Below is a detailed documentation of the major functions in the provided `oblivion.js` file, which powers the Oblivion Alchemy Recipes Finder web application. The documentation focuses on key functions, their purposes, parameters, return values, and roles in the application's functionality. The application is designed to find and display alchemy recipes for the game *The Elder Scrolls IV: Oblivion*, considering skill-based effect restrictions, ingredient availability, and user-defined filters. Each function is described with its logic, usage, and any notable behaviors or dependencies.

Overview

`oblivion.js` is a JavaScript file that implements the logic for an interactive web-based tool to generate and filter alchemy recipes for *Oblivion*. It integrates with an HTML interface (`index.html`) and uses jQuery, jQuery UI, and the Pako library for gzip decompression. The tool allows users to select ingredients, specify recipe sizes (2–4 ingredients), set an alchemy skill level (0, 25, 50, 75), and apply filters (e.g., Shivering Isles ingredients, effect inclusion/exclusion). Recipes are pre-generated and stored in `recipes.json.gz` , loaded at startup, and filtered based on user inputs.
Key features:

- Displays recipes sorted by frequency, effect count, or no sorting.
- Supports skill-based effect restrictions (e.g., higher alchemy skill unlocks more effects per ingredient).
- Filters recipes by ingredient frequency, Shivering Isles inclusion, and effect presence.
- Provides interactive UI elements like ingredient autocompletion, tooltips, and pagination.

The major functions handle tasks such as initializing the UI, loading data, filtering recipes, rendering results, and managing user interactions.

Major Functions

1. `getCallerLineNumber()`

Purpose: Debugging utility to retrieve the line number and context of the calling function from the stack trace.

Parameters: None

Returns: `string` - A string representing the stack frame of the caller (third line of the stack trace).

Description:

- Extracts the third line from the error stack to identify the calling function and its location.
- Used for debugging to trace where functions are called (e.g., in `refresh`).
- Currently returns the entire stack line (e.g., function name and file location) but has commented-out code to extract just the line number or function name.

Usage:

```javascript
console.log(getCallerLineNumber()); // Outputs: "refreshResults @ oblivion.js:123"
```

Notes:

- Primarily used in `refresh` for debugging.
- Relies on `Error.stack`, which may vary across browsers.

---

2. `$(document).ready()`

Purpose: Initializes the application when the DOM is fully loaded.

Parameters: None (jQuery event handler).

Returns: None
Description:

- Fetches and decompresses the pre-generated recipes from `i/recipes.json.gz` using the Pako library.
- Parses the JSON data into `preGeneratedRecipes`, a global object mapping recipe sizes (e.g., `"2"`, `"3"`, `"4"`) to arrays of recipe arrays `[ingredients, effectIds, skill, purity, value]`.
- Sets up the jQuery UI autocomplete for the ingredient input field (`#autocomplete`) using the `relIngredient` array.
- Binds change event listeners to form inputs (`recipeSize`, `alch`, `freq`, `si`, `pure`, `sort`, `asc`) to trigger `refresh` when modified.
- Initializes the effects dropdown (`#effects`) with all available effects from the `effects` array.
- Calls `buildEffects` to populate the effect list UI.
- Applies jQuery UI button styling to buttons.

Usage:

- Automatically executed when the page loads.
- Sets up the initial state and event listeners.

Notes:

- Depends on external libraries: jQuery, jQuery UI, and Pako.
- Assumes `recipes.json.gz` is correctly formatted and accessible.
- Logs the number of loaded recipe sizes to the console.

---

3. `buildEffects()`

Purpose: Populates the effect list UI (`#listeffects`) with all available effects and adds tooltips.

Parameters: None

Returns: None

Description:

- Generates HTML for each effect in the `effects` array, displaying the effect name, value (e.g., `$43`), and color-coding based on affinity (green for positive, red for negative).
- Sets the `#listeffects` element's content and makes it visible.
- Attaches jQuery UI tooltips to each effect, using `hoverEffect` to generate tooltip content dynamically.
- Tooltips show ingredients that have the effect, sorted by effect position and ingredient name.

