical user id_ or just *user id*. The user id is also used externally to uniquely identify the user when (for example) recording topic histories. The user id is **usually** just the login name, but it doesn't need to be. It just has to be a unique 7-bit alphanumeric and underscore string that can be mapped to/from login and wiki names by the user mapper.

The canonical user id should **never** be seen by a user. On the other hand, core code should never use anything **but** a canonical user id to refer to a user.

Terminology

- A **login name** is the name used to log in to TWiki. Each login name is assumed to be unique to a human. The Password module is responsible for authenticating and manipulating login names.
- A **canonical user id** is an internal TWiki representation of a user. Each canonical user id maps 1:1 to a login name.
- A **wikiname** is how a user is displayed. Many user ids may map to a single wikiname. The user mapping module is responsible for mapping the user id to a wikiname.
- A **group id** represents a group of users and other groups. The user mapping module is responsible for mapping from a group id to a list of canonical user ids for the users in that group.
- An **email** is an email address asscoiated with a **login name**. A single login name may have many emails.

NOTE:

- wherever the code references $cUID, its a canonical_id
- wherever the code references $group, its a group_name
- $name may be a group or a cUID

ClassMethod new `($session)`

Construct the user management object that is the facade to the BaseUserMapping and the user mapping chosen in the configuration.

ObjectMethod finish `()`

Break circular references.

ObjectMethod loginTemplateName `() -> templateFile`

allows UserMappings to come with customised login screens - that should preffereably only over-ride the UI function

ObjectMethod supportsRegistration `() -> boolean`

#return 1 if the main UserMapper supports registration (ie can create new users)

ObjectMethod initialiseUser `($login) -> $cUID`

Given a login (which must have been authenticated) determine the cUID that corresponds to that user. This method is used from TWiki.pm to map the $REMOTE_USER to a cUID.

randomPassword()

Static function that returns a random password. This function is not used in this module; it is provided as a service for other modules, such as custom mappers and registration modules.

ObjectMethod addUser `($login,$wikiname,$password,$emails,$mcp) -> $cUID`

- `$login` - user login name. If `undef`, `$wikiname` will be used as the login name.
- `$wikiname` - user wikiname. If `undef`, the user mapper will be asked to provide it.
- `$password` - password. If undef, a password will be generated.
- `$mcp` - must change password flag.

Add a new TWiki user identity, returning the canonical user id for the new user. Used ONLY for user registration.

The user is added to the password system (if there is one, and if it accepts changes). If the user already exists in the password system, then the password is checked and an exception thrown if it doesn't match. If there is no existing user, and no password is given, a random password is generated.

$login can be undef; $wikiname must always have a value.

The return value is the canonical user id that is used by TWiki to identify the user.

StaticMethod mapLogin2cUID `($login) -> $cUID`

This function maps an arbitrary string into a valid cUID. The transformation is reversible, but the function is not idempotent (a cUID passed to this function will NOT be returned unchanged). The generated cUID will be unique for the given login name.

This static function is designed to be called from custom user mappers that support 1:1 login-to-cUID mappings.

ObjectMethod getCanonicalUserID `($identifier) -> $cUID`

Works out the TWiki canonical user identifier for the user who either (1) logs in with the login name $identifier or (2) has the wikiname $identifier.

The canonical user ID is an alphanumeric string that is unique to the login name, and can be mapped back to a login name and the corresponding wiki name using the methods of this class.

Note that if the login name to wiki name mapping is not 1:1, this method will map a wikiname to one of the login names that corresponds to the wiki name, but there is no guarantee which one.

Returns undef if the user does not exist.

ObjectMethod findUserByWikiName `($wn) -> \@users`

- `$wn` - wikiname to look up

Return a list of canonical user names for the users that have this wikiname. Since a single wikiname might be used by multiple login ids, we need a list.

If $wn is the name of a group, the group will **not** be expanded.

ObjectMethod findUserByEmail `($email) -> \@users`

- `$email` - email address to look up

Return a list of canonical user names for the users that have this email registered with the user mapping managers.

ObjectMethod getEmails `($name) -> @emailAddress`

If $name is a cUID, return their email addresses. If it is a group, return the addresses of everyone in the group.

The password manager and user mapping manager are both consulted for emails for each user (where they are actually found is implementation defined).

Duplicates are removed from the list.

ObjectMethod setEmails `($cUID,@emails)`

Set the email address(es) for the given user. The password manager is tried first, and if it doesn't want to know the user mapping manager is tried.

