**Joint Global Change Research Institute (JGCRI)**

**Community Emissions Data System (CEDS): Guide for New Project Members**

Project Head: Steve Smith

Written by Emily Voelker, Tyler Pitkanen, Jon Seibert, and Rachel Hoesly.

Last updated: August 31, 2015.

Table of Contents

1. Background 2

2. Data Plan 3

3. Version Control 4

Git 4

GUI 5

Command Line 5

4. File System Layout 5

5. System Flow 6

6. Prerequisites 7

Package Installation: 7

Proprietary Data: 7

7. Running Individual Scripts in R 8

8. Running the System with a Makefile 8

Running in Mac OS 8

Running in Windows 10

9. Troubleshooting with Libraries 12

In Mac OS: 12

In Windows: 13

10. Troubleshooting with Java 13

In Windows: 13

In Mac OS: 14

11. Naming/Numbering Conventions 14

R Script File Naming 14

Output File Naming 15

Makefile Code block Numbering 16

12. Making Changes 16

13. CEDS-Specific Coding and Comment Conventions 17

Function Header Comments 17

Script Header Comments 18

Script Header Code 18

Use of Basic CEDS IO Functions 19

Logging 19

Metadata 20

Diagnostics 20

14. Module B & C Overview and Data Addition Instructions 21

Overview: 21

Adding data: 21

Adding a new emissions species: 22

15. Module E Overview 22

The purpose of this document is to bring new interns and project members up to speed on the current progress of the emissions data system.

# Background

The emissions data system aims to provide time-series country (sometimes sub country) emissions estimates for greenhouse gases, anthropogenic acidifying gases (NH3, NOX, SO2), carbonaceous gases (CO2 , CH4, and CO) and carbonaceous aerosols (BC and OC). In its completed form, it will consist of a number of modules, as outlined in Figure 1.

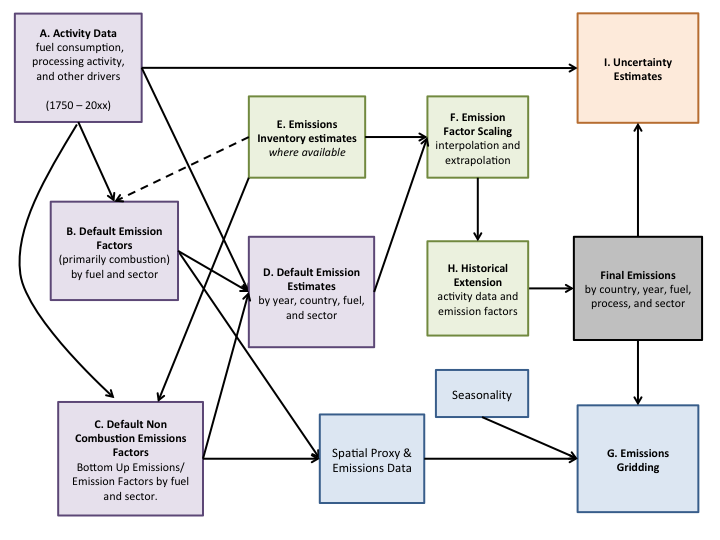


Figure 1 CEDS module flow.

The project was originally begun using GCAM resources and shares similarities and some sets of functions. The starting point was Module A, processing fuel and energy consumption data. Because of this, Module A does not follow the same file naming convention as other Modules.

# Data Plan

When it is complete, CEDS will use many data sources there are a few types of data sources.

* Activity Data:
  + Energy: IEA, BP Energy
  + GDP
  + Population: UN, World Bank
* Inventories used for default emission factors. Detailed multiyear inventories available by country, sector and fuel.
  + USNEI, GAINS, SPEW (bond)
* Supplemental country emissions inventories: inventories that are not available by sector and fuel. Mostly used for validation and scaling.
  + EDGAR, REAS, country inventories (UNFCC)
* Historical sources (pre 1950) to inform emissions factor trends
  + Too many to list. Mostly very specific sources.

# Version Control

CEDS project now uses Github for version control.

Repository URL: <https://github.com/JGCRI/CEDS.git>

This is a private repository. If you do not have permissions, contact Ben Bond-Lamberty ([BondLamberty@pnnl.gov](mailto:BondLamnerty@pnnl.gov)).

## Git

Basic guides to get started with github: <https://guides.github.com/>

Github works similarly to svn with a few key differences. Instead of logging differences like svn, git tracks snapshots of the project at each point in time. Instead of changing/adding a document then committing to the project, workflow for using git is, in general, as follows:

1. Edit/add documents
2. Stage/cache the files to get ready to commit
3. Commit the changes with comment to the local repository
4. Push the commits to the remote repository

Comment on each commit, like before. You can Push multiple commits, so you don’t have to push after every commit. It’s good practice to commit often with comments then push multiple commits when you need to.

Like updating the repository with svn, you “pull” changes from the remote repository to update your working copy.

Git documentation: <http://git-scm.com/doc>

Github works from the command line like terminal (mac) or Powershell(windows) but it’s easiest to download Github GUI. The installer includes both the command line version of Git as well as the GUI. Sourcetree is another good GUI option, available for both mac and windows.

## GUI

GitDesktop:

<http://windows.github.com>

<http://mac.github.com>

GitHub desktop works well for visualizing changes, and simple actions like committing or starting a new branch. More complicated actions will require use of the command line.

SourceTree:

<https://www.atlassian.com/software/sourcetree/overview>

Setting up with SourceTree:

Under New Repository, click clone respository from URL. Fill in the url from above, the file path to a non existent folder titled CEDS (likely \Documents\CEDS), and fill in passwords when prompted.

