Postman utilities

Defines a set of utility scripts for working with Postman. These scripts a written in Typescript, and designed to be compiled down to a single compressed file, which is then imported by postman during environment setup.

This project uses pnpm, but any package manager will do. You will need to modify the commands below, however.

Overview

Requirements

- Host machine with Visual studio code on it.
- node >= 16
- npm or pnpm or yarn

Contained in this repo:

- `src`: Typescript source files for the project.
  - `src/pm.d.ts`: Typescript definitions for Postmark app. Only defined to the degree that this project needs them. Should not be relied upon externally.
- `built`: Build output for js files after typescript compilation. Should only be a single output file.
- `test`: Mocha unit tests. Testing approach from here.
- `.mocharc.json`: For testing.
- `.nycrc.json`: For testing.
- `register.js`: for testing.
- `serve.js`: For serving built code locally, for testing changes.

Development

Initial setup: `pnpm install`. To build the output: `pnpm build`. To test the output: `pnpm test` To serve the built output locally at http://localhost:9999: `pnpm serve`. This URL can be then use in a task such as **Environment: DEV** in the URL bar. That setup endpoint will store the built script results locally for later reuse in the rest of the collection. This approach is for when you need to make modifications to utils, not necessarily when the endpoints need modification.

Currently, to import this script into Postman, you'll set up the following as a pre-request:

```
// MAIN
set_global_utils = (pm) => {
    pm.globals.unset('hebbia.utils_script_text');
    const r = pm.response;
    if (r.code !== 200) {
        throw new Error("Could not fetch Hebbia utils from postman-utils repo: "
+ r.code);
    }
    const script_text = r.text();

    eval(script_text);
    const utils = this.PostmanUtilsFactory.default(pm);
    if (typeof utils !== 'object') {
        throw new Error("Utils object not found as expected");
    }
    // Script text is valid, store it.
    pm.globals.set('hebbia.utils_script_text', script_text);
    hebbia.getUtils = () => utils;
    hebbia.hasUtils = () => true;
};

// Default accessor values.
// Mostly just a means of easily determining if utils are available.  The methods
// shown will be overridden by evaluated script.
// do NOT use 'var', 'const', or anything else here.  It messes up the scope, and
// will result in 'hebbia' being undefined for other callers.
hebbia = {
    hasUtils: () => false,
    getUtils: () => {
        throw new Error("Attempted to retrieve utils without calling Environment:
endpoint first.  More info: https://github.com/abeal-hottomali/postman-utils");
    }
}
// Use a header to allow Environment: calls to reset scripts.
if (Object.keys(pm.request.getHeaders()).includes('reset-postman')) {
    console.info("Resetting Postman");
    pm.globals.unset('hebbia.utils_script_text');
}
// Populate utils scripts if set.
// Default version will just throw errors.  eval() command will replace this.
let script_text = pm.globals.get('hebbia.utils_script_text');
if (typeof script_text === 'string') {
    // Replaces global utils object.
    eval(script_text);
    utils_factory = this.PostmanUtilsFactory.default;
    delete this.PostmanUtilsFactory.default;
    utils = utils_factory(pm);
    if (typeof utils !== 'object') {
        pm.globals.unset('hebbia.utils_script_text')
        throw new Error("Utils object could not be constructed.  Clearing utils
script text.");
    }
    // Script evaluated ok. Replace methods on global object with versions that
    // return utils.
    hebbia.getUtils = () => utils;
    hebbia.hasUtils = () => true;
}



/**
 * Prerequest: Called before every request in this collection.  Modify as needed.
 */
if (hebbia.hasUtils()) {
    const env = hebbia.getUtils().getEnv();
    hebbia.getUtils().prerequest(pm);
    // Add the Postman export version number, so the API can prompt for an
upgrade.
    pm.request.headers.add({
        key: 'version',
        value: env.get('version')
    });

    const x_portal_neutral = JSON.stringify({
        "caller": "admin-client",
        "subdomain": "admin"
    });
    // if (!pm.request.headers.has("X-Portal")) {
    //     console.info('No X-portal header defined, using ' + x_portal_neutral);
    //     pm.request.headers.add({
    //         key: 'X-Portal',
    //         name: 'X-Portal',
    //         disabled: false,
    //         value: x_portal_neutral
    //     });
    // }
    // Add Accept to every request if the script does not supply its own, so that
we always get JSON or JSONLD output.
    // simulating the the call came from the front-end client.
    if (!pm.request.headers.has("Accept")) {
        utils.getLogger().log('No Accept header defined, adding one to ensure
JSON or JSONLD output.', 'info', 2);
        pm.request.headers.add({
            key: 'Accept',
            name: 'Accept',
            disabled: false,
            value: 'application/ld+json, application/json'
        });
    }
}
```

Use this in your Environment setup script:

```
set_global_utils(pm);
const env = hebbia.getUtils().getEnv();
env.reset({
    'baseUrl': 'https://localhost:8443',
    'environment': 'development',
    'verbosity': 1,
    'enableBlackfire': false
});
```

Deployment

TODO