Supplementary Material: Document S1, Application Source Code Availability

This document provides instructions for obtaining the source code for the three case study applications as well as for using a starter template to create a new application using the Interactive Catchment Explorer (ICE) framework.

Case Study Applications

The following table provides links to the archived snapshots and active repositories for the source code of the three case study applications. The archived snapshots contain the version of the code as it existed at the time of publication. The code in the active repositories may differ as each application is updated following publication. Instructions for building each application is provided in the `README.md` file in each repository.

Application	Archived Snapshot	Active Repository
Northeast	https://doi.org/10.5281/zenodo.4058455	https://github.com/walkerjeffd/ice-northeast
Crown of the Continent Ecosystem	https://doi.org/10.5281/zenodo.4058453	https://github.com/walkerjeffd/ice-cce
Lower Mississippi-Gulf	https://doi.org/10.5281/zenodo.4058446	https://github.com/walkerjeffd/ice-lmg

Starter Template

For readers interested in creating a new application for other datasets and model outputs, we have created a starter template for the ICE framework. This template contains a working demo of the ICE framework that presents two example datasets in order to demonstrate the proper file structures and formats.

Please note that use and modification of this template requires some familiarity with modern web application development, JavaScript programming, and the use of the various third-party libraries (specifically vue.js, d3.js, crossfilter.js, leaflet.js, among others). We have attempted to make this template as simple as possible so that the number of changes to the source code are minimal. The bulk of the effort is related to processing the data into the appropriate formats and creating the configuration files that define the datasets and variables.

Download Source Code

Similar to the case study applications, the following table provides a static snapshot of this template as well as a link to the active repository. The `README.md` file in this repository contains intstructions for using the template, similar to what is provided in this file.

Application	Archived Snapshot	Active Repository
Starter Template	https://doi.org/10.5281/zenodo.4058490	https://github.com/walkerjeffd/ice-template

To obtain the template code as it existed at the time of publication, navigate to the archived snapshot URL, and download the zip file containing the complete source code (`ice-template-v1.0.0.zip`). Unzip this file to your target working directory.

Prerequisites

To develop a new ICE application, you will need to install Node.js (v12 or newer) as well as some global packages.

Instructions for installing Node.js on Windows, Mac or Linux can be found at https://nodejs.org/en/.

After installing Node, use the node package manager (`npm`) to install the Vue CLI and http-server.

```
npm install -g @vue/cli http-server
```

Quick Start – Running the Demo

Complete the following steps to run the demo application, which includes two example datasets. See `data/scripts/README.md` for details about these datasets and how they were created.

Project setup

Install dependencies defined in the `package.json` file.

```
npm install
```

Run Development Server

To start a development server that runs the demo application, run the `dev` command using `npm`, which serves the `data/` folder at port `8000`, and runs the vue CLI `serve` command.

```
npm run dev
```

Then open your browser and navigate to http://127.0.0.1:8080/. This page should contain the demo application for the two example dataset (Northeast, Northwest).

Building a New Application

This section describes the primary steps for adapting this template to new datasets.

Step 1: Generate Themes

A `theme` in the ICE framework contains the configuration, data, and geospatial data associated with a single dataset.

Each theme is comprised of the following files:

```
theme.json          - configuration file
data.csv            - CSV file containing the dataset
layer.json          - GeoJSON file containing the spatial features
features/<ID>.json - JSON files containing additional data for each individual feature
```

These files are organized by subfolders within the `data/` folder. The files for each theme must be stored within a subfolder named after the theme id (e.g., `data/northeast/` contains the files for the `northeast` theme).

theme.json

Each `theme.json` must contain attributes for the `id`, `label`, `description`, and `variables`.

```
{
  "id": "northeast",
  "label": "Northeast Example",
  "description": "Example dataset of the northeast U.S.",
  "variables": [
    ...
  ]
}
```

The `variables` attribute must be an array that defines a series of properties for each variable in the dataset.

Numeric variables must have the following format.

```
{
```

```
  "id": "variable1",          // must match column name in the data.csv file
  "label": "Variable 1",      // how the variable is displayed in the interface
  "description": "A dummy variable from 0 to 1",
  "units": null,              // units (optional)
  "type": "num",              // must be "num" for numeric variables
  "group": "Variable Group", // the group this variable belongs too
                              // (dropdown menus list variables by group)
  "map": true,                // true if this variable should be an option for the map variable
  "filter": true,             // true if this variable should be an option for the crossfilters
  "formats": {                // format specification for displaying values
                              // see https://github.com/d3/d3-format#locale_format
    "value": ".1%",           // for writing a single value (e.g. tooltip)
    "filter": ".1%"           // for the crossfilter axis labels
  },
  "scale": {
    "domain": [0, 1],         // 2-element array defining min/max values of this variable
    "transform": "linear"     // transformation type, can be "linear" or "log"
  }
}
```

Categorical variables must have the following format.

```
{
  "id": "variable3",          // must match column name in the data.csv file
  "label": "Variable 3",      // how the variable is displayed in the interface
  "description": "A categorical variable", // additional description
  "units": null,              // null for categorical variables
  "type": "cat",              // must be "cat" for numeric variables
  "group": "Variable Group", // the group this variable belongs too
                              // (dropdown menus list variables by group)
  "map": true,                // true if this variable should be an option for the map variable
  "filter": true,             // true if this variable should be an option for the crossfilters
  "formats": {
    "value": null,            // null for categorical variables
    "filter": null            // null for categorical variables
  },
  "scale": {
    "domain": [               // domain is an array that maps each unique value in the dataset to a label
                              // (e.g. `A` -> `Value A`)
      {
        "value": "A",         // value used in data.csv
        "label": "Value A"    // label for displaying this value
      },
      {
        "value": "B",
        "label": "Value B"
      },
      {
        "value": "C",
        "label": "Value C"
      }
    ]
  }
}
```

layer.json

The `layer.json` file defines the geospatial data (i.e., coordinates) of the spatial features using GeoJSON format. This file must use the WGS84 (EPSG 4326) projection (latitude/longitude).