## Command Line

There is a lot more documentation on the internet for using git with the command line than there is for using git with any GUI. If you’re doing anything more complicated that really simple changes then you probably need to learn to use git through the command line.

Git Cheat Sheet: https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf

# File System Layout

The main directory (and base for the git checkout) is named emissions-data-system. Within this folder, there are seven subfolders and the Makefile.

* The “**code**” directory contains all the R scripts currently integrated into the system, separated into modules A-E and the “parameters” directory, which contains header files and scripts with helpful functions.
* The “**diagnostic-output**” directory contains any and all diagnostic files created by system scripts.
* The “**documentation**” directory contains all available information about the system itself, including this file.
* The “**exe**” directory contains the executables to run the system or scrub the output folders, and all the necessary files.
* The “**final-emissions**” directory will contain the final output(s) of the system, once they exist.
* The “**input**” directory contains all integrated input data for the system scripts, separated by category.
* The “**intermediate-output**” directory contains all outputs of the system scripts to be used by later scripts in the sequence.

A visual mapping of the file system organization is available under the “File System Chart” tab of system\_documentation.xlsx, in the “documentation” folder.

# System Flow

In general:

1. Collect and process driver data (ex. fuel use) to estimate sector level emission factors
2. Use trends in driver data to estimate emission factor trends (historical and recent years)
3. Calculate default emissions factors from activity data and emission factors.
4. Detailed emissions inventory data is used when available and judged to be reliable (ex. US NEI (United States National Emissions Inventory))
5. Estimate uncertainty throughout

NOTE: Non-Combustion Emissions = Process Emissions. Be careful to note the difference between “processing” data (the action of manipulating data) versus process emissions or process activity data (non combustion emissions/data)

|  |  |
| --- | --- |
| Module | Description |
| A | Loads and Processes IEA and BP energy data, the primary energy activity Data used in CEDS |
| A1 | Energy: Takes in raw IEA energy data and splits it by country |
| A2 | Energy: Splits and maps to CEDS fuels and sectors |
| A3 | Energy: Extends energy data with BP data, Extrapolates data to more recent years |
| A4 | Energy: Expands data to include entries for all possible ids and countries |
| A5 | Load and process and combine other Activity Data |
| B | Calculate combustion Emissions Factors (EF) |
| C | Default Non Combustion Emissions and EF by country, fuel, sector – estimate default fuel emission factors for different emission species mostly from specific inventories |
| C1 | Bottom up Non Combustion Emissions |
| C2 | Calculate NC emissions factors |
| D | Calculate and Combine Combustion and Non Combustion Emissions and EFs |
| D1 | Calculate Default Combustion Emissions |
| D2 | Combine Non Combustion and Combustion Emissions |
| D3 | Combine Non Combustion and Combustion Emissions Factors |
| E | Emission Inventory Estimates |
| F | Emission/Emission Factor Scaling |
| H | Historical Extension |
| H1 | Extend Activity Data |
| H2-3 | Extend Emission Factor Data |
| H4 | Final Corrections |
| S | Summarize final data |

# Prerequisites

## Package Installation:

R code in CEDS require additional R packages that must be installed and loaded. Packages must be installed once (while connected to the internet) with the install.packages() function and loaded every time a new R session is started with the require() or library() functions.

R scripts in the system will automatically load packages, but the necessary packages will need to be installed prior to running the system the first time. If a packages is required, but not installed a warning message will appear.

The following packages are loaded in section 0 of every script, when the global\_settings.R script is sourced.

c( "ggplot2", "magrittr", "pbapply", "plyr", "dplyr", "reshape", "stringr", "XML", "readxl", 'zoo', 'gridExtra')

When installing/loading some of these packages, they will automatically load/install other required packages. However, ensuring that the packages listed above and below are installed, should be sufficient for the system to run.

The following packages are loaded in individual R scripts that call them:

c(“FAOSTAT”, “XLConnect”, 'scales')

This is not an exhaustive list.

## Proprietary Data:

In order to be able to run the CEDS system as a whole, one must acquire a copy of the data files “OECD\_E\_Stat.csv” and “NonOECD\_E\_Stat.csv” (and their metadata files) from the JGCRI shared server and place them in the “emissions-data-system/input/energy” directory. They are required by the script A1.2.IEA\_downscale\_ctry.R, but are proprietary data and not allowed to be a versioned part of a public-domain system such as CEDS.

The files can be found under the “CEDS/OECD Energy Data” directory on the JGCRI shared server. Project members without access to this server must acquire a copy from another project member. There is not yet a system in place to allow outside users access to this data for the purposes of running CEDS.

# Running Individual Scripts in R

Individual scripts can be run in R directly, and edited in whichever text editor you prefer (note that all R scripts must have the file extension .R or .r). Working directory settings and required source files are handled at the top of each individual script, with a standardized header and a section to import the required header and function files, as well as any of the optional function files available.

To run an individual script, you must have R (current version is 3.2.1) installed on your computer, as well as one of the GUIs (RGui or RStudio). If the script you wish to run is open within the GUI, simply select the entire file (Ctrl-A), or the sections you wish to run, and press Ctrl-R to run it. If it is open in an external text editor, copy the script and paste it into the console section of your GUI.

# Running the System with a Makefile

To run the entire CEDS system, simply navigate to the “exe” subfolder of the system and run the “make-all.bat” executable. The Makefile system will detect any changes made and re-build the outputs as necessary. If the system is up to date, it will do nothing. To force the entire system to run again, run the “make-clean.bat” executable to remove all intermediate outputs and log files, forcing the Makefile system to build the system from the first output file again, and running all integrated scripts.