Usage:

- Called during initialization in `$(document).ready()` .

```javascript
buildEffects(); // Populates #listeffects with effect list and tooltips
```

Notes:

- Uses `effects` array: `[name, worth, affinity]` .
- Relies on `hoverEffect` for tooltip content.
- Tooltip positioning is configured to appear to the right of the effect name.

---

4. `hoverEffect()`

Purpose: Generates tooltip content for effects, listing ingredients that have the effect and their effect positions.

Parameters: None (called by jQuery UI tooltip, with `this` as the DOM element).

Returns: `string` - HTML string for the tooltip, containing a table of ingredients with the effect.

Description:

- Retrieves the effect ID from the element's `data-id` attribute.
- Maps `allIngredients` to find ingredients containing the effect, including their effect position (1-based index).
- Sorts ingredients by effect position and then alphabetically by name.
- Builds an HTML table with columns for effect position and ingredient name, grouping by position.
- Returns a fallback message if the effect ID is invalid.

Usage:

- Used as the `content` callback for effect tooltips in `buildEffects` and `addHoverEffects`.

```javascript
$("#listeffects .effect").tooltip({ content: hoverEffect });
```

Notes:

- Uses `allIngredients` and `effects` arrays.
- Handles edge cases (e.g., invalid `effectId`) with a warning and empty table.
- Optimizes sorting and grouping for readability in the tooltip.

---

5. `hoverIngredients()`

Purpose: Generates tooltip content for ingredients, showing their effects and frequency.

Parameters: None (called by jQuery UI tooltip, with `this` as the DOM element).

Returns: `string` - HTML string for the tooltip, listing the ingredient's effects and frequency.

Description:

- Retrieves the ingredient ID from the element's `data-name` attribute.
- If `id` is 1000 (placeholder for empty ingredient slots), returns an empty string.
- Gets the ingredient's effects from `allIngredients[id][1]`.
- Generates HTML for each effect, color-coded by affinity (green for positive, red for negative).
- Includes the ingredient's frequency (`allIngredients[id][2]`).

Usage:

- Used as the `content` callback for ingredient tooltips in `refreshResults` and `addHoverEffects`.

```javascript
$("#ingredients .ingredient").tooltip({ content: hoverIngredients });
```

Notes:

- Depends on `allIngredients`, `effects`, and `relEffect`.
- Sums effect worths in the tooltip (though not displayed in the current implementation).

---

6. `addFilter()`

Purpose: Adds a filter based on an effect to include or exclude recipes.

Parameters: None

Returns: None

Description:

- Reads the filter type (`hasEffect`, 1 for "Has", 0 for "Not") from `#filter` and the effect ID from `#effects`.
- Creates a filter entry: `[description, filterFunction]`, where `description` is a string (e.g., "Has Burden") and `filterFunction` checks if the effect is present (`effectIds.includes(effectId)`) based on `hasEffect`.
- Adds the filter to the global `filters` array.
- Resets the pagination `offset` to 0 and calls `refresh(false)` to update the results.

Usage:

- Triggered by clicking the “Add Filter” button.

```javascript
addFilter(); // Adds a filter and refreshes results
```

Notes:

- Filters are applied in `refreshResults` to exclude recipes.
- Uses `relEffect` for effect names.

---

7. `addItemFilter(ingredientId, hasIngredient)`

Purpose: Adds a filter to include or exclude recipes based on an ingredient.

Parameters:

- `ingredientId` (`number`): Index of the ingredient in `allIngredients`.
- `hasIngredient` (`boolean`): `true` to include recipes with the ingredient, `false` to exclude.

Returns: None
Description:

- Creates a filter entry: `[description, filterFunction]`, where `description` is a string (e.g., “Has Alkanet Flower”) and `filterFunction` checks if the ingredient is present (`ingredients.includes(ingredientId)`) based on `hasIngredient`.
- Adds the filter to the global `filters` array.
- Resets the pagination `offset` to 0 and calls `refresh(false)` to update the results.

Usage:

- Triggered by clicking the “add” or “exclude” icons next to an ingredient in the UI.

```javascript
addItemFilter(0, true); // Filter for recipes with Alkanet Flower
```

Notes:

- Similar to `addFilter` but operates on ingredients instead of effects.

- Uses `relIngredient` for ingredient names.

