**User Class – (user)**

**Attributes**

* user\_ID – protected – The users unique identifier
* user\_type – protected – The user type array (ID, Name)
* user\_status – protected – The user status array(ID, Name)
* user\_name – protected – The user’s full name
* user\_email – protected – The user’s email address
* user\_phone – protected – The user’s phone number.
* user\_carrier – protected – The user’s phone carrier array(ID, Name).
* userPermList – protected – List of user’s permissions based on userType
* userTypeList – protected – List of possible user types
* userStatusList – protected – List of possible user statuses
* typePermList – protected – Full list of user type permissions (all types, all perms)

**Methods**

* \_\_construct() – This constructor will initialize attributes to their default values and if provided with a user ID will call the Initialize() method.
  + ACCEPTS:
    - $userID – optional, The user’s unique identifier
* ID() – Returns the user’s unique identifier
* Email() – Returns the user’s email address
* Phone() – Returns the user’s phone information (Phone#, Carrier ID, Carrier Name)
  + ACCEPTS:
    - $index – optional, selects which piece of phone data to return
  + RETURNS: Phone# | Carrier ID | Carrier Name
* Type() – Returns the user’s type/group (ID, Name)
  + ACCEPTS:
    - $index – optional, selects which piece of type/group data to return
  + RETURNS: ID | Type/Group Name
* Status() – Returns the user’s status information (ID, Name)
  + ACCEPTS:
    - $index – optional, selects which piece of status data to return
  + RETURNS: ID | Status Name
* Name() – Returns the user’s name information
  + ACCEPTS:
    - $index – optional, selects which part of the full name to return, if the index is outside the max range it will return the last part of the name.
  + RETURNS: User’s Full Name | Part of User’s Name
* Login() – Logs the user into the system using the provided user credentials.
  + ACCEPTS:
    - $userEmail – The user’s email address
    - $userPass – The user’s raw password
    - $remember – Flag to remember the user for 24hrs.
  + RETURNS: True | False
* Remember() – Validates a user’s uniquely generated user code and logs the user into the system. If the code key given is over 24hrs old it will expire and no longer be valid. The unique code key for remembering is **not equal** to the key created by MakeyKey().
  + ACCEPTS:
    - $cKey – The user’s uniquely generated user code key.
  + RETURNS: True | False
* Logout() – Logs a user out of the system, destroys session data, destroys cookie data. User will NOT be remembered.
  + ACCEPTS: void
  + RETURNS: True | False
* MakeKey() – Generates a unique user code key, only valid for \*this user on \*this system. The code key generated is **not equal** to the code key made for Remember().
  + ACCEPTS:
    - $email – optional, the user’s email address, if not given the current user object email attribute will be taken, failure if no email is available.
  + RETURNS: Url\_Encoded Code Key | False
* Validate() – Validates a unique user code key generated by MakeKey(). Keys over 24hrs old will not be considered valid.
  + ACCEPTS:
    - $encoded\_key – An url\_encoded user code key generated by MakeKey()
  + RETURNS: True | False
* Create() – Uses the details provided to create a new user in the system.
  + ACCEPTS:
    - $email – The user’s email address
    - $password – The user’s raw password
    - $typeID – The unique identifier for the type/group the user should be.
    - $data – Extra attribute data array as (name => value) pairs.
  + RETURNS: New User’s ID | False
* Update() – Updates the user’s stored information based on the data provided. If no changes are made, the method will return false, check Error attribute for “no changes made” error phrase to clarify failure due to error.
  + ACCEPTS:
    - $data – Data array of (name => value) pairs.
  + RETURNS: True | False
* Kill() – Kills \*this user, by deleting all traces of them from the database.
  + ACCEPTS: void
  + RETURNS: True | False
* Search() – This method searches the database in order to try and locate a specific or multiple possible users based on a given "needle", paired with a filter option such as "email" etc.. (supports partial needles) [Cannot filter by User-Defined Attributes, yet]
  + ACCEPTS:
    - $filter – The specific user attribute to use as a search focus to reduce result
      * **Type** – user’s type/group, ID or name accepted
      * **First Name** – user’s first name
      * **Last Name** – user’s last name
      * **Status** – the users status, ID or name accepted
      * **Phone** – the user’s phone number
      * **Email** – the user’s email address (default)
    - $needle – Word, phrase, number, or partial of any, the value to search for.
    - $BoE\_flag – Boolean flag to determine the placement of the wild symbol.
      * **False/0** – Begins with $needle, wild symbol **after** needle.
      * **True/1** – Ends with $needle, wild symbol comes **before** needle
  + RETURNS: Array of UserIDs | False