All modules are included in a Makefile in the emissions data system. Running the modules through the Makefile is advantageous because “make” will automatically run only what needs to be run to keep everything properly updated.

## Running in Mac OS

1. Navigate to the emissions-data-system folder using “cd” and the path to the emissions-data-system directory. This is likely:

cd /Users/<user name>/Documents/CEDS

1. Run the “make” command

make

To rerun the entire system use the “make clean-all” command, then the “make” command

make clean-all

make

Further Explanation:

To run the make file, navigate within terminal to the emissions-data-system folder (the folder you checked out through the git repository), and run the command “make”.

To navigate to the CEDS:

The command “pwd” will show you the path to the current directory.

The command “cd” followed by nothing will navigate you pack to the default home directory. This is likely your user directory with the path “/User/<user name>.

Change the directory to the emissions-data-system folder using the command “cd” followed by the path to the directory. For example

cd /Users/<user name>/Documents/CEDS

You can check to make sure that you have navigated to that folder by using the command “pwd” to show you the path or the command “ls” which will list the items in directory you have navigated to. If you navigated to the right directory, “ls” should ouput:

Makefile diagnostic-output exe input

code documentation final-emissions intermediate-output

which are the contents of the CEDS folder.

To run the system, type the command “make” which will run the makefile in the directory you are in.

For Example: On Rachel’s computer:

cd /Users/hoes919/Documents/CEDS

make

Notes on first time use and setup:

You may need to install Command Line Tools and/or make. This can both be done by installed Xcode, free from the Apple App Store. It can also be done using HomeBrew.

Open Terminal (found in the application folder)

Type the command “make” into the terminal window and hit enter. If terminal gives you an error other than

\* No targets specified and no makefile found. Stop.

you need to install the package.

Whithin Xcode, you can install Command Line Tools by selecting “Preferences”, clicking the “Download” tab, clicking “Components”, Clicking “Install” on the command line tools line.

## Running in Windows

There are a number of options for running a Makefile in Windows. The “make” functionality is not native to the Windows operating system, so it must be downloaded. There are various options for doing this, but one of the simplest to operate is through the DOS shell, and the .bat files included with the system.

**DOS:**

In order to utilize this method, it is necessary to acquire copies of the necessary .exe files from the JGCRI shared server. They are not included with the system in order to allow for cross-OS compatibility at a later date.

1. Download make functionality

You will need a JGCRI login separate from your computer to log in to the shared server. If you do not have one, speak to your resident tech guru.

First, go to: [//jgcri.umd.edu/share/IA Modeling Files/AgLU/AgLU Data System/Additional Windows files](file:///\\jgcri.umd.edu\share\IA%20Modeling%20Files\AgLU\AgLU%20Data%20System\Additional%20Windows%20files\)

Copy the folder “needed-exes” and paste it with your other project files on your computer hard drive. It is highly recommended that you place them within the “exe” folder of the system, but not absolutely necessary.

1. Specify Paths

Because “make-all.bat” and “make-clean.bat” are .bat files, Windows will automatically try to run them instead of opening them, even if you right-click on it and select “Open”. In order to edit the files, open WordPad or a similar text editor, and select the file from the “Open” browser. If you have Notepad++ installed, there should also be an “Edit with Notepad++” option in the right-click popup menu.

You will need to specify the location of the “needed-exes” folder within “make-all.bat” and “make-clean.bat”- ex:

SET PATH=%PATH%;C:\Users\Seib306\Documents\emissions-data-system\exe\needed-exes;C:\Program Files\R\R-3.2.1\bin\x64;

Note: this set path also worked (don’t know why)

SET PATH=%PATH%;C:\Users\Seib306\Documents\emissions-data-system\exe\needed-exes\bin;C:\Program Files\R\R-3.2.1\bin\x64;

If a SET PATH line does not already exist, create one. This must always begin with SET PATH=%PATH%; in order to avoid overwriting other paths needed by the shell. The first path you specify should point to the “bin” folder within the “needed-exes” folder. The second must point to the x32 or x64 folder where your version of R is installed. A template is as follows:

SET PATH=%PATH%;<path-to-exes>;<path-to-R>;

In addition, if you had to set the variable R\_LIBS\_USER for an R User Library (see *Troubleshooting with Libraries* below), you must also do so here, in the same format:

SET R\_LIBS\_USER=<path-to-lib>;

Ex: SET R\_LIBS\_USER=C:\Users\Seib306\Documents\R\win-library\3.2

You can also change the target being made if necessary- simply replace “all” or “clean” in the line to set CMD with the desired target. This is generally a bad idea- it would be better to simply copy the batch file and make a new “run-\_.bat” for a new Make target.

Once this is complete, the system can be run in its entirety by executing “run-all.bat”. Ensure that your Makefile is up to date and is using the correct references, or else it is liable to run all the programs whether or not anything is out of date, or to simply break.

**Other options for Make include:**

* **Cygwin**: <https://cygwin.com/install.html>
  + In order to ensure the proper files are downloaded with Cygwin during installation:
    - On the Select Packages screen, under All->Devel, ensure that the Bin box is checked for the file labeled “make: the GNU version of the ‘make’ utility”. Src is not necessary.
    - In addition, if you wish to utilize the emacs editor, it can be selected under All->Editors.
  + This option also provides an alternative command environment. This can be accessed by running Cygwin.bat after downloading.
  + When installing this, the user is given the option to install other command line functionalities as well (e.g. R commands like Rscript and R CMD BATCH, or gcc functionalities).
* **GnuWin32**: <http://gnuwin32.sourceforge.net/packages/make.htm>
  + This option will provide the “make” command to the Windows command prompt
* **VisualStudio**
  + This option will provide the “nmake” command to the Windows command prompt

For these other options:

Makefiles can be run by opening the command prompt, pointing to the location of the Makefile, and entering “make” (or “nmake”) into the command prompt. This will run the Makefile in the specified directory location. To point to the appropriate directory, commands such as “cd” and “dir” can be used. An example process is shown below (note the “>” indicate entered code; they are not typed by the user).