---

8. `refresh(rebuildMatches)`

Purpose: Initiates the process to update the recipe results based on user inputs.

Parameters:

- `rebuildMatches` (`boolean`): If `true`, rebuilds the `matches` array by filtering pre-generated recipes.

Returns: None

Description:

- On first run, makes the effect list (`#listeffects`) and controls (`#controls`) visible and sets `firstRun` to `false`.
- Clears the results (`#results`) and displays a "Generating recipes" message in `#warn`.
- Schedules `refreshResults(rebuildMatches)` to run after a 20ms delay using `setTimeout` to allow UI updates.

Usage:

- Called when form inputs change (e.g., recipe size, alchemy skill) or ingredients/filters are modified.

```javascript
refresh(true); // Rebuild matches and update results
```

Notes:

- Logs the `rebuildMatches` parameter and caller line number for debugging.
- The delay ensures the UI updates before heavy computation.

---

9. `refreshResults(rebuildMatches)`

Purpose: Filters and renders the recipe results based on user inputs and filters.

Parameters:

- `rebuildMatches` (`boolean`): If `true`, rebuilds the `matches` array by calling `filterPreGeneratedRecipes`.

Returns: None

Description:

- Reads user inputs: `pure` (potion type), `freq` (exclude frequency), `si` (Shivering Isles), `recipeSize`, `alchemySkill`, `maxResults`, `sort`, and `asc` (ascending sort).
- If `rebuildMatches` is `true`, calls `deleteRare` to update `have` and `exclude` arrays based on `freq` and `si`, resets `offset`, and rebuilds `matches` using `filterPreGeneratedRecipes`.
- Sorts `matches` based on the `sort` type (0: none, 1: effect count, 2: frequency) and `asc` flag.
- Renders the "Your Ingredients" section (`#added`) with the current `have` and `exclude` lists, including icons for removing or filtering ingredients.
- Applies filters to `matches` to create `filteredMatches`, excluding recipes that:
  - Fail any filter in `filters` (via `filter` function).
  - Contain ingredients with frequency ≤ `freq`.
  - Include Shivering Isles ingredients when `si` is `false`.
  - Don't match the `pure` type (0: any, 1: pure positive, 2: pure negative, 3: pure).
- Generates an HTML table for paginated results (`offset` to `offset + maxResults`), showing ingredients, effects, and sort value.
- Adds tooltips to ingredients and effects using `addHoverEffects`.
- Updates the UI with the results and a pagination summary (e.g., "Showing 1 to 40 of 2582 recipes").

Usage:

- Called by `refresh` after a delay.

```javascript
refreshResults(true); // Rebuild and render results
```

Notes:

- Global variables: `filteredMatches`, `maxResults`, `offset`.
- Handles edge cases: empty `have` (displays "No ingredients selected"), empty `filteredMatches` (displays "No recipes found").
- Logs `filteredMatches.length` for debugging.

---

10. `filterPreGeneratedRecipes(alchemySkill, recipeSize)`

Purpose: Filters pre-generated recipes based on alchemy skill and recipe size, updating effect usability.

Parameters:

- `alchemySkill` (`number`): The player's alchemy skill level (0, 25, 50, 75).
- `recipeSize` (`number`): Number of ingredients in the recipe (2, 3, or 4).

Returns: `Array` - Filtered recipes, each as `[ingredients, usableEffects, skill, purity, value]`.

Description:

- Filters `preGeneratedRecipes[recipeSize]` to include only recipes where the required skill (`pregen[2]`) is ≤ `alchemySkill`.
- For each recipe, calls `getUsableEffectsForRecipe` to determine usable effects based on `alchemySkill`.
- Recalculates `purity` using `calcPurity` based on the usable effects.
- Updates the `value` array's effect count (`value[1]`) to reflect usable effects.
- Filters out recipes with fewer than `recipeSize - 1` usable effects (e.g., at least 3 effects for a 4-ingredient recipe).

Usage:

- Called in `refreshResults` when `rebuildMatches` is `true`.

```javascript
matches = filterPreGeneratedRecipes(75, 4); // Filter 4-ingredient recipes for skill 75
```

Notes:

- The `recipeSize - 1` filter is stricter than Oblivion's mechanics (which allow as few as one effect).
- Depends on `getUsableEffectsForRecipe` and `calcPurity`.

