# 

# dataValidation.php Documentation

### function validateUsername($username)

Validates a string as a username. A username must have between 3 and 30 characters inclusive, start with a letter,, and may contain: capital and lower case letters, numbers between 0 and 9, and a period and a dash.

* **Parameters:** $username — string — - the username string to validate.

### function validatePassword($password)

Validates a password string. A password must contain at least 6 characters.

* **Parameters:** $password — string — - the string to validate as a password.

### function validatePasswordConfirmation($password, $confirmPassword)

Validates a string as a password, and compares the two strings to guarantee that the passwords match.

* **Parameters:**
  + $password — string — - the string to be validated as a password.
  + $confirmPassword — string — - The String to be compared to the password string, this must match the $password string or an error is thrown.

### function validateEmail($email)

Validates a string as an email. Uses filter\_var to validate the email

* **Parameters:** $email — string — - The string to be validated as an email.

### function validatePhone(&$phone)

Validates a string as a phone number. The given string is scrubbed for any characters that are not numbers, then a check is made to test if the scrubbed string length is equal to 10.

* **Parameters:** string — - The memory location of a string to be scrubbed and validated as a phone number.

### function validateDate($year, $month, $day)

Validates three ints as components of a date, month, day, year. Uses php function checkdate to validate.

* **Parameters:**
  + $year — int — - an integer that represents a year.
  + $month — int — - An integer that represents a month.
  + $day — int — - An integer that represents a day.

### function validateTime($time, $format = "hh:ii:ss")

Validates a string as a time based on a provided format. The format string is provided a default value hh:ii:ss. Uses the php date object to validate.

* **Parameters:**
  + $time — string — - A string to be validated as a time.
  + $format — string — - a string to be used as a template for how to validate the given time. Has a default value of hh:ii:ss.

### function validateNotificationByEmail(&$field)

Validates a string as a boolean value for notify by email. The resulting value will be 1 for true, and 0 for false.

* **Parameters:** $field — string — - The memory location of a string that is a boolean value.

### function validateNotificationByPhone(&$field)

Validates a string as a boolean value for notify by phone. The resulting value will be 1 for true, and 0 for false.

* **Parameters:** $field — string — - The memory location of a string that is a boolean value.

### function setCheckboxValue($field)

Scrubs a string as a boolean variable. If given a string value of true or 1, then return 1, otherwise, return 0. This is used for validating html checkboxes.

* **Parameters:** $field — string — - The string boolean value to be scrubbed.
* **Returns:** int — - The scrubbed boolean value.

### function validatePhoneCarrier($carrier)

Validates a string as a phone carrier based on the supported carriers of our application. Insures that the given string matches one of the carriers that are defined in config.php as a constant.

* **Parameters:** $carrier — string — - A string to be validated as a phone carrier.

# 

# delete.php Documentation

### function delete()

File delete.php description This code will do the following in this order: 1. Restrict access to certain tables within this API on certain conditions. 2. Parse through the entered arrays, dynamically creating a SQL string Query. 3. Query the database with the dynamically generated query.

This file can function without the restrictions in step 1, but will not have any restrictions whatsoever, so it will be a security hole. The restrictions in step 1 will be marked with comments denoting the beginning of restricitons and the end of restrictions. Feel free to try it without the restrictions. I’d advise against deleting them, but commenting them out should be fine.

# 

# deleteProxy.php Documentation

### function deleteFromConferenceAPI()

The main function for the delete proxy. makes validation checks to insure the data provided is complete, then adds the user/admin id to the data. Not all requests need to be run through the proxy, most requests can be made to the API directly. Only requests that are spacific to a user or admin need to be requested through the proxy. Only supports the delete method for: the conference table for an admin, the user\_conference table for users, and the user\_schedule table for users.

### function addSessionVariableToData($idName)

A helper method used to perform validation checks on the provided data to insure that the data is complete, ready to be sent back to the API, and includes the $idName session variable. At the end of this function, an http delete request is made using the httpRequester to the conference API with the new data.