ObjectMethod getMustChangePassword `($cUID) -> $flag`

Returns 1 if the $cUID must change the password, else 0. Returns undef if $cUID not found.

ObjectMethod getUserData `($cUID) -> $dataRef`

Return a reference to an array of hashes with user data, used to manage users. Each item is a hash with:

- `{name}` - name of field, such as "email"
- `{title}` - title of field, such as "E-mail"
- `{value}` - value of field, such as "jimmy@example.com"
- `{type}` - type of field: `text`, `password`, `checkbox`, `label`
- `{size}` - size of field, such as `40`
- `{note}` - comment note, if any

User management forms can be build dynamically from this data structure. Each password manager may return a different set of fields.

ObjectMethod setUserData `($cUID,$dataRef)`

Set the user data of a user. Same array of hashes as getUserData is assumed, although only `{name}` and `{value}` are used.

ObjectMethod isAdmin `($cUID) -> $boolean`

True if the user is an admin

- is $TWiki::cfg{SuperAdminGroup}
- is a member of the $TWiki::cfg{SuperAdminGroup}

ObjectMethod isInList `($cUID,$list) -> $boolean`

Return true if $cUID is in a list of user **wikinames**, **logins** and group ids.

The list may contain the conventional web specifiers (which are ignored).

ObjectMethod getLoginName `($cUID) -> $login`

Get the login name of a user. Returns undef if the user is not known.

ObjectMethod getWikiName `($cUID) -> $wikiName`

Get the wikiname to display for a canonical user identifier.

Can return undef if the user is not in the mapping system (or the special case from initialiseUser)

ObjectMethod webDotWikiName `($cUID) -> $webDotWiki`

Return the fully qualified wikiname of the user

ObjectMethod userExists `($cUID) -> $boolean`

Determine if the user already exists or not. A user exists if they are known to to the user mapper.

ObjectMethod eachUser `() -> TWiki::IteratorofcUIDs`

Get an iterator over the list of all the registered users **not** including groups.

list of canonical_ids ???

Use it as follows:

```
    my $iterator = $umm->eachUser();
    while ($iterator->hasNext()) {
        my $user = $iterator->next();
        ...
    }
```

ObjectMethod eachGroup `() -> TWiki::ListIteratorofgroupnames`

Get an iterator over the list of all the groups.

ObjectMethod eachGroupMember `($group) -> $iterator`

Return a iterator of user ids that are members of this group. Should only be called on groups.

Note that groups may be defined recursively, so a group may contain other groups. This method should **only** return users i.e. all contained groups should be fully expanded.

ObjectMethod isGroup `($name) -> boolean`

Establish if a $name refers to a group or not. If $name is not a group name it will probably be a canonical user id, though that should not be assumed.

ObjectMethod isInGroup `($cUID,$group) -> $boolean`

Test if the user identified by $cUID is in the given group.

ObjectMethod eachMembership `($cUID) -> $iterator`

Return an iterator over the groups that $cUID is a member of.

ObjectMethod checkLogin `($login,$passwordU) -> $boolean`

Finds if the password is valid for the given user. This method is called using the login name rather than the $cUID so that it can be called with a user who can be authenticated, but may not be mappable to a cUID (yet).

Returns 1 on success, undef on failure.

TODO: add special check for BaseMapping admin user's login, and if its there (and we're in sudo_context?) use that..

ObjectMethod setPassword `($cUID,$newPassU,$oldPassU,$mcp) -> $boolean`

If the $oldPassU matches matches the user's password, then it will replace it with $newPassU.

If $oldPassU is not correct and not 1, will return 0.

If $oldPassU is 1, will force the change irrespective of the existing password, adding the user if necessary.

Otherwise returns 1 on success, undef on failure.

$mcp is the "must change password flag" that forces the user to change the password on next login

ObjectMethod passwordError `() -> $string`

Returns a string indicating the error that happened in the password handlers TODO: these delayed error's should be replaced with Exceptions.

returns undef if no error

ObjectMethod removeUser `($cUID) -> $boolean`

Delete the users entry. Removes the user from the password manager and user mapping manager. Does **not** remove their personal topics, which may still be linked.

ObjectMethod _USERMANAGER `($twiki,$params)`

ObjectMethod _userManagerQueryUsers `($params)`

ObjectMethod _userManagerEditUser `($params)`

ObjectMethod _renderUserDataField `($fieldRef)`

This topic: TWiki > TWikiUsersDotPm
Topic revision: r6 - 2011-08-21 - TWikiContributor