> cd C:/users/Seib306/documents/emissions-data-system

> make

Other commands, for example commands for running R files, are also not native to Windows. These must be downloaded separately as well, for example during the installation of Cygwin. Commands like “Rscript” or “R CMD BATCH” are necessary, as they are used in the Makefile. These commands can also be used in the command line independently to run specified individual scripts, if desired.

# Troubleshooting with Libraries

At present, the R code modules we are using require several packages to run properly: ex. “ggplot2”. If these packages are not already present in your R “Library”, the system should attempt to install them for you. However, due to system permissions, it is likely that you will be denied permission to install them into the default library.

## In Mac OS:

As of 8.7.15 the automatic package installation is not working on macs. The solution right now is the manually install the packages used in the system using the install.packages() function. The list of packages used can be found near the top of the global\_settings.R file in the parameters folder.

## In Windows:

If this occurs, the system should offer you the option to create a “personal” or “user” library. Do so. You must simply choose a location for R to designate as your User library, and it will do the rest.

Afterwards, run the command “.libPaths()” to check that it properly added the new library to the system. It should print out the file path for the original library, under Program Files, and the new User library that you just added.

If the new library does not show up, search for the file entitled “.Renviron” or just “Renviron”. Open it with your preferred text editor: WordPad seems to be best thus far. Once open, scroll down to the entry for R\_LIBS\_USER, if there is one. If it exists, remove any comment operators (#) preceding it, and edit the designation to your new User library. If it does not exist, create it.

ex. R\_LIBS\_USER=${R\_LIBS\_USER-'C:/Users/Seib306/Documents/R/win-library/3.1'}

This should ensure that R recognizes the user library, and will know to look there for packages and install them there if necessary.

# Troubleshooting with Java

Some of the packages used in CEDS require the interface package “rJava”. If you experience errors when loading or installing rJava relating to “missing” or “unable to load” executables (.dll files), follow these steps to restore functionality:

## In Windows:

1. Ensure you have installed the most up-to-date versions of R, Java, and the rJava package, and that R and Java are matched to the bit level of your OS. (I.e. all are 32-bit if 32-bit Windows, or all 64-bit if 64-bit Windows.) To determine bit level, look under “System Type” in the System panel of the Control Panel.

* **R:** Version displayed on GUI startup and in the full filename of the RGui or RStudio executable. To download the most recent version, visit: <https://cran.r-project.org/bin/windows/base/>
* **Java:** Version displayed in folder names within “Java” folder, usually located under C:\Program Files. To download the most recent version, visit: <https://www.java.com/en/download/manual.jsp#win>
* **rJava**: Run the command “install.packages(‘rJava’)”.

1. Ensure that jvm.dll and rJava.dll are in the system PATH variable, and that JAVA\_HOME is set correctly.
   * Open the menu for environment variables: Start -> Control Panel -> System -> Advanced System Settings -> “Advanced” tab -> Environment Variables
   * Under “User Variables”, JAVA\_HOME should point to the directory where your most recent version of Java is installed. (ex. C:\Program Files\Java\jre1.8.0\_51 )
   * Under “System Variables”, PATH (or Path) should have multiple entries in it, separated by semicolons (;).
     + Find jvm.dll within the directory specified by JAVA\_HOME. It will likely be within the subfolders “\bin\client” or “\bin\server”. If there is no entry pointing to a jvm.dll in PATH already, add this full filepath to it. If there is one, replace it. (ex. C:\Program Files\Java\jre1.8.0\_51\bin\server\jvm.dll; )
     + Find rJava.dll within the rJava folder in the library where your R packages are installed. This will either be a user library that you chose the location for, or the system library under C:\Program Files\R. If there is no entry pointing to an rJava.dll in PATH already, add this full filepath to it. If there is one, replace it. (ex. C:\Users\Seib306\Documents\R\win-library\3.2\rJava\libs\x64\rJava.dll)
2. Restart your computer.

## In Mac OS:

1. Make use of this website: <https://www.google.com>

# Naming/Numbering Conventions

## R Script File Naming

**Modules B and C**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module Letter and # | Function # and Label | Species | Emission Type | Data Type |
| A1, A2… | 1.base | SO2 | comb | emissions |
| B1,… | 1.default | BC | NC | EF |
| C1,C2,C3,C4… | 2.add |  | total | activity |
|  | 3.proc |  |  |  |

All R Scripts should include:

* Module letter and number (ex: C1)
* Function Number and label (ex: 2.add)
* Emission Species and Type if applicable (not applicable to activity data) (ex: SO2\_NC)
* Data Type (ex: emissions)
* Other identifiers if necessary (ex: the type of activity data)

In the following format:

[X#].[#.Function]\_[species]\_[emissions type]\_[data type]\_<other identifier if needed>.R

More detail on #.Function types can be found in Section 10 on page 14.

example:

C3.2.add\_SO2\_NC\_emissions\_all.R

**Modules D**

[Country]\_[data source]\_[emissions type]\_<other identifiers>.R

## Output File Naming

Note: Module A does not follow this naming convention

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module Letter | Species | Emission Type | Data Type | Other identifier |
| A | SO2 | comb | emissions | db |
| B | BC | NC | EF | metadata |
| C |  | total | activity |  |
| E |  |  |  |  |

All R Scripts should include:

* Module letter. Do not include the module number. Output files are named for the module they are created in, NOT the subsequent module they will be used in.
* Emission Species and Type if applicable (not applicable to activity data) (ex: BC\_comb)
* Data Type (ex: emissions)
* Other identifiers if necessary (ex: the type of activity data)

In the following format:

[X].[species]\_[emissions type]\_[data type]\_<other identifier>.csv

## Makefile Code block Numbering

Makefile code blocks are numbered to reflect the module letter and number (where possible), and their order in the makefile. They do not reflect the function number in the R script name) More information on Makefiles can be found in comments in the actual makefile (titled “makefile” in the emissions-data-system folder.

