|  |
| --- |
| BuildSimLand Module |
| SAROPS Version 2.2 |
|  |
| This document describes the basic approach and algorithms of the BuildSimLand Module in SAROPS 2.2. |
| This document contains data that is delivered to the Government with Unlimited Rights in accordance with contract HSCG44-16-F-001007 and FAR 52.227-17. |
| Thomas M Kratzke, Strategic Data Systems, 19 September 2018 |

Table of Contents

[I. Introduction 0](#_Toc1984140)

[II. Sector Mobile Example 1](#_Toc1984141)

[III. Building GE Files for ADD\_POLYGON, DELETE\_POLYGON, and SUBSTITUTE\_PATH 3](#_Toc1984142)

[IV. ADD\_CONNECTOR 5](#_Toc1984143)

[V. Setting up the Sample Run 6](#_Toc1984144)

[A. Overview of runBsl.bat 6](#_Toc1984145)

[VI. Chain of “AMENDs” and Building from Scratch 7](#_Toc1984146)

[VII. Phase4Adjs Directory 8](#_Toc1984147)

[REFERENCES 9](#_Toc1984148)

List of Tables

[Table 1 Steps for Creating a New Land File 1](#_Toc1984134)

List of Figures

[Figure 1: Sector Mobile Request 2](#_Toc1984135)

[Figure 2: "Starting Context" 3](#_Toc1984136)

[Figure 3: DELETE\_POLYGON 3](#_Toc1984137)

[Figure 4: Dauphin Island "Peninsula" 5](#_Toc1984138)

[Figure 5: Using ADD\_CONNECTOR WATER to Remove 2 Isthmuses 6](#_Toc1984139)

BuildSimLand Module

SAROPS Version 2.2

# Introduction

There are four major components of the SAROPS Sim module. Two of them are highly visible; the Simulator which produces a dynamic probability distribution, and the Planner that uses this probability distribution to assign boxes to SRUs. In this document, we discuss a 3rd module called *BuildSimLand*, which we will refer to as *BSL*.

The program *BSL* produces a jar file of data that is used by the simulator, and shape files that are used by SAROPS in its display. Together, these results are called SimLand. Both *BSL* and SAROPS use the classes found in com.skagit.utils, but neither uses classes in the other’s class hierarchy; they are separate programs.

Currently, *BSL* takes about 90 minutes to run, which is down from 36 hours. If additional corrections are needed (e.g. missing islands or errant coast lines), we must produce files that specify these corrections, and rerun *BSL* to produce a new version of SimLand.

*BSL* can create multiple SimLand Versions as this document will illustrate. But rather than wade through this document, you might want to set up BSL and run it on some machine that you can spare several cycles on; BSL uses about 12 gigabytes of memory and the faster the processor the better.

## Get your Hands (a little) Dirty

To get a SimLand Version running, you need the zip file SimLandVersions0.zip and a SIM64 installer from Feb 24, 2019 or later. Unzip SimLandVersions0.zip to some directory we refer to as <SimLandVersions> and copy the script file runBsl.txt from <SIM64 InstallDir/BuildSimLandDir> to <SimLandVersions> and then rename it runBsl.bat. You might have to edit the 1st line of runBsl.bat to refer to the correct <SIM64 InstallDir>, but then it should run. The first SimLand Version that will be created will be 2019-02-17(Mobile Bay#) and you can check the creation of log files to monitor its progress. Most of the time is taken up by creating lower resolution shape files (for the land recognition of SLDMB processing) and the main jar file and full resolution shape files are built relatively quickly.

In 2.2, we are relying heavily on Google Earth (GE) to provide data for *BSL*, and hence we are using kml files. kml files are xml files with special tags. We have very simply kml documents, and we read and write a tiny subset of kml’s features. Although there is a commercial 3rd party library JAK for reading and writing kml within Java, we wrote our own code, basing it on the existing xml facilities within the current code. In Table 1, we list a very broad description of how additional corrections are added.

In Table 1, we list the main steps of running *BSL*. The rest of the document gives an example that illustrates all of these steps, including every step in Table 1, Step 2, except for 2.a and 2.c. These two would be used to delete an individual island, and all islands of a given level or higher. It is easy to do without these two.