---

11. `getUsableEffectsForRecipe(ingredients, effectIds, alchemySkill)`

Purpose: Determines which effects in a recipe are usable based on the alchemy skill and ingredient effect positions.

Parameters:

- `ingredients` (`Array<number>`): Array of ingredient IDs.
- `effectIds` (`Array<number>`): Array of effect IDs for the recipe.
- `alchemySkill` (`number`): The player's alchemy skill level (0, 25, 50, 75).

Returns: `Array<number>` - Array of effect IDs that are usable.

Description:

- Filters `effectIds` to include only effects that are:
  - Present in at least two ingredients (corrected version, as per the previous fix).
  - Accessible based on the `alchemySkill` and the effect's position in each ingredient's effect list:
    - Index 0: Always usable.
    - Index 1: Requires skill ≥ 25.
    - Index 2: Requires skill ≥ 50.
    - Index 3: Requires skill ≥ 75.
- The corrected version (from the previous response) counts valid ingredients and requires at least two to share the effect, aligning with Oblivion's pairwise effect rule.

Usage:

- Called in `filterPreGeneratedRecipes` to update recipe effects.

```javascript
const usableEffects = getUsableEffectsForRecipe([0, 1, 4, 10], [0, 37], 75); // Returns [0, 37]
```

Notes:

- The original version incorrectly required all ingredients to have the effect, causing issues for Recipe Size = 4.
- Corrected version ensures compatibility with Oblivion's alchemy mechanics.

---

12. `findMatches(alchemySkill, recipeSize)`

Purpose: Generates recipes from the user's selected ingredients (`have`) based on alchemy skill and recipe size.
Parameters:

- `alchemySkill` (`number`): The player's alchemy skill level.
- `recipeSize` (`number`): Number of ingredients in the recipe (2, 3, or 4).

Returns: `Array` - Array of recipes, each as `[ingredients, effectIds, value]`.

Description:

- Generates all possible combinations of `recipeSize` ingredients from `have`.
- For each combination:
  - Gets usable effects for each ingredient based on `alchemySkill` (via `getUsableEffects`).
  - Computes pairwise effect intersections (via `getPairwiseEffects`).
  - Filters effects using `canUseEffect` to ensure skill-based accessibility.
  - Excludes redundant recipes (where a smaller subset produces the same effects) using `isRedundantRecipe`.
  - Calculates recipe value (frequency, effect count, none, ingredient string) using `worth`.
- Returns an array of valid, non-redundant recipes.

Usage:

- Called in `addItem` to update `matches` when ingredients are added.

```javascript
matches = findMatches(75, 2); // Generate 2-ingredient recipes for skill 75
```

Notes:

- Used as a fallback or for dynamic recipe generation, but the application primarily uses `preGeneratedRecipes`.
- Computationally intensive for large `have` arrays and `recipeSize = 4`.
- Includes helper functions: `getUsableEffects`, `getPairwiseEffects`, `isRedundantRecipe`, `getCombinations`.

---

13. `canUseEffect(effectId, alchemySkill, ...ingredientIds)`

Purpose: Checks if an effect is usable for the given ingredients based on the alchemy skill.

Parameters:

- `effectId` (`number`): ID of the effect to check.
- `alchemySkill` (`number`): The player's alchemy skill level.
- `...ingredientIds` (`number[]`): Variable number of ingredient IDs.

Returns: `boolean` - `true` if the effect is usable for all provided ingredients, `false` otherwise.

Description:

- For each ingredient, checks if the effect is present and its position is accessible:
  - Index 0: Always usable.
  - Index 1: Requires skill ≥ 25.
  - Index 2: Requires skill ≥ 50.
  - Index 3: Requires skill ≥ 75.
- Returns `false` if the effect is missing or inaccessible for any ingredient.

Usage:

- Used in `findMatches` to validate effects in `getPairwiseEffects`.

```javascript
canUseEffect(37, 75, 0, 1); // Check if Light is usable for Alkanet Flower and Alocasia Fruit
```

Notes:

- Ensures skill-based restrictions are applied consistently.
- Used in dynamic recipe generation, not pre-generated filtering.

