Urban Boundary Model
User Guide

Prepared by: James Harvie
Spatial Edge Consulting
March 1, 2018

SPATIALEDGE CONSULTING

Table of Contents

Introduction

The purpose of this document is to describe the processes and procedures utilized by the Urban Boundary Model in enough detail that future maintenance of the model would be possible in the event the original developer was not available.

Model Summary

The Urban Boundary Model was designed to generate geographical areas that define concentrations of population in Canada. The typical approach would be to rely heavily on population data provided by Statistics Canada to define these areas. Unfortunately, available data does not provide enough spatial detail outside of major urban centers making it impossible to accurately assess smaller communities.

To mitigate this problem, the model places more emphasis on road density to define the extents of the boundaries and uses population amounts to classify those boundaries with respect to their importance. By doing this, the model leverages the high, positive correlation between population and road density i.e. the greater number of roads the higher the population. Once the boundaries have been generated population estimates are applied allowing them to be classified as being either a Primary, Secondary, or a Tertiary urban boundary.

A fourth classification of urban boundary is also defined, called Supplementary boundaries Supplementary boundaries are urban boundaries which fall just short of the threshold values needed to make them Tertiary boundaries but contain a significant number of Genworth properties making these areas important to Genworth. It should be noted that Supplementary boundaries are generated and saved to a separate data file. They are not included in the result files until approval at a later date.

Importance to Genworth

The Urban boundary model allows Genworth to be more granular on how we think about risk related to location. By utilizing road density and population we can classify properties into 4 categories, Primary, Secondary, Tertiary and Rural. We can then use these categorizations in our risk tools and models to assess the potential for loss and the severity of loss. The goal of the model is to refine the way we define Urban and Rural properties on the periphery. Our current methodology categorizes Urban and Rural based on the FSA. When the Urban boundary is layered in, the impact of using this extra layer becomes apparent. Figure 1.0 demonstrates how our historical portfolio aligns by both the current Urban definition and the new geocoded Urban Boundary model. Most Urban files are categorized as Primary, but there are over 6% of loans that do not align to a true primary urban market and the majority of that are actually Rural (4.3%). Even

more apparent is the Rural definition, where only 41.6% of properties remain as Rural, and 16.3% are considered Primary under the new model.

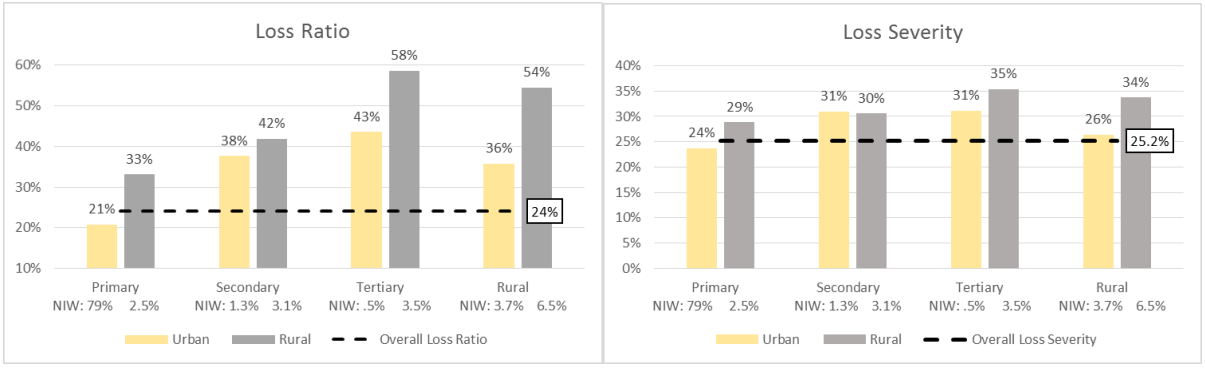
Figure 1.0

FSA code	Urban				Total FSA
GEOcode	P	S	T	Rural	
% Total NIW	79.0%	1.3%	0.4%	3.7%	84.3%
% Urban NIW	93.7%	1.5%	0.4%	4.3%	100.0%

FSA code	Rural				Total FSA
GEOcode	P	S	T	Rural	
% Total NIW	2.5%	3.1%	3.5%	6.5%	15.7%
% Rural NIW	16.3%	19.8%	22.4%	41.6%	100.0%

Looking at historical performance by both the FSA and Geocoded definition of Urban and Rural the distinction between categories is evident, as illustrated in Figure 2.0. For example, Urban Primary loans perform better than the portfolio average of 24% with a 21% loss ratio, and Secondary, Tertiary and Rural loans that we think about as Urban today perform significantly worse. Along the same vein, Rural Primary loans do not perform as well as their Urban counterparts with a 37% loss ratio historically vs. 21% for Urban Primary. The power of the layering these 2 components together gives the ability to be more precise in how we weight, and decision based on property location.