1. Create a “Context” kml for the region of interest and load it into GE.
   1. This is optional but it is useful to get a snapshot of “where we are with SimLand that can be displayed, so we will know the effect of our modifications.
2. Using GE, click in any of the following features:
   1. DELETE\_INNERMOST (Placemark),
   2. DELETE\_POLYGON (Polygon),
   3. DELETE\_THRESHOLD (Placemark),
   4. SUBSTITUTE\_PATH (Polyline),
   5. ADD\_CONNECTOR LAND, (Polygon),
   6. ADD\_CONNECTOR WATER (Polygon),
   7. ADD\_POLYGON LAND (Polygon), and
   8. ADD\_POLYGON WATER (Polygon).
3. Store each of these changes in its own file and organize these files in a way that indicates the order they must be processed. We will discuss that more later.
4. Run *BSL*. This produces:
   1. The jar file of data
   2. The shape files form of these output files
   3. A comparison kml file that, when displayed in GE, shows the differences between the old land files and the new ones.
5. Load the jar file, the shape files, the comparison file, and any other file that may be of interest onto the ftp server.

Table 1 Steps for Creating a New Land File

# Sector Mobile Example

We work through an example concerning a few islands in Mobile Bay, AL. The request to modify the shoreline was received as Figure 1, where the red X’s mean that something should be fixed.

The northern red X indicates that Gaillard Island is missing. The southern red X indicates that the shoreline of Dauphin island is unsatisfactory.

We created a context file for that area to load into GE. To do this, you could run a SAROPS case in the region and use the shape files of that case. These would have to be converted to kml to be used by GE, and there are several sites that give instructions on doing so. Our GE view, together with this SAROPS context, is shown in Figure 2. Gaillard Island is clearly missing and the southern peninsula of Dauphin Island is missing.

We will change SimLand with a sequence of operations. In our 1st two steps, we will Add a Polygon (Gaillard Island), and delete all of the islands near Dauphin Island. Then for illustration purposes, we will create a peninsula representing Dauphin Island by modifying the mainland’s shoreline. There is a bridge connecting the mainland to Dauphin island and, in our first modification, this bridge will be an isthmus that connects the mainland to Dauphin Island. Hence, Dauphin island will be part of the mainland.

So far, we have considered ADD\_POLYGON LAND, DELETE\_POLYGON, and SUBSTITUTE\_PATH to create Gaillard Island, clear out all the islands near Dauphin Island, and create the peninsula respectively. Hence, we have discussed 2.b, 2.d, and 2.e. We wll not discuss 2.a or 2.c. We will use 2.e to remove the isthmuses that we created with our SUBSTITUTE\_PATH, thus making Dauphin Island an island again.



Figure 1: Sector Mobile Request

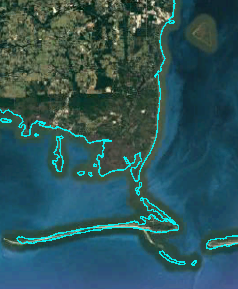


Figure 2: "Starting Context"

# Building GE Files for ADD\_POLYGON, DELETE\_POLYGON, and SUBSTITUTE\_PATH

ADD\_POLYGON LAND and DELETE\_POLYGON are not opposites of each other. The former is used to add a polygon (e.g. Gaillard Island), whereas the latter is used to remove multiple islands. The order of the operations is important and it is always safe to delete existing polygons 1st. In Figure 3, we delete all the old (blue “context”) polygons that are completely enclosed in the white polygon.

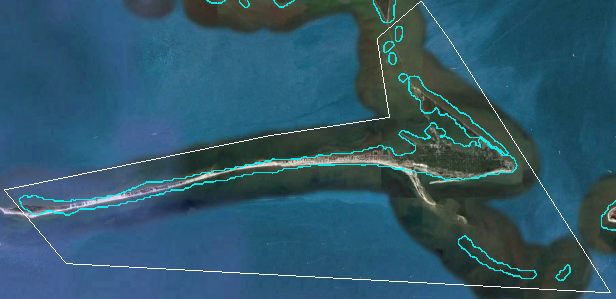


Figure 3: DELETE\_POLYGON

The white polygon was built in GE by creating a “Polygon” and naming it DELETE\_POLYGON. We will save the polygon into a file later, rename that file, and place it in an appropriate folder later.

For both ADD\_POLYGON LAND and DELETE\_POLYGON, we use GE to “click in” a polygon. For simplicity’s sake, we store each polygon in its own file, and put them somewhere that *BSL* will read them. *BSL* will know what to do with the polygon simply by reading the polygon’s name. Normally, it is important to trace out all lines and polygons so that “land is to the left.” However, GE does not always retain the sense of rotation and hence we must specify LAND or WATER for both ADD\_POLYGON and ADD\_CONNECTOR. It is not necessary for DELETE\_POLYGON because we simply gather all the polygons that are completely within the input polygon.

Next, we will extend the shoreline by using SUBSTITUTE\_PATH. This command does not take a polygon, but rather a line. GE *does* retain the order of the points as they are “clicked in,” so it is very important to click them in so that land is to the left. In Figure 4, we show that we clicked in a large peninsula representing Dauphin island together with the bridge (isthmus) that connects it to the mainland.