---

14. `calcPurity(effectIds)`

Purpose: Determines the purity of a recipe based on its effects' affinities.

Parameters:

- `effectIds` (`Array<number>`): Array of effect IDs.

Returns: `number` - Purity value:

- `1`: All effects are positive (affinity = 1).
- `2`: All effects are negative (affinity = 0).
- `0`: Mixed or no effects.

Description:

- Checks the `affinity` (from `effects[effectId][2]`) of each effect.
- Returns `1` if all effects are positive, `2` if all negative, `0` otherwise.

Usage:

- Called in `filterPreGeneratedRecipes` to update recipe purity.

```javascript
const purity = calcPurity([0, 37]); // Returns 0 (mixed: Burden is negative, Light is positive)
```

Notes:

- Used to filter recipes by purity type in `refreshResults`.

---

15. `add()`

Purpose: Adds a user-entered ingredient to the `have` list and updates results.

Parameters: None
Returns: None

Description:

- Gets the ingredient name from `#autocomplete`, converts to lowercase, and finds its index in `relIngredient`.
- If the ingredient is valid and not already in `have`, adds it to `have`, removes it from `exclude` (via `removeRare`), and calls `refresh(2)` to update results.
- Clears the autocomplete input.

Usage:

- Triggered by clicking the "Add" button next to the autocomplete field.

```javascript
add(); // Add ingredient from #autocomplete
```

Notes:

- The `refresh(2)` argument is unusual (should be `true` / `false`); likely a typo or legacy code.
- Validates input to prevent duplicates or invalid ingredients.

---

16. `addAll()`

Purpose: Adds all ingredients to the `have` list and updates results.

Parameters: None

Returns: None

Description:

- Sets `have` to include all ingredient indices from `allIngredients`.
- Calls `refresh(true)` to rebuild and display results.

Usage:

- Triggered by clicking the "Add All Ingredients" button.

```javascript
addAll(); // Add all ingredients and refresh
```

Notes:

- Resets `have` to all ingredients, ignoring previous selections.
- Used in the default scenario described in the original query.

---

17. `deleteRare(freq, si)`

Purpose: Updates `have` and `exclude` lists based on frequency and Shivering Isles filters.

Parameters:

- `freq` (`number`): Minimum frequency threshold (1–4).
- `si` (`boolean`): Whether to include Shivering Isles ingredients.

Returns: None
Description:

- Clears `exclude` and rebuilds it with indices of ingredients that:

    - Are from Shivering Isles (`allIngredients[i][3] === 1`) and `si` is `false` .
    - Have a frequency (`allIngredients[i][2]`) ≤ `freq` .
- Sets `have` to all ingredient indices not in `exclude` .
- Logs debugging information if `dbg & 2` is set.

Usage:

- Called in `refreshResults` when `rebuildMatches` is `true` .

```javascript
deleteRare(1, true); // Exclude ingredients with frequency ≤ 1, include Shivering Isles
```

Notes:

- Modifies global `have` and `exclude` arrays.
- Critical for filtering recipes based on user settings.

---

18. `addHoverEffects()`

Purpose: Attaches tooltips to effect and ingredient elements in the results table.

Parameters: None
Returns: None

Description:

- Adds jQuery UI tooltips to `.effect` elements using `hoverEffect` and `.ingredient` elements using `hoverIngredients` .
- Configures tooltip positioning (effects to the left, ingredients to the right).

Usage:

- Called in `refreshResults` after rendering the results table.

```javascript
addHoverEffects(); // Add tooltips to results
```

Notes:

- Called with a 100ms delay to ensure DOM elements are rendered.
- Enhances user experience by providing effect and ingredient details on hover.

---

19. `debug(recipeSize, i, alchemySkill)`

Purpose: Debugging function to analyze a specific pre-generated recipe.

Parameters:

- `recipeSize` (`number`): Recipe size (2, 3, or 4).
- `i` (`number`): Index of the recipe in `preGeneratedRecipes[recipeSize]`.
- `alchemySkill` (`number`): The player's alchemy skill level.