* **Parameters:** $idName — string — - the name of the login system id session variable. Is either user\_id, or admin\_id.

# 

# forgotPasswordFunctions.php Documentation

### function processForgotPasswordRequest()

This is the main function for this module. Given an email through a post request, a user is looked up by email address, which must be unique. A database query is made to insert the forgot password token, and the current time plus the experation constant found in config.php. Once the database query is complete, an email is sent to the user with a custom built url pointing at the reset forgot password page. The url includes a query string that has two arguments, token, which equals to the uniquely randomly generated string that identifys the user as valid, and email, with is the email address the user’s account is associated with.

### function checkIfUserExists(&$pdoUtil)

Performs a check to see if the user exists based on the provided email address in the post request. uses pdo to query the database, if no records are returned, then an error is thrown. Otherwise, the found record is returned.

* **Parameters:**
  + $pdoUtil — PDOUtil — - the custom PDO wrapper for making database queries.
  + $results — array — - The array returned from the PDO query, which holds only one record.

### function generateToken(&$pdoUtil, $email)

Generates a cryptographicly random string of characters for the forgot password token to securely identify the user. Also inserts into the database the token, and the time of experation based on the given email address. When a user recieves an email, they have only so much time before that url expires, and they have to request a new url for resetting their password. The experation time is based on the current time plus a constant defined in config.php that designates in seconds how long the user has until the token expires.

* **Parameters:**
  + $pdoUtil — PDOUtil — - The memory location of the custom object that is a wrapper for php’s PDO class.
  + $email — string — - the email that is associated with the user account that the password is being reset for.
* **Returns:** string — Returns the randomly generated token.

### function sendForgotPasswordEmail($username, $userEmail, $token)

Sends an email to the provided email address including the users username, and the url to reset their forgot password. The URL is constructed here, poinging at the resetForgotPassword.php file, and includes the query string arguments of token and email. Uses a third party library (phpMailer) to send the email.

* **Parameters:**
  + $username — string — - the username of the user to reset the passsword for.
  + $email — string — - The email address associated with the account to reset the password for.
  + $token — string — - The randomly generated token that identifys the account whose password is being requested to reset.

# 

# get.php Documentation

### function get()

File get.php description This code will do the following in this order: 1. Restrict access to certain tables within this API on certain conditions. 2. Parse through the entered arrays, dynamically creating a SQL string Query. 3. Query the database with the dynamically generated query.

This file can function without the restrictions in step 1, but will not have any restrictions whatsoever, so it will be a security hole. The restrictions in step 1 will be marked with comments denoting the beginning of restricitons and the end of restrictions. Feel free to try it without the restrictions. I’d advise against deleting them, but commenting them out should be fine.

# 

# getProxy.php Documentation

### function getFromConferenceAPI()

The main function for the get proxy. makes validation checks to insure the data provided is complete, then adds the user/admin id to the data. Not all requests need to be run through the proxy, most requests can be made to the API directly. Only requests that are spacific to a user or admin need to be requested through the proxy. Only supports the get method for: the conference table for an admin, the user\_conference table for users, and the user\_schedule table for users.

### function addSessionVariableToData($idName)

A helper method used to perform validation checks on the provided data to insure that the data is complete, ready to be sent back to the API, and includes the $idName session variable. At the end of this function, an http get request is made using the httpRequester to the conference API with the new data.

* **Parameters:** $idName — string — - the name of the login system id session variable. Is either user\_id, or admin\_id.

# 

# httpRequester.php Documentation

### class HTTPRequester

This class is taken from the programming form stack overflow. source: https://stackoverflow.com/questions/5647461/how-do-i-send-a-post-request-with-php This is a wrapper for curl for making get, post, put, and delete calls to an API.