Figure 2.0



Possibly insert text/images showing the difference between the other methodologies used by GNW to define urban areas, namely FSA, Market Code, old DMTI boundaries

High Level Methodology

The Urban Boundary Model divides all data processing into three categories defined by geographical size, national, provincial and a customize area called a model area. The following describes the types of processes performed at each of these categories/levels. See Table 1 for a graphic representation.

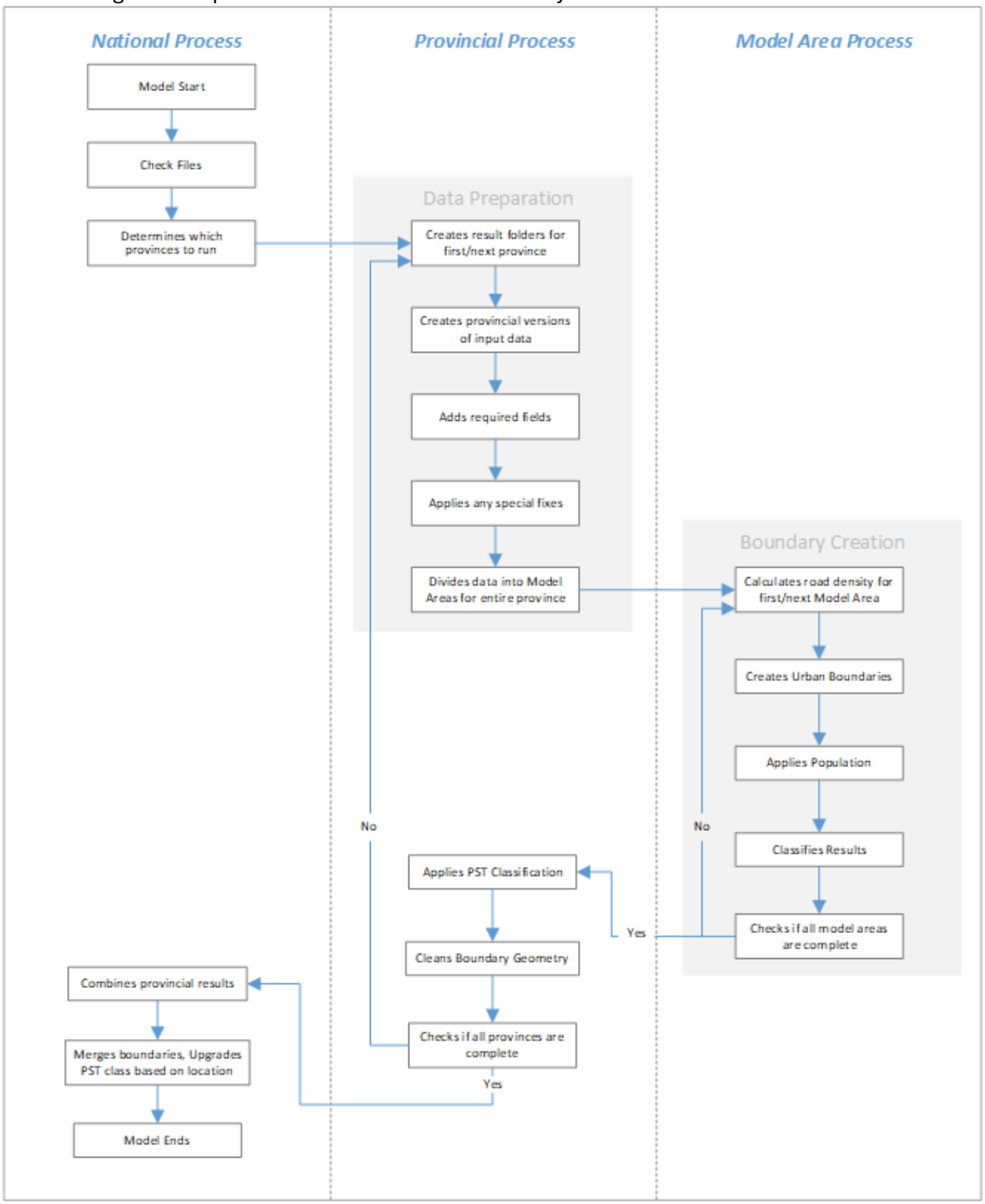
- **National** - Checks are performed on input data to detect any obvious issues that may result in the model failing such as required fields missing from data. When all provinces have completed processing then all results are combined into a single national map layer. Results are then further processed to remove any artifacts such as boundary overlaps that occurred between adjacent.