Returns: None
Description:

- Retrieves the recipe at `preGeneratedRecipes[recipeSize][i]`.
- Calls `analyze` to log recipe details (ingredients, effects, skill).
- Calls `getUsableEffectsForRecipe` and logs the original and usable effect IDs.

Usage:

- Used for debugging specific recipes, as in the original query.

```javascript
debug(4, 95, 75); // Analyze recipe index 95 for size 4, skill 75
```

Notes:

- Outputs detailed recipe information to the console.
- Helped identify the issue with `getUsableEffectsForRecipe` in the original query.

---

20. `analyze(recipeSize, i)`

Purpose: Logs detailed information about a specific pre-generated recipe for debugging.

Parameters:

- `recipeSize` (`number`): Recipe size (2, 3, or 4).
- `i` (`number`): Index of the recipe in `preGeneratedRecipes[recipeSize]`.

Returns: None

Description:

- Retrieves the recipe: `[ingredientIds, effectIds, skill, purity, value]`.
- Logs the recipe as JSON.
- Logs each ingredient's ID, name, and effects (with effect IDs and names).
- Logs the recipe's effects and required skill.

Usage:

- Called by `debug` to provide detailed recipe output.

```javascript
analyze(4, 95); // Log details of recipe index 95 for size 4
```

Notes:

- Formats output for readability, including effect mappings.
- Useful for verifying recipe data integrity.

Data Structures

- **Global Variables**:
  - `preGeneratedRecipes`: Object mapping recipe sizes (`"2"`, `"3"`, `"4"`) to arrays of `[ingredients, effectIds, skill, purity, value]`.
  - `have`: Array of ingredient indices the user has selected.
  - `exclude`: Array of ingredient indices excluded due to frequency or Shivering Isles filters.
  - `matches`: Array of filtered recipes `[ingredients, effectIds, skill, purity, value]`.
  - `filteredMatches`: Subset of `matches` after applying user filters.
  - `filters`: Array of `[description, filterFunction]` for effect or ingredient filters.
  - `offset`: Pagination offset for displaying results.

- `maxResults` : Maximum number of results per page.

- **Constants**:
  - `effects` : Array of `[name, worth, affinity]` for all alchemy effects.
  - `allIngredients` : Array of `[name, effectIds, frequency, isShiveringIsles]` for all ingredients.
  - `relIngredient` : Array of ingredient names.
  - `relEffect` : Array of effect names.
  - `relWorth` : Array of effect worths.
  - `relAffinity` : Array of effect affinities (0: negative, 1: positive).
  - `ii` : Object mapping ingredient names to indices.

Dependencies

- **External Libraries**:
  - jQuery 3.6.0: DOM manipulation and event handling.
  - jQuery UI 1.13.2: Autocomplete, tooltips, and button widgets.
  - Pako 2.1.0: Gzip decompression for `recipes.json.gz` .
- **Files**:
  - `index.html` : Defines the UI structure.
  - `oblivion.css` : Styles the application.
  - `recipes.json.gz` : Pre-generated recipe data.
  - Icon files (`delete.png` , `plus.png` , `script_add.png` , `script_delete.png`).

Notes

- **Alchemy Mechanics**: The tool reflects *Oblivion*'s alchemy system, where:
  - Effects are unlocked based on alchemy skill (1st effect at 0, 2nd at 25, 3rd at 50, 4th at 75).
  - Potion effects come from effects shared by at least two ingredients in pairwise combinations.
  - Recipes are pre-generated to optimize performance, stored in `recipes.json.gz` .
- **Bugs Fixed**: The previous response corrected `getUsableEffectsForRecipe` to handle pairwise effect sharing, resolving the issue where no recipes were found for Recipe Size = 4.
- **Potential Improvements**:

- Relax the `recipeSize - 1` filter in `filterPreGeneratedRecipes` to allow recipes with fewer effects.
- Cache results of `getUsableEffectsForRecipe` for performance.
- Add error handling for missing or malformed `recipes.json.gz`.
- Clarify the `refresh(2)` call in `add` (likely should be `refresh(true)`).

This documentation covers the core functionality of `oblivion.js`. If you need further details on specific functions, additional debugging, or enhancements, please let me know!