It is worth noting that this is a single polyline. Keeping land to the left, we started at the western of the two northernmost points, traced the west side of the bridge before going along the north side of the arm that extends to the west. We continued along the south side of this arm, picked up the southern peninsula, circled around the “interior bay,” and finished by following the east side of the island and the east side of the bridge.

We did not do this in one sitting and it is here that GE’s editing is very useful. It allows one to restart the editing of a polygon, as well as insert and delete points, all the while providing the imagery for guidance.

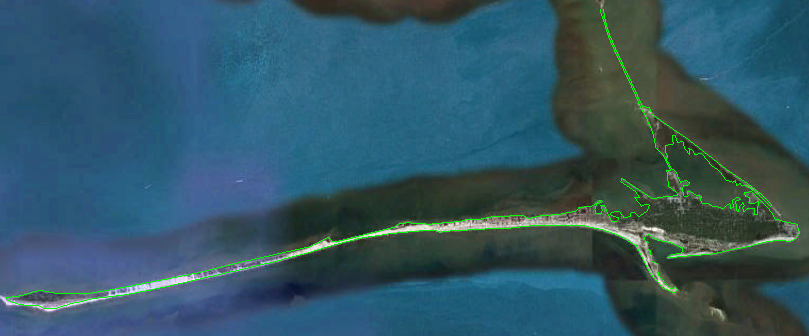


Figure 4: Dauphin Island "Peninsula"

# ADD\_CONNECTOR

If we stopped here, we would have left the isthmuses in, which is visually acceptable. However, they are really bridges and particles should be able to drift under them so we must remove the isthmuses representing the bridges. We will do this with an ADD\_CONNECTOR WATER.

Notice that unlike the previous operations, removing the isthmuses affects multiple polygons and even creates two new islands; these islands go from being part of the mainland to “new” islands. Although DELETE\_POLYGON affected several polygons, there was no interaction between these polygons; they were simply all deleted. ADD\_CONNECTOR is more powerful and in fact, all editing could be done with an appropriate sequence of ADD\_CONNECTORs.

The idea behind ADD\_CONNECTOR is to simply declare a polygon as land or water. If one wished to add a dam, he would use ADD\_CONNECTOR LAND and the polygon that he enters would be interpreted as land. In our case, we wish to remove two isthmuses, so we use ADD\_CONNECTOR WATER. Our clicked in ADD\_CONNECTOR is shown in Figure 5.



Figure 5: Using ADD\_CONNECTOR WATER to Remove 2 Isthmuses

Our ADD\_CONNECTOR also illustrates the importance of “order.” We must make sure that the SUBSTITUTE\_PATH is processed before the ADD\_CONNECTOR is.

# Setting up the Sample Run

In this section, we show how to run the program and give several examples of doing so. *BSL* is just a DOS program with several arguments. It is easiest to run this from a .bat file and we provide an example. Here, we discuss that sample.

To follow this discussion, unzip SimLandVersions0.zip into a directory of your choice, which we shall refer to as <SimLandVersions>. This directory will contain a directory Stable and several SimLand Version directories. Copy the file runBsl.bat file from <SIM64 InstallDir>/BuildSimLandDir into <SimLandVersions>.

## Overview of runBsl.bat

To explain runBsl.bat file, we must review a few “tricks” about .bat files that we use. A caret symbol (^) indicates that the command and its list of arguments continues on the next line. Otherwise, the command and its arguments end with the line. An argument that has a space in it must be enclosed with double-quotes. Otherwise, the space would indicate the end of that argument.

Our runBsl.bat is given as Equation 1.

C:\SAROPS\SIM64\BuildSimLandDir\BuildSimLand.exe ^

AMEND "2019-02-17(Mobile Bay#)" "2018-08-16(Jarvis Island)" SimLand2.2\_f.b ^

AMEND "2019-02-20(LALB#)" "2018-08-17(Mobile Bay#)" SimLand2.2\_f.b ^

AMEND "2019-02-24(Bonneville Dam#)" "2019-02-20(LALB#)" SimLand2.2\_f.b ^

PHASE1 "2019-02-24(Bonneville Dam)" "2019-02-20(LALB#)" Phase1Gshhg-bin-2.3.7/gshhs\_f.b SimLand2.2\_f.b ^

STOP

pause

Equation : Sample Script for Running BSL

Because of the carets, all but the last line is the BuildSimLand command; the pause simply keeps the DOS window up until we are ready to close it.