- **Provincial** – For each province a series of procedures are performed to prepare the data for boundary generation. This includes the creation of file folders where working files and final results are placed, the division of input data by province and then by the smaller model areas, and finally the addition of any required fields to some input data and applying data fixes where necessary. When urban boundaries are complete for all areas in the province, they are combined into a single provincial map layer

- **Model Area** – The model area is the smallest geographical area used by the model and is where the urban boundaries are generated. This involves calculating road density, generating boundaries, applying additional information such as population, and classifying results.

The following diagram illustrates the general process flow of the model as described by the three categories above.

Table 1 - High level representation of the Urban Boundary Model



Model Outputs

All model outputs are stored in the 04_Results folder within the model's directory structure. The results for each run are stored to a sub folder with a name starting with the prefix "UrbanAreas_" followed by the date the model was initiated in the form yyymmdd. If the model is initiated more than once on a given date, then results are overwritten.

Among the output files, there are five that are always produced regardless of the number of provinces selected to be processed. These are described as follows.

ModelSettings

A MapInfo table containing a record of the user defined settings used to create the urban boundaries.

Detail1	Detail2	Detail3
Processed	10	AB, BC, MB, NB, NL, NS, ON, PE, QC, SK,
RoadCnt	25	Threshold of roads needed to build a boundary
BuffCnt1	20	Buf Count where density >= RoadCnt
RoadCnt2	20	Lowest Buf Count threshold
CertCnt	100	Number of GNW points required to keep a supplimenta

BoundaryMergeLog

A MapInfo table containing a list of boundaries that were merged with another because their boundaries overlapped. The lower classed boundary is always merged with the higher classed boundary. The table identifies the boundary that was merged and the boundary into which was merged (target). The following is an example of this output file.

MergeID	MergeName	MergePro	TargetID	TargetName	TargetProv	Description
117	Entwistle	AB	243	Entwistle	AB	Tertiary merged with Secondary
287	Amelia	AB	265	Amelia	AB	Tertiary merged with Secondary
779	Morna	NB	740	Grand Bay	NB	Tertiary merged with Secondary
1,363	Kincardine	ON	1,039	Kincardine	ON	Tertiary merged with Secondary
1,730	Notre-Dame-de-Beaulac	QC	1,980	Notre-Dame-de-Beaulac	QC	Tertiary merged with Secondary

BoundaryUpgradeLog

A MapInfo table containing a list of boundaries which had their urban classification upgraded due to their proximity to a higher classed boundary. The table identifies which boundaries were upgraded and what new classification it was given. The distance used to determine if a boundary is upgraded is a user defined variable entered in Configuration file. It was initially set at 2km. The following is and example of the BoundaryUpgradeLog

CommName	ModAreaName	Prov	UpgradeInfo
Rural West Big Lake	Edmonton	AB	3 to 1 - Intersection within 2 km
Huntingdon	Abbotsford	BC	3 to 1 - Intersection within 2 km
Cumberland	Comox	BC	2 to 1 - Intersection within 2 km
Royston	Comox	BC	3 to 1 - Intersection within 2 km
Boundary Bay	Delta	BC	3 to 1 - Intersection within 2 km

LogFile

A MapInfo table containing a record of all the milestones reach while the model was running. This file is a copy of the one generated by the model which is located in the \UB_Model\1_Program\ folder. It is copied to the Results folder once all processing is complete.