### public static function HTTPGet($url, array $params)

Make HTTP-GET call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

### public static function HTTPPost($url, array $params)

Make HTTP-POST call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

### public static function HTTPPut($url, array $params)

Make HTTP-PUT call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

### public static function HTTPDelete($url, array $params)

Make HTTP-DELETE call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

# 

# includeConfig.php Documentation

### function includeConfigFile()

Check the session variable user, to check which user type is being accessed, user or admin. Once the user is determined, the appropriate config.php file is included.

# 

# loginFunctions.php Documentation

### function login()

This is the main function for this module. Provided a username and password through a post request, the user is authenticated, and if the credentials are incorrect, then an error is thrown. First the username and password are validated, then a call to loginUser is made to authenticate the user. On success, the user is redirected to the landing page, other wise an error is thrown.

### function loginUser(&$pdoUtil, $username, $password)

Authenicate a user based on a provided username and password. Retrieve the user account information for the provided username. Check if the lockout attempts and time are valid. If the user is not locked out, compare the provided password with the password in the database based on the username. Throw an error if any issue arises in the authentication process. The lockout attempts are based on a constant defined in config.php, a user is only able to attempt loging in so many times until they are locked out for a specified amount of time based on a constant in the config.php file. A user is looked up based on the provided username, and all of the account information is returned. The provided password is hashed, and compared with the password hash in the database associated with the username. If the passwords do not match, then a the lockout attempts are incremented and an error is thrown. Session variables are created for all of the user account information to allow for accessing protected pages and future use.

* **Parameters:**
  + PDOUtil — - The memory location of the custom PDO wrapper object.
  + $username — string — - The username to authentacate @param string $password - the password to authenticate.

### function getSQLSelectForAllFields()

A helper method to construct the sql statement for collecting the necessary data from the user\_acounts table for logging in. The SQL query is constructed based on a constant defined in config.php, which defines the aditional fields in the database that are user data other than the username, password, and email which are required for this login in system to work.

* **Returns:** string — - the constructed sql query

# 

# logoutFunctions.php Documentation

### function logout()

This is the main function for this module. Performs the needed actions to logout of the currently logged in account. The session is destroyed, and the user is redirected to the login page. Note that because the admin and user udilize the same session, by logging out of either the user or admin, will log you out of the other at the same time.

# 

# pdoUtil.php Documentation

### class PDOUtil

PDOUtil class depends on the credentials for the database found in creds.php uses the singleton pattern to insure that there is not more than one connection to the database open at any given time. Instantiate PDOUtil using the createPDOUtil method, query, and close. the query method returns an associative array based on the results of the query. Sometimes there will be nothing in the array, it will be a 1d array, or it could be a 2d array.

* **Example:** \* $pdoUtil = PDOUtil::createPDOUtil();
* $results = $pdoUtil->query(“select \* from Users where id=?”, [12]);
* echo var\_dump($results);
* $pdoUtil->close();

### private function \_\_construct()

The private constructor only to be called by createPDOUtil This method creates a PDO object by using the credentials for the target database found in creds.php

### public static function createPDOUtil()

Creates an instance of PDOUtil and returns it if there is not already an existing instance of PDOUtil if there is an existing instance, return that instance.

### public function query($sql, $variables)

Make a query using the open PDO connection You may include an array of arguments to this object based on the order they would be put into the prepared statement an empty array means there are no arguments for the query This method returns an associative array based on the query results. The array can differ, being empty, 1d, or 2d. This function uses prepared statements to improve security, and mitagate SQL injections.

* **Parameters:**
  + $sql — string — - the sql query
  + $variables — array — - The parameters to be used in the prepared statement.
* **Returns:** array — - The results from the sql query as an associative array.

### public function getLastInsertedID()

This method gets the id of the last inserted record

* **Returns:** string — - the id of the last record inserted into the database. Useful if your ids are generated automatically.

### public function close()

This closes the PDO connection to the database.

