User Guide for make\_gdx.py

Author/Programmer: Austin Milt ([milt@wisc.edu](mailto:milt@wisc.edu))

Last Updated: March 23, 2016

# Intro

make\_gdx.py is a Python script made to facilitate conversion of GIS or other barrier optimization data into GDX format used by GAMS habitat optimization models (e.g. Habitat\_Opt.gms). This guide is a brief summary of the syntax/rules to follow when formatting data so that make\_gdx.py operates smoothly when run.

I encourage you to familiarize yourself with the current GAMS habitat optimization model via its user guide before continuing. I’ve attempted to be very general here to accommodate possible changes to the GAMS models, but some specific examples are used.

# Installation

As a general user you shouldn’t need to install anything (because you wont be running make\_gdx.py). If you wish to run make\_gdx.py, you will need to have Python 2.7+ installed, the most recent version of GAMS, and the GAMS Python API. If you need help installing these, contact the author.

# Data Files

There are two data files that are passed into make\_gdx.py:

* table.csv is an Excel-style CSV containing barrier data in a long table format, e.g. what you would find in the ‘barriers’ dataset in the Fishwerks Hydrography database.
* definitions.csv is an Excel-style CSV containing other optimization data, e.g. budgets, as well as a mapping from the column names in table.csv to the model symbols in the GAMS habitat optimization model file.

Below I describe how each of these should be formatted and general variations/options that can be used.

# table.csv

This Excel-style CSV contains barrier optimization data as a long table. The basic syntax is simple: each row represents a unique barrier in the hydrography database, and columns are optimization data associated with those barriers. If you are familiar with the Fishwerks hydrography database, then this file should already be apparent; it is essentially the ‘barriers’ dataset with some additional/optional fields. Here are some basic rules about the syntax in table.csv:

* Generally columns can have any name, but should be unique across the columns to avoid having some data ignored.
* Each row represents a barrier and its associated optimization data.
* Regardless of the habitat optimization model being run, you will have to have a column that specifies a *unique* ID for each barrier. Most other data could be optional depending on the model being run.

I’ll avoid being overly specific about other required columns since these will depend on the GAMS model being run. However, here are some important differences from the ‘barriers’ dataset you should be aware of based on the version of the GAMS habitat optimization model when I wrote this:

* In addition to the convention of using ‘-1’ to indicate barrier is a root (has nothing downstream), a column that does this same thing in binary (0/1) is required. A simple Excel formula achieves this, i.e. ‘=if(<downstream ID column>=-1, 1, 0)’.
* One or more binary columns designate which barriers are candidates for projects, e.g. removal and/or lampricide. Each project should have its own column.

# definitions.csv

This Excel-style CSV contains other optimization data not indexed by barriers defined in table.csv. The specific information needed in this file depends on the GAMS model being run. Generally, any data not already contained in table.csv or the GAMS model file directly *must* *be* specified in definitions.csv. Here are the basic rules of syntax for this file:

* Each row represents a unique combination of a) model run and b) symbol defined at a current index. For more on these, keep reading and check out the GAMS habitat optimization model guide.
* Three columns are required:
  1. Run: This designates the model run for which the symbol in the current row is defined. You must specify a run because make\_gdx.py makes a separate GDX for each model run. More on this in a separate section below.
  2. Symbol: This designates the symbol for which you are currently specifying model run values. In addition, it specifies which indices are being defined in the following column. More in a separate section below.
  3. Values is Column Name: This designates whether the Values column (next) is a symbol value(s) or if it identifies a column in table.csv. More in a separate section below.
  4. Values: This designates the values of the symbol being defined in the current row. Values may be either model symbol values or names of columns in table.csv that make\_gdx.py will interpret automatically. More in a separate section below.
* Any other columns are not used by make\_gdx.py. You will find descriptions for each symbol used in the current GAMS habitat optimization model in a ‘Notes’ column in the example definitions.csv included.