The 1st line is simply the full path of the executable and it is possible, though not likely, that you will have to change this. Each of the next 4 lines is the set of parameters for building a SimLand Version. Each SimLand Version has its own directory and data files within it, and they are all relative to <SimLandVersions>. This .bat file will create SimLand Versions for the sub-directories "2019-02-17(Mobile Bay#)", "2019-02-20(LALB#)", "2019-02-24(Bonneville Dam#)", and "2019-02-24(Bonneville Dam)", and these SimLand Versions will be built in that order. These directories and their data already exist as examples as a result of unzipping SimLandVersions0.zip into <SimLandVersions>.

The STOP is the final argument for the *BSL* program and simply indicates that there are no more cases. It’s optional.

You can then double-click the runBsl.bat to build the SimLand Versions. This will take a long time and it’s best to try just one at a time. When you’re confident that you have it set up, run it on a fast machine that you can ignore for a while (e.g., overnight). When you double-click runBsl.bat, the correct behavior is that practically nothing happens on the command prompt window. You can monitor progress by checking log files in the SimLand Version being worked on.

# Chain of “AMENDs” and Building from Scratch

The directory 2018-08-16(Jarvis Island) will not be affected; it is the starting point for building the other SimLand Versions. As such, this directory’s structure is slightly different. Each of the SimLand Version sub-directories (including Jarvis Island) contains a sub-directory called data, and each of these contains a directory called Phase4Adjs. But 2018-08-16a(Jarvis Island) also contains a sub-directory LandJarDir. The data inside of Jarvis Island’s LandJarDir is necessary for building the SimLand Version 2019-02-17(Mobile Bay#). The script will build Mobile Bay#, resulting in, among other results, Mobile Bay#’s own LandJarDir. Mobile Bay#’s LandJarDir will be used to build 2019-02-20(LALB#). LALB#’s LandJarDir will be used to build 2019-02-24(Bonneville Dam#).

Until now, we have been “amending” previous SimLand Versions. The final SimLand Version 2019-02-24(Bonneville Dam) will be built “from scratch.” Bonneville Dam will use only the data in Stable and its own Phase4Adjs directory. The reader might expect that there will be no need for Bonneville Dam to reference LALB#, since it doesn’t use LALB# and in fact, that is correct. However, we still must provide some argument there in case we want to compare the results of Bonneville Dam to some other SimLand Version (in this case, LALB#). Hence, the rest of Bonneville Dam’s arguments are SimLand Version to compare it against, starting Gshhs File in Stable, and LandJarDir file to compare it against.

# Phase4Adjs Directory

*BSL* builds each SimLand Version in several phases. The 1st 3 phases of *BSL* use the data from the Stable directory. When we “amend,” we skip these 3 phases and go directly to “Phase 4.” The data of Phase 4 is the set of kml files inside of data/Phase4Adjs. These files contain the “minor adjustments” listed in Table 1.2.

Phase4Adjs contains the cumulative list of minor adjustments. But when we amend a previous version, we process only those minor adjustments that are not part of the previous version’s Phase4Adjs. Hence, we can (and do) keep the entire list of adjustments in each SimLand Version; the duplicates get ignored when amending anyway. This makes it easier to track what adjustments are in which Version and when we start from scratch, we can use the same set minor adjustments.

Each minor adjustment in Phase4Adjs must be in some sub-directory of Phase4Adjs, and this sub-directory’s name must start with 2 digits, a decimal, and 2 more digits. These 5 characters at the beginning of the directory’s name are interpreted as a number, and we call the sub-directory a “numbered sub-directory.” The numbers of the numbered sub-directories indicate the order in which the processing takes place. For example, all the minor adjustments in 00.00-OldAdjs will be processed before those in 01.00-Carib. Each kml file can have multiple minor adjustments, and there can be several kml files within a numbered sub-directory. All the minor adjustments in all the kml files inside of a numbered sub-directory are processed in an unspecified order.

For example, the corrections in WA-OR fell into 3 categories: Delete Islands, Modify Mainland, and Add Island. We processed them in that order, and we did not care about the order of the individual islands being deleted, nor the order of the mainland adjustments, since these adjustments were well separated. But clearing out the dead islands first minimized the chance of confusion that could occur if we were to move a shoreline so that it crossed an island that was scheduled for deletion. In general, it is safest to delete polygons first, then modify the shoreline, then add the new islands.

We include Bonneville Dam# and Bonneville Dam to illustrate amending (Bonneville Dam#) and starting from scratch (Bonneville Dam). Because our convention is to keep the cumulative list of minor corrections in Phase4Ads, we simply made a copy of Phase4Adjs when we readied Bonneville Dam. In addition, Bonneville Dam illustrate ADD\_CONNECTOR LAND.

# REFERENCES