# 

# dataValidation.php Documentation

### function validateUsername($username)

Validates a string as a username. A username must have between 3 and 30 characters inclusive, start with a letter,, and may contain: capital and lower case letters, numbers between 0 and 9, and a period and a dash.

* **Parameters:** $username — string — - the username string to validate.

### function validatePassword($password)

Validates a password string. A password must contain at least 6 characters.

* **Parameters:** $password — string — - the string to validate as a password.

### function validatePasswordConfirmation($password, $confirmPassword)

Validates a string as a password, and compares the two strings to guarantee that the passwords match.

* **Parameters:**
  + $password — string — - the string to be validated as a password.
  + $confirmPassword — string — - The String to be compared to the password string, this must match the $password string or an error is thrown.

### function validateEmail($email)

Validates a string as an email. Uses filter\_var to validate the email

* **Parameters:** $email — string — - The string to be validated as an email.

### function validatePhone(&$phone)

Validates a string as a phone number. The given string is scrubbed for any characters that are not numbers, then a check is made to test if the scrubbed string length is equal to 10.

* **Parameters:** string — - The memory location of a string to be scrubbed and validated as a phone number.

### function validateDate($year, $month, $day)

Validates three ints as components of a date, month, day, year. Uses php function checkdate to validate.

* **Parameters:**
  + $year — int — - an integer that represents a year.
  + $month — int — - An integer that represents a month.
  + $day — int — - An integer that represents a day.

### function validateTime($time, $format = "hh:ii:ss")

Validates a string as a time based on a provided format. The format string is provided a default value hh:ii:ss. Uses the php date object to validate.

* **Parameters:**
  + $time — string — - A string to be validated as a time.
  + $format — string — - a string to be used as a template for how to validate the given time. Has a default value of hh:ii:ss.

### function validateNotificationByEmail(&$field)

Validates a string as a boolean value for notify by email. The resulting value will be 1 for true, and 0 for false.

* **Parameters:** $field — string — - The memory location of a string that is a boolean value.

### function validateNotificationByPhone(&$field)

Validates a string as a boolean value for notify by phone. The resulting value will be 1 for true, and 0 for false.

* **Parameters:** $field — string — - The memory location of a string that is a boolean value.

### function setCheckboxValue($field)

Scrubs a string as a boolean variable. If given a string value of true or 1, then return 1, otherwise, return 0. This is used for validating html checkboxes.

* **Parameters:** $field — string — - The string boolean value to be scrubbed.
* **Returns:** int — - The scrubbed boolean value.

### function validatePhoneCarrier($carrier)

Validates a string as a phone carrier based on the supported carriers of our application. Insures that the given string matches one of the carriers that are defined in config.php as a constant.

* **Parameters:** $carrier — string — - A string to be validated as a phone carrier.

# 

# delete.php Documentation

### function delete()

File delete.php description This code will do the following in this order: 1. Restrict access to certain tables within this API on certain conditions. 2. Parse through the entered arrays, dynamically creating a SQL string Query. 3. Query the database with the dynamically generated query.

This file can function without the restrictions in step 1, but will not have any restrictions whatsoever, so it will be a security hole. The restrictions in step 1 will be marked with comments denoting the beginning of restricitons and the end of restrictions. Feel free to try it without the restrictions. I’d advise against deleting them, but commenting them out should be fine.

# 

# deleteProxy.php Documentation

### function deleteFromConferenceAPI()

The main function for the delete proxy. makes validation checks to insure the data provided is complete, then adds the user/admin id to the data. Not all requests need to be run through the proxy, most requests can be made to the API directly. Only requests that are spacific to a user or admin need to be requested through the proxy. Only supports the delete method for: the conference table for an admin, the user\_conference table for users, and the user\_schedule table for users.

### function addSessionVariableToData($idName)

A helper method used to perform validation checks on the provided data to insure that the data is complete, ready to be sent back to the API, and includes the $idName session variable. At the end of this function, an http delete request is made using the httpRequester to the conference API with the new data.