## Run column

The ‘Run’ column is used to differentiate between otherwise identical symbol definitions spanning multiple model runs. A separate GDX file is produced for each model run, currently with the naming convention ‘data\_run<#>.gdx’. Syntax and options for this column are:

* Values must be integers *with the exception* of optionally defining a default set of symbol values to be used for symbols not explicitly defined for a run. To specify a default value, leave the Run column blank.
* Values do not have to be sequential, e.g. your runs can be 1, 3, 999, 8, 4.
* Symbols *do not* need to be redefined for every run. By default, make\_gdx.py will take symbol values not defined for a run from the default run, i.e. defined by a blank value in the Run column. If no defaults are specified, make\_gdx.py assumes the run with the smallest value contains defaults. And if no default value exists for a required symbol, the script will fail. Thus, symbols required by the GAMS model must be defined either with a default or in the smallest-value run, but can be changed across runs by specifying the run number. See the example definitions.csv for examples.

## Symbol column

The ‘Symbol’ column specifies the symbol being defined on the current row as well as the specific indices being defined in the ‘Values’ column. The basic syntax is as follows

<symbol name>(<first index>,<second index>,…)

For example ‘cost(Barriers,removal)’ would tell make\_gdx.py that this row defines barrier removal costs over the set of barriers in the network. More on syntax:

* The <symbol name> *must match a symbol name in the GAMS model*. Since these may change across version of the GAMS model, I will not list them here. However, you can see the example definitions.csv and the GAMS habitat optimization model user guide for specific symbol names and what they mean.
* Likewise, each index must be either the name of another symbol or of a specific index for which this row is being defined. In the above example, ‘Barriers’ is the set of barriers and matches the set name in the GAMS model, while ‘removal’ is one of the Projects (another GAMS model set) that can be executed at a barrier. You will find that both of these are (necessarily) defined in other rows of definitions.csv.
* When a symbol has no indices to define, simply leave off the parentheses.
* If two symbol rows share the same run number, they must be either a) different symbols, or b) the same symbol with different indices. Otherwise, the script will fail and indicate where an issue occurred.
* Symbols *do not* need to be redefined for every run. By default, make\_gdx.py will take symbol values not defined for a run from the default run, i.e. defined by a blank value in the Run column. If no defaults are specified, make\_gdx.py assumes the run with the smallest value contains defaults. And if no default value exists for a required symbol, the script will fail. Thus, symbols required by the GAMS model must be defined either with a default or in the smallest-value run, but can be changed across runs by specifying the run number. See the example definitions.csv for examples.

## Values is Column Name column

The ‘Values is Column Name’ column is a binary/Boolean column specifying whether the values in the ‘Values’ column comes from table.csv and thus indicates a column name, or is a (series of) value(s) defined in defintions.csv. To indicate the former (a column name), specify ‘y’. To indicate the latter (a series of values) specify ‘n’. See the next section for more information.

## Values column

The ‘Values’ column specifies either a) a value for the current symbol and its indices being defined on the current row or b) the name of a column in table.csv corresponding to the values for the symbol on the current row. How do you know which is needed? Simple: if the values are indexed over Barriers, you are specifying a column name in table.csv. If the values are defining a symbol over other indices, e.g. Projects, you are specifying the value in the definitions file. Syntax rules are as follows:

* Anything is allowed and make\_gdx.py will automatically interpret what kind of data you are specifying. To accomplish this, make\_gdx.py reads the GAMS model file and determines the data type based on that. More specifically, anything that is a Set in the GAMS model will be kept as a string, while anything that is Parameter or Scalar will be converted to a floating point value (though this may happen after values are loaded from table.csv).
* Separate multiple values with a comma and, preferably, no spaces, e.g. ‘Fish1,Fish2,…’
* Numbers should be specified according to Python conventions. The only place this might deviate from expected is when you need to specify positive or negative infinity, in which case you should write either ‘inf’ or ‘-inf’.

# Going Advanced

If you want to do more customization, e.g. change the names of required columns or other default values in make\_gdx.py, you should open make\_gdx.py and start at the top. A short summary is included at the top of the script. The vast majority of customization would be done by changing either the all-caps constants defined at the top of the script or the file name defaults at the very bottom of the script. For more specific help on the individual functions run within make\_gdx.py, simply load the module and call the help function for those or read the script file, e.g.

>> from make\_gdx import \*

>> help(load\_data)