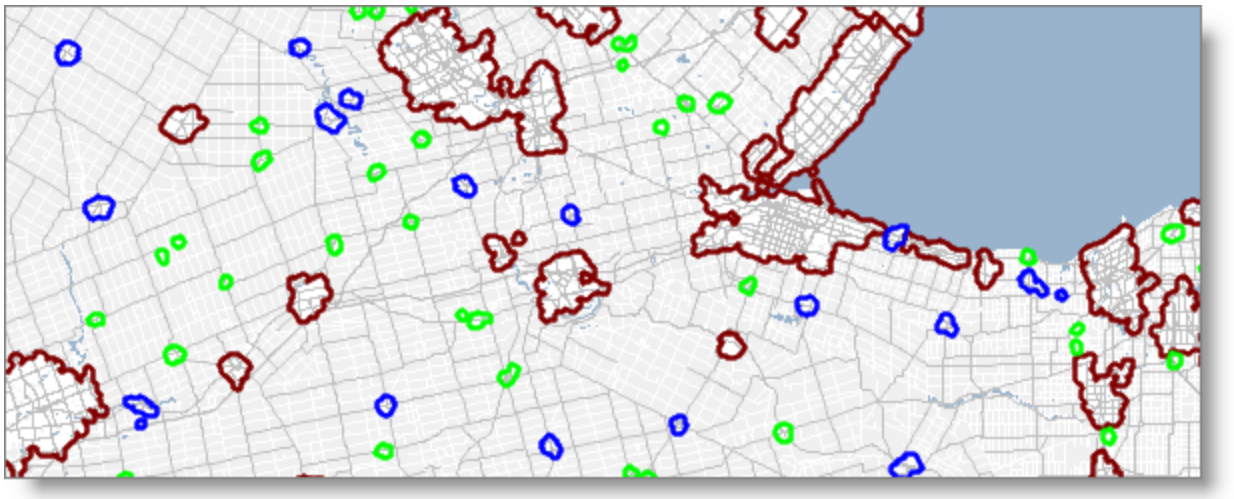
StartDate	StartTime	ProcName	Prov	Message1	Message2	Message3	Message4	Message	Message6
8/10/2019	12:20:09.0	Model Start	-						
8/10/2019	12:20:09.0	Model Start	-	Email Notification Sent	Error flag in Config File is reset to zero	Date/Time format initialized			
8/10/2019	12:20:09.0	Result Folder	-	Already Exists	Deleting and recreating folder				
8/10/2019	12:20:09.0	Model Start	-	Result folder created with toc					
8/10/2019	12:20:09.0	Model Start	-	Deleting and re-creating XTer					
8/10/2019	12:20:11.0	Model Start	-	Picking Provinces	Provinces to be run: AB, BC, MB, NB, N				
8/10/2019	12:20:11.0	Data Check 1		Road Network: Found					
8/10/2019	12:20:11.0	Data Check 1		Muni Boundaries: Found					

SuppBoundsFINAL

A MapInfo boundary file containing supplementary boundaries. These boundaries represent communities which fall just short of the thresholds used by the model, however, there are a significant number of insured properties in the area to warrant investigation. Currently these boundaries are not included in the final results, however, they are generated for future consideration. This boundary file will only be generated if the provinces included in the model's run actually contain this type of boundary.

UrbanAreas_

A MapInfo region file containing boundaries classified as either Primary, Secondary, or Tertiary. If all provinces are selected to be processed, then the Results folder will contain a separate region file for each province along with a national version. The provincial files will be named with "UrbanAreas_" as the prefix, and the province two letter abbreviation as the suffix, for example, UrbanAreas_AB.TAB. The national version of the boundaries will have the same prefix but will hav "CAN as the suffix, for example, UrbanAreas_CAN. If a smaller number of provinces are selected, then a national version is not created.



Results of the Urban Boundary Model for Ontario showing Primary (brown), Secondary (blue), and Tertiary (green) boundaries.

If all provinces are selected to be processed, then several additional files are generated. The national coverage file UrbanAreas_CAN is segregated by the three urban classifications.

- UrbanAreas_CAN_Prim.TAB
- UrbanAreas_CAN_Sec.TAB
- UrbanAreas_CAN_Ter.TAB

In addition, copies of these files are saved with projection system using the 1983 North American Datum using the following naming convention.

- UrbanAreas_CAN_NAD83.TAB
- UrbanAreas_CAN_Prim_NAD83.TAB
- UrbanAreas_CAN_Sec_NAD83.TAB
- UrbanAreas_CAN_Ter_NAD83.TAB

Processing Time

The following table contains estimates on processing time required for each province.