* **Parameters:** $idName — string — - the name of the login system id session variable. Is either user\_id, or admin\_id.

# 

# forgotPasswordFunctions.php Documentation

### function processForgotPasswordRequest()

This is the main function for this module. Given an email through a post request, a user is looked up by email address, which must be unique. A database query is made to insert the forgot password token, and the current time plus the experation constant found in config.php. Once the database query is complete, an email is sent to the user with a custom built url pointing at the reset forgot password page. The url includes a query string that has two arguments, token, which equals to the uniquely randomly generated string that identifys the user as valid, and email, with is the email address the user’s account is associated with.

### function checkIfUserExists(&$pdoUtil)

Performs a check to see if the user exists based on the provided email address in the post request. uses pdo to query the database, if no records are returned, then an error is thrown. Otherwise, the found record is returned.

* **Parameters:**
  + $pdoUtil — PDOUtil — - the custom PDO wrapper for making database queries.
  + $results — array — - The array returned from the PDO query, which holds only one record.

### function generateToken(&$pdoUtil, $email)

Generates a cryptographicly random string of characters for the forgot password token to securely identify the user. Also inserts into the database the token, and the time of experation based on the given email address. When a user recieves an email, they have only so much time before that url expires, and they have to request a new url for resetting their password. The experation time is based on the current time plus a constant defined in config.php that designates in seconds how long the user has until the token expires.

* **Parameters:**
  + $pdoUtil — PDOUtil — - The memory location of the custom object that is a wrapper for php’s PDO class.
  + $email — string — - the email that is associated with the user account that the password is being reset for.
* **Returns:** string — Returns the randomly generated token.

### function sendForgotPasswordEmail($username, $userEmail, $token)

Sends an email to the provided email address including the users username, and the url to reset their forgot password. The URL is constructed here, poinging at the resetForgotPassword.php file, and includes the query string arguments of token and email. Uses a third party library (phpMailer) to send the email.

* **Parameters:**
  + $username — string — - the username of the user to reset the passsword for.
  + $email — string — - The email address associated with the account to reset the password for.
  + $token — string — - The randomly generated token that identifys the account whose password is being requested to reset.

# 

# get.php Documentation

### function get()

File get.php description This code will do the following in this order: 1. Restrict access to certain tables within this API on certain conditions. 2. Parse through the entered arrays, dynamically creating a SQL string Query. 3. Query the database with the dynamically generated query.

This file can function without the restrictions in step 1, but will not have any restrictions whatsoever, so it will be a security hole. The restrictions in step 1 will be marked with comments denoting the beginning of restricitons and the end of restrictions. Feel free to try it without the restrictions. I’d advise against deleting them, but commenting them out should be fine.

# 

# getProxy.php Documentation

### function getFromConferenceAPI()

The main function for the get proxy. makes validation checks to insure the data provided is complete, then adds the user/admin id to the data. Not all requests need to be run through the proxy, most requests can be made to the API directly. Only requests that are spacific to a user or admin need to be requested through the proxy. Only supports the get method for: the conference table for an admin, the user\_conference table for users, and the user\_schedule table for users.

### function addSessionVariableToData($idName)

A helper method used to perform validation checks on the provided data to insure that the data is complete, ready to be sent back to the API, and includes the $idName session variable. At the end of this function, an http get request is made using the httpRequester to the conference API with the new data.

* **Parameters:** $idName — string — - the name of the login system id session variable. Is either user\_id, or admin\_id.

# 

# httpRequester.php Documentation

### class HTTPRequester

This class is taken from the programming form stack overflow. source: https://stackoverflow.com/questions/5647461/how-do-i-send-a-post-request-with-php This is a wrapper for curl for making get, post, put, and delete calls to an API.