While modules and scripts are labeled with:

<module letter (capital)><module number>.<function number> (ex: C1.2.add\_activity\_gdp.R)

make file code blocks are labeled with:

<double module letter (lower case)><module number>-<block ordering/sequence> (ex: cc1-1)

For example: All the C1.1 and C1.2 scripts are called in the same code block labeled cc1-1.

As noted in the comments in the make file, sometimes calling a single R script will require 2 paragraphs/blocks in the make file, if a single R script has multiple outputs (targets). If this is the case, these blocks are noted with the same number followed by lowercase a or b.

For example: C1.3.proc\_activity.R is called by the second block for module C1 and required 2 paragraphs, so they are labeled cc1-2a and cc1-2b. Note – the number following the dash notes the order of the block in the c1 section of the makefile (2), not the function number of the R script (3).

# Making Changes

## General Procedures

All project members are required to seek approval and a code review for all changes made to the system before committing them to the master branch. In addition, all project members must keep the “system\_documentation.xlsx” and “CEDS\_code\_change\_tracker.xlsx” files updated with descriptions of the changes made any alterations or additions reflected appropriately.

If you are coding any additional scripts, they should be written and presented according to the R style guide; you can also emulate the style of previous scripts (but please read the style guide first). As you write new code, too, be sure to be very meticulous in your commenting, so that those following you can understand what you’ve done.

In addition to commenting, it may be beneficial to keep a progress journal, to help you

remember what changes you’ve made, and why. This can be especially helpful if you need to re-address a change made past your memory.

## Domain Names and File Mapping

To make changes to the CEDS system, careful attention must be paid to organization and mapping files. In additional to simply using file path names to source files, there is a mapping file called domainmapping.csv that assigns a *domain name* to key emissions-data-system folders. This is used so that code can be written without detailed information about the organization of input files. If you change the arrangement of the folder system, you must make sure to then make changes to the mapping file domainmapping.csv.

For example, say you need to alter file “IEA\_product\_downscaling.csv” slightly from the original version, so you copy and paste it from the original location and put it the “energy\_data” folder, and then edit it. It was originally located in “emissions-data-system/input/mappings”. In the “domainmapping.csv” file, this location’s domain is called “MAPPINGS”. So having made that change, you must go into the code and make it read from the “ENERGY\_DATA” domain, instead of the “MAPPINGS” domain, whenever you want to use your altered version of “IEA\_product\_downscaling.csv”.

In another situation, perhaps the “energy\_data” folder gets too messy, and you want to add subfolders to better organize it. You would also have to go into “domainmapping.csv” and edit the pathways, and the new names of the folder. For example, you might add a subfolder called “bp\_data,” and put domainmapping.csv and “BP\_energy\_data.xlsx” within it. Now, you have to go into “domainmapping.csv”, and create a new domain for this folder. You’d give it a pathname to direct towards it, and a name like “BP\_DATA”. And then, in the code, you’d have to make sure it sources that file from “BP\_DATA”—instead of the “ENERGY\_DATA” directory they had formerly used.

Note that it is not necessary to add an entry to the domain mapping system for every new folder. For emissions data, in particular, there are many data sources, so it is preferable not to add an entry to the domain mapping system for every new sub-folder. Instead, prefix the file names with the subfolder name within the appropriate domain. Emissions files for Canada, for example, might be placed in a “Canada” sub-folder inside the EM\_INV domain. Prefix file names with “Canada/” and supply the domain EM\_INV.

Large files or groups of files can also be stored as zip files. R can read directly from zip archives.

**Note: all file pathways should be specified using a forward slash “/”.** This ensures that code will function properly on windows and unix systems. Do not use a backslash “\”.

# CEDS-Specific Coding and Comment Conventions

## Function Header Comments

Whenever defining a significant function that will be used multiple times (i.e. most functions in one of the function header files), it must be documented with a standard set of header comments. These must consist of:

* Function Name
* Brief (short description)
* Details (full description)
* Dependencies (other functions or header files required for the function to work)
* Author(s)
* Parameters (description and use of each parameter of the function)
* Return (the return value of the function, if any)
* Input Files (if any)
* Output Files (if any)
* Notes (any other information of note)
* Usage Examples (one or two example function calls)

Any sections that cover multiple lines should be indented in the second line and onwards. The parameters listed below the “Parameters” heading should be indented.

See any of the function header files (ex. process\_db\_functions.R) for examples.

All functions in function header files must also be documented in the “Header Functions Allocation” tab of System\_Documentation.xlsx.

## Script Header Comments

Whenever creating a new R script to add to the CEDS system, it must be documented with a standard set of header comments. These must consist of:

* Script Name
* Author(s)
* Date Last Modified
* Script Purpose (description of the primary functionality of the script and its outputs)
* Input Files (if any)
* Output Files (if any)
* Notes (any other information of note)
* TODO (any remaining work to be done refining or altering the script)

See any CEDS R script for examples.