Prov	Start Time	End Time		Total Time
AB	1:58:14 PM	4:19:29 PM		2:21:15
BC	4:19:29 PM	6:20:25 PM		2:00:56
MB	6:20:25 PM	6:59:56 PM		0:39:31
NB	6:59:56 PM	7:37:59 PM		0:38:03
NL	7:37:59 PM	7:54:11 PM		0:16:12
NS	7:54:11 PM	8:24:28 PM		0:30:17
ON	8:24:28 PM	1:11:45 AM		4:47:16
PE	1:11:45 AM	1:16:43 AM		0:04:58
QC	1:16:43 AM	5:21:35 AM		4:04:52
SK	5:21:35 AM	6:27:06 AM		1:05:31
				16:28:51

Running the Model

The Urban Boundary Model can be run in two different manners. The first is within an instance of MapInfo Professional that is running in the background, where the application's interface is not visible on the screen. The second is within an open instance of MapInfo Professional.

It is not recommended that separate instance of the MapInfo Professional run the model at the same time as it will likely fail due to read/write conflicts with the same file.

Running in Background

Running the model in the background will initiate the model in an instance of MapInfo Professional whereby the application's interface is not visible on screen.

1. Navigate to the...\UB_Model\ folder

2. Run the UBM_Start.BAT file

A Command window will appear from which the MapInfo Professional is opened and the UBM program is launched. This window is independent from the model and may be closed manually at any time. A second Command window will appear for a brief moment but will close itself when required. This window is related to the sending of the email message notifying stake holders that the model has been initiated. You can track the model's progression by opening the LogFile.CSV file located in the ...\UB_Model\1_Program\ folder.

Running in an open MapInfo Professional application

Running the model in an open instance of MapInfo Professional provides the advantage of having access to the model's progress in real time. When it is run in this manner, the UBM program opens the Message window in MapInfo and continually updates it as the model runs. There are two ways to start the model in this manner, each are described below.

1. In a Windows Browser window, navigate to the \UB_Model\1_Program\ folder. Double click on the UBM_Project.MBX file to run the program. This will first open an instance of MapInfo Professional and then launch the UBM program

2. In an open instance of MapInfo Professional select Tools>Run MapBasic Program from the main menu. Navigate to the \UB_Model\1_Program\ folder, select and open the UBM_Project.MBX file.

Shutting down the model

If there is a reason to shut the model down before it has fully completed, the recommended method is to simply run the ForcedShutdown_RunMe.BAT file located in the ...\UB_Model \ folder.

This batch file performs two actions. The first is to launch a separate instance of MapInfo Professional, and the second, runs the ForcedShutdown.MBX program. This program opens a table called ForcedShutdown_Table.TAB and updates the ErrFlag field with a value of 1.

The result of this action is the model will cease to run, however, this may not happen immediately. A shutdown will only occur when the model reaches a place in the code where the ErrDetect procedure is called. This could take a few minutes.

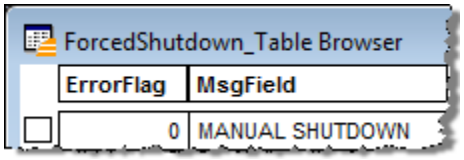
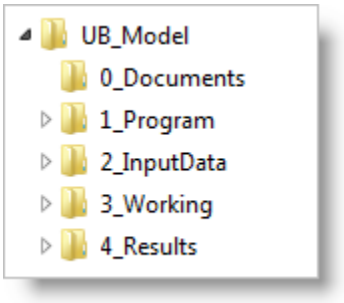


Image shows the single record contained in the ForcedShutdown_Table. When the ErrorFlag field is set to 1 the model will shut down all processing.

Folder Structure

The model uses five folders to organize input/output files and program components, each are described briefly below and in more detail on the following pages. The names of these five main folders contain a number to help enforce the order in which they are displayed when sorted alphabetically.

0_Documents - contains the model's documentation



1_Program – contains all program elements including executables, log files, and any custom input tables that are used to support the model and do not change.

2_InputData – contains a copy of the original input data used to by the model. Any new data provided by vendors is copied here. All data is left unmodified.

3_Working – contains the modified versions of the input data as required by the model. This includes the provincial versions of each data set.

4_Results – contains the results of the current and all historical runs of the model. Results are stored in a folder that is named with the date in which the model was initiated. If the model is run multiple times on the same day then only the results of the last run are retained, all others are overwritten.

Program Folder

The Program folder contains all program elements including executables, configuration and log files required by the model. Tables that are used by the model but are not considered input data are contained in the ...\1_Program\SupportFiles folder.