### public static function HTTPGet($url, array $params)

Make HTTP-GET call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

### public static function HTTPPost($url, array $params)

Make HTTP-POST call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

### public static function HTTPPut($url, array $params)

Make HTTP-PUT call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

### public static function HTTPDelete($url, array $params)

Make HTTP-DELETE call

* **Parameters:**
  + $url — string — - The url to target with your HTTP request.
  + $params — array — - The array of arguments that are to be passed along with the request.
* **Returns:** HTTP-Response — body or an empty string if the request fails or is empty

# 

# includeConfig.php Documentation

### function includeConfigFile()

Check the session variable user, to check which user type is being accessed, user or admin. Once the user is determined, the appropriate config.php file is included.

# 

# loginFunctions.php Documentation

### function login()

This is the main function for this module. Provided a username and password through a post request, the user is authenticated, and if the credentials are incorrect, then an error is thrown. First the username and password are validated, then a call to loginUser is made to authenticate the user. On success, the user is redirected to the landing page, other wise an error is thrown.

### function loginUser(&$pdoUtil, $username, $password)

Authenicate a user based on a provided username and password. Retrieve the user account information for the provided username. Check if the lockout attempts and time are valid. If the user is not locked out, compare the provided password with the password in the database based on the username. Throw an error if any issue arises in the authentication process. The lockout attempts are based on a constant defined in config.php, a user is only able to attempt loging in so many times until they are locked out for a specified amount of time based on a constant in the config.php file. A user is looked up based on the provided username, and all of the account information is returned. The provided password is hashed, and compared with the password hash in the database associated with the username. If the passwords do not match, then a the lockout attempts are incremented and an error is thrown. Session variables are created for all of the user account information to allow for accessing protected pages and future use.

* **Parameters:**
  + PDOUtil — - The memory location of the custom PDO wrapper object.
  + $username — string — - The username to authentacate @param string $password - the password to authenticate.

### function getSQLSelectForAllFields()

A helper method to construct the sql statement for collecting the necessary data from the user\_acounts table for logging in. The SQL query is constructed based on a constant defined in config.php, which defines the aditional fields in the database that are user data other than the username, password, and email which are required for this login in system to work.

* **Returns:** string — - the constructed sql query

# 

# logoutFunctions.php Documentation

### function logout()

This is the main function for this module. Performs the needed actions to logout of the currently logged in account. The session is destroyed, and the user is redirected to the login page. Note that because the admin and user udilize the same session, by logging out of either the user or admin, will log you out of the other at the same time.

# 

# pdoUtil.php Documentation

### class PDOUtil

PDOUtil class depends on the credentials for the database found in creds.php uses the singleton pattern to insure that there is not more than one connection to the database open at any given time. Instantiate PDOUtil using the createPDOUtil method, query, and close. the query method returns an associative array based on the results of the query. Sometimes there will be nothing in the array, it will be a 1d array, or it could be a 2d array.

* **Example:** \* $pdoUtil = PDOUtil::createPDOUtil();
* $results = $pdoUtil->query(“select \* from Users where id=?”, [12]);
* echo var\_dump($results);
* $pdoUtil->close();

### private function \_\_construct()

The private constructor only to be called by createPDOUtil This method creates a PDO object by using the credentials for the target database found in creds.php

### public static function createPDOUtil()

Creates an instance of PDOUtil and returns it if there is not already an existing instance of PDOUtil if there is an existing instance, return that instance.

### public function query($sql, $variables)

Make a query using the open PDO connection You may include an array of arguments to this object based on the order they would be put into the prepared statement an empty array means there are no arguments for the query This method returns an associative array based on the query results. The array can differ, being empty, 1d, or 2d. This function uses prepared statements to improve security, and mitagate SQL injections.

* **Parameters:**
  + $sql — string — - the sql query
  + $variables — array — - The parameters to be used in the prepared statement.
* **Returns:** array — - The results from the sql query as an associative array.

### public function getLastInsertedID()

This method gets the id of the last inserted record

* **Returns:** string — - the id of the last record inserted into the database. Useful if your ids are generated automatically.

### public function close()

This closes the PDO connection to the database.