There are also standards for naming the script itself, as well as any variables and functions declared- see section 9 (Naming/Numbering Conventions).

## Script Header Code

Immediately following the header comments in each CEDS R script, there must be a block of code (section 0) that finds the “emissions-data-system/input” directory and sets the working directory to that location, as well as defining PARAM\_DIR as the location of the “emissions-data-system/code/parameters” directory. These steps are necessary to ensure that auxiliary function files and input files can be located by the CEDS read and source functions.

Following this, there are places to assign the name of the script, the first message to print to the log, and any auxiliary function files required, each to its own variable. The script must then source the “header.R” file and run its initialize() function using that input, in order to start the subsystems that allow logging and easy input/output. “global\_settings.R” and “IO\_functions.R” are always included in the auxiliary files to be sourced, and “common\_data.r” is included by default.

Every CEDS R script must duplicate this functionality. There exists a template for the basic form, named “R Script template.R” in the documentation folder- when making a new script, it is easiest to copy the template and fill in the initialization variables and description as necessary.

Section 0 is also the place to load any extra packages (not included in global\_parameters.R)

## Use of Basic CEDS IO Functions

The readData() and writeData() functions have been designed to be as flexible as might be required. With the exception of a final output of the system, all data reads and writes in CEDS R scripts should use these two functions.

Standard use of readData() involves a domain name, a file name, and (if reading an .xlsx file) an extension and sheet name. The default extension is .csv, so it is not necessary to specify that when reading a .csv file. Giving no sheet name causes readData() to read all available sheets and return a list of data frames, rather than a single data frame, if there are multiple.

Reading a .csv: readData( “MED\_OUT”, “A.energy\_data” )

Reading one sheet from an .xlsx:

readData( “MAPPINGS”, “Master\_Fuel\_Sector\_List”, “.xlsx”, “Fuels” )

Reading all sheets from an .xlsx:

readData( “MAPPINGS”, “Master\_Fuel\_Sector\_List”, “.xlsx” )

As writeData() only outputs .csv files, the standard use requires only a data frame object, a domain name, and a name to assign the file.

writeData( A.energy\_data, “MED\_OUT”, “A.energy\_data” )

Both IO functions have multiple other options that can be utilized as needed. See their header comments in IO\_functions.R for further details.

## Logging

There are three log functions: logStart(), printLog(), and logStop().

* logStart() is automatically called during the invocation of initialize() at the beginning of each script. It sets up the background processes that track the operations performed during the runtime of the script, and prints the first message to the log (the log\_msg that is passed to initialize()).
* Each call of printLog() literally prints the parameter string to the log as a message- this also appears in the console when running individual scripts or executing the system.
* logStop(), which takes no parameters, closes the log, writes the tracked input and output information to IO\_documentation.csv, and finishes the .log and .d files for the script (in the code/logs directory).

The .log file for the script is a record of all messages printed to the console while the script is running, including those from readData(), writeData(), all three log functions, and any other console output generated by the script, including warnings. The .d file for the script is a Makefile-style listing of all dependencies of the script (i.e. input files it relies upon).

IO\_documentation.csv is an automatically-generated file containing a list of all input files and output files for each script, along with a description of the script and its output pulled from System\_Documentation.xlsx.

When using printLog() to add messages to the log, it is best used minimally, to declare the purpose or function of specific sections of the script.

## File Metadata

A metadata entry is essentially just a classified comment, such as “Natural gas liquids (kt) in final consumption go uncounted before 1990 since shares cannot be calculated from 1990 values” for a year range and sector in the Former Soviet Union.

Every output created within CEDS is assigned metadata that is output as a separate file, unless the user specifically disables it within a writeData() call. This metadata is gathered from the metadata files, if any, of the inputs to the script, and any calls to addMetaData() that manually add metadata to be included in the output. If input files have no metadata, or have the option to read metadata turned off, the output will have a default metadata placeholder for the input file’s entry in the metadata file.

## Value Metadata

To be added later

## Diagnostics

Diagnostic outputs are created for the specific purpose of examining some relationship within the data being produced by a script- for example, one of the diagnostic outputs of A3.1.IEA\_BP\_extension.R compares the given world sums of its input data against calculated sums across all countries.

In general, there is no need to create a diagnostic output unless there is a specific question about the data that needs to be answered. If a new diagnostic is created, it should be written to the “diagnostic-output” directory.

# ISO-Sector-Fuel-Units-Year

CEDS identifies values (emissions, emission factors, driver/activity data) by country (iso code), sector, fuel, and year.

A complete list of countries, sectors, and fuels can be found in

input/mappings/Master\_Fuel\_Sector\_List.xls and input/mappings/Master\_Country\_List.csv

# Module A

Module A processes driver data. Combustion Energy data is primarily from IEA and BP data (processed in Module A2-A4), while non-combustion driver data is from various sources (Module A5).

Through the processing of driver data, sectors are either treated as combustion sectors or process sectors, as defined in Master\_Fuel\_Sector\_List.xls. Sectors with energy data are treated as combustion data by default unless specified other wise in IEA\_process\_sector.csv, which note specific IEA product/flow combinations to be used as non-combustion driver/activity data. During processing of the IEA energy data (A.2.1.IEA\_en\_bal), if a process sector (as noted by Master\_Fuel\_Sector\_List.xls) contains combustion data not addressed in IEA\_process\_sector.csv, the system gives a warning, removes the non process data (for the process sectors), and writes that data to a diagnostic file

# Module B & C Overview and Data Addition Instructions

## Overview:

Most of the R scripts for CEDS Modules B and C have been designed to be modular in and of themselves, for ease of use in integrating additional data into the Makefile system. The scripts for each section are separated into three levels:

* “base” scripts (\_.1) create blank or base-level databases for default emissions factors, activity data, and default emissions.
* “add” scripts (\_.2) reformat specific datasets and use header functions to add the results to their databases.
  + There can be any number of “add” scripts per section.
* “proc” scripts (\_.3) perform final processing on that section’s data, filling in any gaps, matching final form to CEDS Standard Style, and outputting files for the next section of data processing.