Each feature must contain a unique `id`, which corresponds to the `id` column in the `data.csv` file.

Note that in this starter template, the ICE framework is only configured to render point geometries. However, alternative geometries (polylines, polygons) could be supported by modifying the render functions within the `src/components/IceMapLayer.vue` component.


```
{
  "type": "FeatureCollection",
  "name": "layer",
  "crs": {
    "type": "name",
    "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" }
  },
  "features": [
    {
      "type": "Feature",
      "id": "170501090104",
      "geometry": {
        "type": "Point",
        "coordinates": [ -117.939631420651409, 42.590973510329043 ]
      }
    },
    {
      "type": "Feature",
      "id": "170501041304",
      "geometry": {
        "type": "Point",
        "coordinates": [ -116.674185426499221, 42.454648318749527 ]
      }
    },
    ...
  ]
}
```


data.csv

The `data.csv` file contains the theme dataset in comma-separated values (CSV) format. Each row contains the values for one spatial feature, and each column represents a different variable. This table must include a column named `id`, which corresponds to the `id` field of the GeoJSON layer. For each of the other columns, the `theme.json` file must contain a variable definition with the `id` attribute equal to the column name (see above).


```
id,variable1,variable2,variable3
170501090104,0.24387659435160458,0.2369458530249091,B
170501041304,0.6368760853074491,1.877845762558082,B
```

```
170402080305,0.7774140483234078,-0.1166326835861824,A
170402100904,0.05460463068448007,0.6259004795984403,A
170501041305,0.21677141194231808,-1.0887155657567542,A
170402060805,0.9966927575878799,-0.06722108592500253,C
...
```

features/ID.json

Lastly, the `features/` subfolder within the theme contains a collection of JSON files, one for each unique spatial features. Each file must be named using the `id` of the corresponding feature (same as the `id` column in `data.csv` and the `id` attribute in the `layer.json` file).

Each file must be in JSON format, and all files should have an identical structure. The purpose of these files is to provide all data associated with a each feature. This may include data that is not part of the `data.csv` (e.g., timeseries).

In the demo application, each feature details file simply contains the values of the three variables as well as the feature id.

```
{
  "id": "010100020101",
  "variable1": 0.6696,
  "variable2": 0.1116,
  "variable3": "C"
}
```

Whenever a feature is selected on the map, the corresponding details file will be loaded and made available to the theme component (see next section).

Step 2: Create Theme Components

For each theme, create a new vue component in the `src/components/themes/` folder. This component is used to display the details of a selected feature in the right-hand side bar when a feature is clicked and the corresponding feature details file (`data/<theme>/features/<id>.json`).

For the demo datasets, each component (`src/components/themes/Northeast.vue` and `src/components/themes/Northwest.vue`) simply provides a pair of tables listing the values of each variable. However, these components can be enhanced to include other data and charts. For more complex examples, see the walkerjeffd/ice-lmg source code.

The name of each theme component must be a CamelCase version of the theme id (e.g., `northeast -> Northeast.vue` or `my-dataset -> MyDataset.vue`).

Step 3: Themes List

The list of available themes is stored in the `src/assets/themes.json` file. This file must contain an array of folders. Each folder has an `id` and `name`, as well as an array of `children`, which contain the themes for that folder. Each theme must have an `id`, `name`, and `description`. Remember that the theme `id` must match the name of the folder in `data/` containing the files for that theme.

To add a new theme, add a new entry to the `src/assets/themes.json` file.

```
[
  {
    "id": "project",
    "name": "Example Projects",
    "children": [
      {
        "id": "northeast",
        "name": "Northeast Dataset",
        "description": "Example dataset of the northeast (Region 01)"
      },
      {
        "id": "northwest",
        "name": "Northwest Dataset",
        "description": "Example dataset of the northwest (Region 17)"
      }
    ]
  },
  ...
]
```

Step 4: Content

Update the text content in the various dialog boxes as well as the application title in the `src/App.vue` file.

Step 5: Run Development Server

Once the files and components for the themes have been created, run the development server to view the application.

```
npm run dev
```

Then open your browser and navigate to http://127.0.0.1:8080/. This page should contain the application with your new themes avaible in the Dataset Browser.

Build Process

The following steps describe how to build the production version of the application.

Step 1: Configuration

Build configuration is set using `.env` files for each environment (`development` , `production`). See vue-cli for details.

There are two required configuration variables:

```
BASE_URL=/                              # root path for the application
VUE_APP_API_URL=http://localhost:8000/  # location for fetching data
```

The `BASE_URL` variable defines the initial URL path of the application when it is deployed to the server. For example, if the target URL is http://example.com/app/ice/ then set `BASE_URL=/app/ice/` .

The `VUE_APP_API_URL` variable defines the root URL where the data will be stored, which may differ from where the application files are deployed. For example, if the data are accessible at http://example.com/data/ice/theme-id then set `VUE_APP_API_URL=http://example.com/data/ice/` .

The default `.env` files can be overriden with `.local` variants (e.g. `.env.development.local`), which are not tracked by the repo.

For production, create a new `.env.production` (or `.env.production.local`) file and define the `BASE_URL` and `VUE_APP_API_URL` .

Step 2: Production Build

Run the `build` command to build the application for production. This command will bundle all of the application files, and write the output to the `dist/` folder.

```
npm run build
```

Step 3: Deployment

After building the application, copy the contents of `dist/` and `data/` to the appropriate folders on the production server.

Disclaimer

Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.