# 

# dataValidation.php Documentation

### function validateUsername($username)

Validates a string as a username. A username must have be

# 

# post.php Documentation

### function post()

File post.php description This code will do the following in this order: 1. Restrict access to certain tables within this API on certain conditions. 2. Parse through the entered arrays, dynamically creating a SQL string Query. 3. Query the database with the dynamically generated query.

This file can function without the restrictions in step 1, but will not have any restrictions whatsoever, so it will be a security hole. The restrictions in step 1 will be marked with comments denoting the beginning of restricitons and the end of restrictions. Feel free to try it without the restrictions. I’d advise against deleting them, but commenting them out should be fine.

# 

# postProxy.php Documentation

### function postToConferenceAPI()

The main function for the post proxy. makes validation checks to insure the data provided is complete, then adds the user/admin id to the data. Not all requests need to be run through the proxy, most requests can be made to the API directly. Only requests that are spacific to a user or admin need to be requested through the proxy. Only supports the post method for: the conference table for an admin, the event table for an admin, the user\_conference table for users, and the user\_schedule table for users.

### function addSessionVariableToData($idName)

A helper method used to perform validation checks on the provided data to insure that the data is complete, ready to be sent back to the API, and includes the $idName session variable. At the end of this function, an http post request is made using the httpRequester to the conference API with the new data.

* **Parameters:** $idName — string — - the name of the login system id session variable. Is either user\_id, or admin\_id.

# 

# put.php Documentation

### function put()

File put.php description This code will do the following in this order: 1. Restrict access to certain tables within this API on certain conditions. 2. Parse through the entered arrays, dynamically creating a SQL string Query. 3. Query the database with the dynamically generated query.

This file can function without the restrictions in step 1, but will not have any restrictions whatsoever, so it will be a security hole. The restrictions in step 1 will be marked with comments denoting the beginning of restricitons and the end of restrictions. Feel free to try it without the restrictions. I’d advise against deleting them, but commenting them out should be fine.

# 

# putProxy.php Documentation

### function putToConferenceAPI()

The main function for the put proxy. makes validation checks to insure the data provided is complete, then adds the user/admin id to the data. Not all requests need to be run through the proxy, most requests can be made to the API directly. Only requests that are spacific to a user or admin need to be requested through the proxy. Only supports the put method for: the conference table for an admin, the event table for an admin, the user\_conference table for users, the user\_accounts table for users, and the user\_schedule table for users.

### function addSessionVariableToData($idName, &$putData)

A helper method used to perform validation checks on the provided data to insure that the data is complete, ready to be sent back to the API, and includes the $idName session variable. At the end of this function, an http put request is made using the httpRequester to the conference API with the new data. After the request to the API has been made and returns, a check is made to see if the request was successfull, and if the table requested is either the user\_acounts or admin\_accounts table, then update the login system session variables with the user’s put data.

* **Parameters:**
  + $idName — string — - the name of the login system id session variable. Is either user\_id, or admin\_id.
  + array — - The actual memory location of the $\_PUT array

### function modifyUserSessionVariables(&$putData)

Updates the users login system session variable with the put data. Modifys the user\_name, user\_email, user\_phone, user\_phoneCarrier, user\_notifyByEmail, and user\_notifyByPhone session variables. It is likely that the user is only changing a couple of these, so the rest are just being reset with the same value.

* **Parameters:** array — - The memory location of $\_PUT

### function modifyAdminSessionVariables($putData)

Updates the admins login system session variable with the put data. Modifys the admin\_name, and the admin\_email session variables. It is likely that the admin is only changing 1 of these, so the rest are just being reset with the same value.

* **Parameters:** array — - The memory location of $\_PUT

### function setSessionVariable($name, $putData)

a helper method for setting session variables to the correct value from the put data. Searches the put data attrs array for the specified attribute name, then assigns the session variable at the given index value to the value found in the put data values array at the found index.