In addition, a few of the sections in modules B and C have different scripts for different emissions types. Where this is the case, there is also a parent script for that section. The Makefile runs the parent script, which chooses a script to run from amongst the emissions-specific scripts using the emissions type passed to it by the Makefile.

The only sections which currently employ this parent script system are B1.1 and C3.2.

The only section (in modules B and C) which does not employ the modular three-tier script system is C2, as there was no need to split the functionality of the single script into parts.

## Adding data:

In order to add a new piece of data to an existing section and/or emissions type, you must merely add a new script to the appropriate module folder, labeled according to the naming system explained in Section 11.

To update the parent script, if there is one, open it in the text editor of your choice and add the name of the new script to the “scripts” variable assignment in the conditional for the appropriate emissions type. (If there is a parent script, there is no need to update the Makefile, as the Makefile only calls the parent script.)

ex. scripts <- c( “C3.2.add\_SO2\_NC\_emissions\_all.R”, “<name\_of\_new\_script>” )

To update the Makefile, open it in the text editor of your choice and add a dependency line under the output listing for that section. Follow the existing format and you shouldn’t have any trouble.

Note: You must use actual tabs when adding new lines to a Makefile. It will not accept spaces as a substitute.

With regards to the actual content of the script, it should be of the same format as all other “Add” scripts, in that it should merely reformat the input data (preferably using header functions) and then call the appropriate “add” function from process\_db\_functions.R to add it to the appropriate database.

## Adding a new emissions species:

In order to add an entirely new emissions species to the system, you must have all appropriate inputs and R scripts (for the system sections that vary by emissions type) ready to add in at the same time. At the time of writing, this is limited to B1.1 and C3.2. The rest of the system is type-agnostic, and only uses the emissions type variable for naming purposes.

Once the scripts and inputs are ready, add the inputs to the dependency lists within the Makefile for the type-dependent outputs. Add a new conditional block to the parent scripts that trigger when “EM” is the abbreviation for the new emissions type, and list the specific scripts to be called in the same manner as the other emissions types. The scripts do not need to be added to the Makefile- the parent scripts serve as a wrapper for all the emissions types.

# Module E Overview

# Module F Overview

The purpose of Module F is to scale CEDS emissions data to the emissions data reported in other inventories. Module F consists of a header file (“emissions\_scaling\_functions.R”), a parent script that calls the inventory specific scripts(“F1.1.inventory\_scaling.R”), a series of scaling scripts, (e.g. “F1.1.UNFCCC\_scaling.R”), and mapping files for each inventory dataset used. The header file contains generalized functions that are called in each scaling script. These functions are used to read and write data, apply mapping files, and perform scaling calculations. Each scaling script reads in an inventory data set and updates the default data in the CEDS data sets.

Module F begins with parent script “F1.1.inventory\_scaling.R”, which calls inventory specific scripts for each emission species. When additional scaling scripts are incorporated they must added to the list of child scripts in this parent script. As of Nov 2015, scaling routines are only in place for SO2. The inventory specific scaling scripts read inventory data and matches it to corresponding CEDS data using a mapping file. A set of scaling ratios is determined from the ratio of inventory data to CEDS data at each available point. If necessary, the data is interpolated or extrapolated to CEDS years as specified. The resulting set of scaling ratios is then multiplied by the input default CEDS EFs to produce the scaled EFs. Finally, scaled emissions are calculated from the scaled EFs.

The scaled EFs and emissions data can then be written out to be scaled by the next inventory; the process is repeated in all following scaling scripts, with the default CEDS sets being replaced by the output of the previous scaling script. Inventory scaling is performed in series, and the scaled EFs and scaled emissions arrays are updated with each script. Scaling the same region more than once will overwrite the earlier scaled values. This means that the order of the scaling scripts is important, and inventories with greater accuracy should be included later to avoid being overwritten by a less accurate inventory.

Each Scaling script has a similar structure:

Section 0: universal section, the same for all scripts

Section 1: define inventory specific variables such as file names, countries and years the inventory includes, and scaling method

Section 1.5: import inventory specific data and put in standard inventory format (iso-sector-fuel-years or iso-sector/fuel-years)

Section 2: read in all other scaling data and define variables using scaling functions

Section 3: aggregate ceds and inventory data to scaling sectors/fuels using scaling functions

Section 4: calculate scaling factors and apply scaling factors to default emissions and emission factors using scaling functions

Section 5: write data

Section 1 – 1.5 are unique to each inventory used for scaling. Sections 0, 2-5 can be identical for all scaling scripts, unless the user would like to define different default options in Section 4 to create scaling factors with the function “F.scaling”.

**Required Files**

Each scaling routine requires an inventory file and a mapping file.

**Inventory files** can be excel sheets that are imported and processed to standard format within the scaling routine (ex. Canada), or imported and processes within Module E (ex. UNFCCC). Before section 2, inventory data must be in standard form with iso, sector/fuel (or both) columns and years in Xyear format.