* PermissionList(), PrivilegeList(), DibsList()
* PermList() – Returns an associative array of all permissions for all user types. (typeID, typeName, permName, etc…)
  + ACCEPTS: void
  + RETURNS: Array of Permissions | False
* Permission(), Permissions(), Privileges(), Dibs()
* Permitted() – If given a specific permission name this method will return true if this user has permission or false if they do not have permission. If no specific permission name is given, the entire permission list for \*this user’s type is returned. If there is no user\_ID or user\_typeID present then all permissions are returned false.
  + ACCEPTS:
    - $permName – The name of a specific permission (case in-sensitive)
  + RETURNS: Assoc. Array | True | False
* NewPerm(), NewDibs()
* NewPermission() – Adds a new user permission to the system. All new permissions default to false for all users. You must give them permission where needed.
  + ACCEPTS:
    - $permName – The name of the new permission, it must be unique and case is in-sensitive.
  + RETURNS: True | False
* AlterPerm() – Renames a permission name. ACP permission cannot be changed.
  + ACCEPTS:
    - $permName – The name of the permission to alter
    - $newName – The new name to give the permission.
  + RETURNS: True | False
* DeletePerm(), KillPerm(), RemoveDibs(), DeleteDibs(), KillDibs()
* RemovePerm() – Severs the connection to the database, if one exists
  + ACCEPTS:
    - $permName – The name of the permission to remove
  + RETURNS: True | False
* PermUpdate() – Updates the permissions of a specified user type/group.
  + ACCEPTS:
    - $userType – The user type ID or type name to update permissions of.
    - $pdata – Associative array of (permName => value) pairs to update.
  + RETURNS: True | False
* TypeList() – Gets a list of all available user types/groups. (ID, Name)
  + ACCEPTS: void
  + RETURNS: Array of Types | False
* NewType() – Creates a new user type/group.
  + ACCEPTS:
    - $typeName – The name to give the new type/group.
  + RETURNS: New Type ID | False
* AlterType() – Alter the name of a specified user type/group.
  + ACCEPTS:
    - $type – The type to be altered, ID or name accepted.
    - $newName – The new type/group name to use.
  + RETURNS: True | False
* DeleteType(), KillType()
* RemoveType() – Deletes a specified user type. Cannot remove default types.
  + ACCEPTS:
    - $type – The name or ID of the type to be removed
  + RETURNS: True | False
* StatusList() – Gets a list of all available user types/groups. (ID, Name)
  + ACCEPTS: void
  + RETURNS: Array of Statuses | False
* NewStatus() – Creates a new user status.
  + ACCEPTS:
    - $name – The name to give the new status
  + RETURNS: New Status ID | False
* AlterStatus() – Alter the name of a specified user status.
  + ACCEPTS:
    - $status – The status to be altered, ID or name accepted.
    - $newName – The new status name to use.
  + RETURNS: True | False
* DeleteStatus(), KillStatus()
* RemoveStatus() – Deletes a specified user status. Cannot remove default statuses.
  + ACCEPTS:
    - $type – The name or ID of the status to be removed
  + RETURNS: True | False
* Message() – Sends an HTML email to \*this user’s email address.
  + ACCEPTS:
    - $file – The HTML file to use for the message
    - $vars – The variables to inject into the file, name => value pairs.
  + RETURNS: True | False
* TxtMsg() – Sends a text message to the user’s phone, NOT IMPLEMENTED
* Unpack() – Checks for valid user information stored within the current session. If no valid user information is found within the session, then the system attempts to Remember() the user. If the UnPack() is successful, \*this user object is populated with the users information.
  + ACCEPTS: void
  + RETURNS: True | False
* Pack() – Packs up the current user object and stores it within the current session. If the object is not Unpack()’ed within 2hrs the object will expire and no longer be valid.
  + ACCEPTS: void
  + RETURNS: True | False
* Initialize() – This method uses a user\_ID to gather the associated user data from the database and initialize \*this object with the users details. If no userID is given then the user\_ID attribute is used. Fails if no user\_ID is available.
  + ACCEPTS:
    - $userID – optional, the unique identifier for the user.
  + RETURNS: True | False
* Scramble() – Hashes and/or Encrypts strings. Hashing is not reversible. Encryption is unique to each installation.
  + ACCEPTS:
    - $string – string to be scrambled
    - $cook – Boolean, flag whether or not to hash the $string before encryption
    - $type – the type of algorithm to use for hashing, default = ‘whirlpool’
      * <http://php.net/manual/en/function.hash-algos.php> **for more details.**
  + RETURNS: Scrambled String
* UnScramble() – Decrypts a scrambled string. If the original string was $cook’ed then the hash will be returned, hashing is not reversible.
  + ACCEPTS:
    - $scrambledString – A string that has been Scramble()’ed
  + RETURNS: Org String | Org String’s Hash