* **Parameters:**
  + $name — string — - the attribute name of the value in the session and in the put data.
  + $putData — array — - The $\_PUT array.

# 

# registerFunctions.php Documentation

### function register()

This is the main function for this module. This module relys a lot on config.php to determine what aditional data is provided other than username, email, and password. HTML form field data is collected and sent to this module to handle user account creation via a post request. The data is validated and scrubbed before any actions are performed. Once the data has passed validation, a check is made to see if the given email address and or username has already been chosen or not. If true, an error is thrown. The next step is to insert the data into the database, where all of the prepared data from the post request is inserted into the database, On success, the user is redirected to the login page.

### function checkIfEmailExists(&$pdoUtil, $email)

A helper function used to check the database to see if the email has already been used or not in the creation of a user account. If the email exists already, then throw an error.

* **Parameters:**
  + PDOUtil — - The memory location of the custom PDO wrapper object.
  + $email — string — - The email address to check for in the database.

### function getSQLInsertAllFields(&$parameters)

Creates the SQL query needed to insert a record into the database. The parameters array is also constructed here. the parameters array is used as the array for the prepared statements, which contains the values to be inserted into the database.

* **Parameters:** array — - The memory location of an array that will contain the values to be inserted into the database record for the new account.

# 

# resetForgotPasswordFunctions.php Documentation

### function resetForgotPassword()

This is the main function for this module. Provided a correct query string get request, this will process a users reset password request. This is expecting the get arguments token, and email, should be accessed via the URL sent to the user by email for resetting their account password. This uses the provided email address to lookup a user and compare the provided token with the token in the database. This validates the user and provides the needed information for resetting the users password. The token does expire after a set amount of time, defined in config.php, if the time stamp in the database is smaller than the current time, then throw a locked out error. Otherwise, validate the post request for the password, and update the database with the new password.

### function validateUser(&$pdoUtil, $email, $token)

Validate that the email and token provided match a user account record in the database. Once located, and determined to be valid, extract the username and return it.

* **Parameters:**
  + PDOUtil — - The memory location of the PDO wrapper object.
  + $email — string — - The email used to identify the user account.
  + $token — string — - The randomly generated token that is used to validate the account for resetting a password.
* **Returns:** string — - the username of the account to reset the passwowrd for.

### function updatePassword(&$pdoUtil, $username, $newPassword, $confirmNewPassword)

Update the user accounts password based on the provided username. The fields are checked that the confirmation password matches the password field, then the password is updated based on the username.

* **Parameters:**
  + PDOUtil — - The memory location of the PDO wrapper object.
  + $username — string — - The username to identify the account by.
  + $newPassword — string — - The new password for the account.
  + $confirmNewPassword — string — - The confirmation of the new password.

### function sendSuccessEmail($username, $userEmail)

On success of the password being reset, a success email is sent out to notify them that their user account password was reset. This uses a third-party library called phpMailer for sending out emails.

* **Parameters:**
  + $username — string — - The username for the account in which the password was reset for.
  + $userEmail — string — - The email address for the user to send the success email to.

# 

# resetPasswordFunctions.php Documentation

### function resetPassword()

This is the main function for this module. If a user knows their existing password, they can change it to something diferent. The first check is to authenticate the users current password, then a check is made to insure that the new password and the confirmation password match. Finally, a query is made to update the users password. This depends on the user already being logged in, sinse the login system session variables are used to validate the user.

### function verifyCurrentPassword($pdoUtil, $username, $currentPassword)

Validates the existing password against the password hash in the database based on the session username session variable. The session username is first verified against the data in the database, if 1 record is returned, then the application continues. The provided password is hashed, and compared to the password hash in the database, if they don’t match, then false is returned.

* **Parameters:**
  + PDOUtil — - The memory location of the PDO wrapper object.
  + $username — string — - The username to identify the account by.
  + $currentPassword — string — - The current password that is used to sign into this account.
* **Returns:** boolean — - returns true if the current password is valid, and false if not.