**Mapping files** define how to relate scaling inventory and CEDS default data through scaling sectors or scaling fuels, as well iso-sector-fuel specific options for scaling routines. Mapping file templates for the 3 possible scaling methods (sector, fuel, both) are in the documentation file. Mapping files must be xlsx spreadsheet with 3 sheets named ‘map’, ’method’, and ‘year’ and located in the CEDS/input/mappings/scaling folder.

The “map” sheet is a crosswalk relating the inventory data to the CEDS data by the scaling method: either fuel, sector or both. It relates the inventory sector/fuel to the scaling sector/fuel and the scaling sector/fuel to CEDS sector/fuel. For example using the sector scaling method, the inv\_sector column maps to the scaling\_sector column, and the ceds\_sector column maps to the scaling\_sector column, but the inv\_sector column does not map to the ceds\_sector column. Entries on the same row in the inv\_sector and ceds\_sector columns have no meaning. Inventory sectors/fuel or CEDS sector/fuels should only be mapped to one scaling sector (although multiple sectors/fuels can be mapped to one scaling sector). If an inventory or ceds sector/fuel is mapped to more than one scaling sector/fuel, the system will match to the first pair in the data frame. The selected scaling sectors/fuels are applied to all countries in the inventory.

The “method” sheet defines iso-scaling\_sector- scaling\_fuel specific interpolation/extrapolation (pre and post inventory years) methods that are different from the specified options when calling scaling functions. Scaling function F.scaling calculates scaling factors and extends them across defined years. When calling the function (explained in more depth below), default interpolation/pre-extrapolation/post-extrapolation methods are defined. The method sheet allows the user to select different method (linear, constant, etc) to apply to different iso-sectors/fuels. This sheet must contain data, even if only a single line of NA values. If no methods are specified then default methods will be applied to all iso-sector/fuels. While “all” is a valid iso entry (apply the data over all countries), “all” is not a valid entry for scaling fuels/sectors; one must list sectors separately if different from default function values.

The “year” sheet defines iso-sector-fuel specific interpolation/extrapolation (pre and post inventory years) methods that are different from the specified options when calling scaling functions. Scaling function F.scaling calculates scaling factors and extends them across defined years. When calling the function (explained in more depth below), default interpolation/pre-extrapolation/post-extrapolation years are defined. This sheet allows the user to extend scaling factor to different years for individual iso-sector/fuels. While “all” is a valid iso entry (apply the data over all countries), “all” is not a valid entry for scaling fuels/sectors; one must list values for sectors/fuels separately if different from specified function values.

**Defined Variables**

Variables that need to be defined in section 1.

inventory\_data\_file : the name of the inventory file, without the extention

inv\_data\_folder : name of the path to the folder the inventory file is in, from domainmapping.csv

sector\_fuel\_mapping: the name of the inventory mapping file, without the extention

mapping\_method : mapping method, must be sector, fuel, or both

inv\_name <- name of the inventory, for labeling diagnostic/intermediate output, not for reading files

region : iso countries included in the inventory

inv\_years<-years covered by the invenotry

**Scaling Functions**

**F.readScalingData** ( inventory = inventory\_data\_file, inv\_data\_folder,

mapping = sector\_fuel\_mapping, method = mapping\_method, region, inv\_name, inv\_years)

Reads in all scaling data, defines variables for scaling and assigns them to the global environment.

**F.invAggregate**( std\_form\_inv, region , mapping\_method,

zeroed\_terms = c(NA, 'NA', 'NA ', '-'))

Aggregates inventory data to scaling sectors/fuels. There are no user defined options in this function

**F.cedsAggregate**( input\_em, region, method = mapping\_method )

Aggregates ceds data to scaling sectors/fuels. There are no user defined options in this function

**F.scaling**( ceds\_data, inv\_data, region,

ext\_start\_year = start\_year, ext\_end\_year = end\_year,

ext = TRUE, interp\_default = 'linear',

pre\_ext\_default = 'constant', post\_ext\_default = 'constant',

replacement\_method = 'none', max\_scaling\_factor = 100, replacement\_scaling\_factor = max\_scaling\_factor)

Calculates scaling factors where both inventory and ceds data are available. Interpolates and extends scaling factors forward and backward if ‘ext’ = TRUE. Also checks and replaces scaling factors if too small or too large.

ext\_start\_year: year to extend scaling factors back to. Default to global environment variable ‘start\_year’ - 1960

ext\_end\_year: year to extend scaling factors forward to. Default to global environment variable ‘end\_year’ - 2013

interp\_default: default interpolation method for scaling factors within the inventory years. Either ‘interpolation’ or ‘constant’. Defaults to linear interpolation.

pre\_ext\_default = default extrapolation method for pre inventory years. Either ‘interpolation’ or ‘constant’. Defaults to ‘constant’.

post\_ext\_default = default extrapolation method for post inventory years. Either ‘interpolation’ or ‘constant’. Defaults to ‘constant’.

replacement\_method = Either 'none' or ‘replace’. If ‘replace’ then function checks scaling factors and replaces values above and below the threshold defined by max scaling factor.

max\_scaling\_factor = if replacement method = ‘replace’. Scaling factors greater than max\_scaling\_factors and less than 1/maximum\_scaling\_factor are replaced by replacement\_scaling factor or 1/replacement\_scaling\_factor.

replacement\_scaling\_factor = value to replace too small/large scaling factors with. Defaults to max\_scaling\_factor.

**F.applyScale** (scaling\_factors)

Applys scaling factors to ceds default data. Creates scaled EF and scaled emissions

**F.write**( scaled\_ef = scaled\_ef, scaled\_em = scaled\_em, domain = "MED\_OUT")

Writes scaled emission factors are ef to intermediate output folder.