Folder Name	File Name	Extension	Description
...\1_Program	UBM_Project	.MBP, MBX	MapBasic project file and executable (mbx).
	UBM_MainProgram_3	.MB, .MBO	MapBasic program component. The .MB file is a text file containing the model's code. The .MBO file is the compiled version.
	UBM_xEmailNotification	.MB, .MBO	
	UBM_xFilePath_v2	.MB, .MBO	
	UBM_xTimeCheck	.MB, .MBO	
	UBM_xUpdateLog	.MB, .MBO	
	2a_CheckFiles_v2	.MB, .MBO	
	2b_xGetFileDate_v1	.MB, .MBO	
	2c_xCopyFile_v1	.MB, .MBO	
	3a_DataPrep_v6	.MB, .MBO	
	3b_xCreateAreaFolders_v2	.MB, .MBO	
	3c_RoadDensity_v8	.MB, .MBO	
	3d_TrimAndFill_v1	.MB, .MBO	
	4_PostProcessing_v2	.MB, .MBO	
	ConfigurationFile	.TAB, .DAT	MapInfo table containing program settings
	LogFile	.TAB, .DAT, .CSV	Contains a description of the model's progress at each step during its run. Two versions are created, a MapInfo .TAB file and a CSV file.
	FileFunctionsLib	.DLL	Contains several functions including the ability to create and delete file folders
	sogGenworthApp	.DLL	
	gdal_csharp	.DLL	All related .DLL's used by the sogGenworthApp.DLL. Components needed for aggregating point data and plotting points along line segments
	gdal_wrap	.DLL	
	gdal200	.DLL	
	geos_c	.DLL	
	msvcp120	.DLL	
	msvcr120	.DLL	
	ogr_csharp	.DLL	
	ogr_wrap	.DLL	
	osr_csharp	.DLL	
	osr_wrap	.DLL	
	proj	.DLL	

Folder Name	File Name	Extension	Description
…\1_Program\SupportFiles	ModelAreas_AB	.TAB, .ID, .IND, .DAT, .MAP	MapInfo region file containing the boundaries for all model areas within each province. MapInfo region file containing the boundaries for all model areas within each province.
	ModelAreas_BC	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_MB	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_NB	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_NL	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_NS	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_ON	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_PE	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_QC	.TAB, .ID, .IND, .DAT, .MAP	
	ModelAreas_SK	.TAB, .ID, .IND, .DAT, .MAP	
	TrimAndFill	.TAB, .ID, .IND, .DAT, .MAP	MapInfo region file containing polygons used for trimming urban boundaries as well as filling in unwanted holes.
	UBC_TrimArea	.TAB, .ID, .IND, .DAT, .MAP	MapInfo region file containing a single boundary around the University of British Columbia. Boundary is used to ensure area is processed with the closest populated area.
	PPN _AB	.TAB, .ID, .IND, .DAT, .MAP	MapInfo point file containing the location of populated places within each province. MapInfo point file containing the location of populated places within each province.
	PPN _BC	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _MB	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _NB	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _NL	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _NS	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _ON	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _PE	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _QC	.TAB, .ID, .IND, .DAT, .MAP	
	PPN _SK	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _AB	.TAB, .ID, .IND, .DAT, .MAP	MapInfo region file containing the voronoi diagram (natural neighbourhood region) for each point in the PPN file for each province. MapInfo region file containing the voronoi diagram (natural neighbourhood region) for each point in the PPN file for each province.
	Voronoi _BC	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _MB	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _NB	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _NL	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _NS	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _ON	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _PE	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _QC	.TAB, .ID, .IND, .DAT, .MAP	
	Voronoi _SK	.TAB, .ID, .IND, .DAT, .MAP	

Input Data Folder

The InputData folder is the main storage area for all input data used to by the model. Any updates from data vendors are copied to the appropriate sub folder. This data is left unmodified by the model. The following table provides a brief description of the data contained in each sub folder.

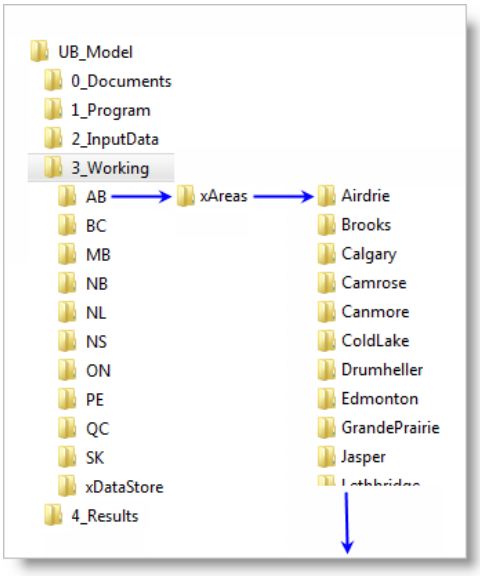
File Name	Vendor	Format	Description
RoadsLine	DMTI	MapInfo .TAB	Street network file
MunicipalityRegion	DMTI	MapInfo .TAB	Municipal boundaries
EnhancedPostalPoint	DMTI	MapInfo .TAB	Point file of 6-digit postal codes
PopTotal	Manifold Data Mining	Text .TXT	Demographic data by 6-digit postal code
GNWPropertyPoint_*date*	Genworth	MapInfo .TAB	Point table of Genworth properties, used for defining supplementary boundaries

Please note, when updating any of these files be mindful of file name changes. All changes must be reflected in the model's configuration file. The model will only utilize data files which are specified in the configuration file.

Working Folder

The purpose of the Working folder is to not only store copies of the original input data but also to organize the large number of interim files generated during each step of the model. Within the working folder there are eleven sub folders, one for storing the copies of the five input data files, called xDataStore, and then one for each of the ten provinces for which the model builds urban boundaries. The purpose of the "x" in the folder name ensures that it is listed last when sorted alphabetically and therefore not being mixed in with the other provincial folders.

Each provincial folder contains several MapInfo files and a subfolder called xAreas. The MapInfo files are mostly provincial versions of the input data along with others created during the model's run. It also contains the final build of the urban areas and the supplementary boundaries for the province. A listing of these files can be found in the table on the following page.



The xAreas folder contains a subfolder for each model area in the province. These folders contain all files needed or created during the generation of both urban and supplementary boundaries. A listing of these files can be found in the table on the following page.

Please note that before processing is initiated for each province, the province folder is deleted ensuring all files found in the folder are from the current run of the model.

A listing of files generated and contained within each of the provincial working folder

File Name	Description
GNWpoints	Provincial version of GNW property points
ModelAreaLUT	Lookup table of all model areas in the province
MultiObjPolys	A listing of municipalities that comprised two or more objects
MultiObjPolys2	Reassignment of municipality regions to the region in which they are located
PopTotal	Provincial version of population data
PostalPointsAll	Provincial version of 6-digit postal code points
Roads_Carto1to5	Provincial version of roads with CARTO class between 1 to 5
SuppBoundsALL_*prov*	All supplementary boundaries in the province
SuppBoundsFINAL_*prov*	Filtered, and final provincial version of the supplementary areas
UrbanAreas_*prov*	Final provincial version of urban areas
xDupMunis_LUT1	Lookup table of municipalities with the same name
xDupMunis_LUT2	Lookup table of duplicated municipalities with the same name and where each municipality are in different model areas

A listing of generated files found in each of the model area working folders

File Name	Description
GNWpoints	GNW property points
PostalPoints	6-digit postal code points
PlaceNamePoints	Customized point file of community locations
Voronoi	Customized file of Voronoi regions generated from PlaceNamePoints
Roads_Carto1to5	Roads with CARTO class between 1 to 5
Roads_Carto2to5	Roads with CARTO class between 2 to 5
Roads_Carto1and2	Roads with CARTO class between 1 and 2
ShortRoads	Roads with CARTO class 2 to 5 that are <=1km
ShortRoadPts	A point file representing the centroid of each road segment in ShortRoads
ShortRoads_Agg100	Short road points aggregated at a distance of 100m
LongRoads	Roads with CARTO class 2 to 5 that are >1km
LongRoadPts	A point file where a point is plotted every 500m along LongRoads segement
LongRoads_Agg500	Long road points aggregated at a distance of 500m
RoadDensityPoints	Combination of short and long roads points from which buffers are created
RoadBuffers	Circular, 1 sqkm buffers generated around road segments
BuffsXXplus	Copy of road buffers table, used for calculating road density
InBoundaryPoints	Postal code points located within an urban area
OutBoundaryPoints	Postal code points located outside of an urban area
Vor_LUT	Lookup table of Voronoi regions used for assigning postal points to a model area
Vor_LUT2	Another lookup table of Voronoi regions used for assigning postal points to a model area

UrbanAreas	The final version of urban boundaries for the model area

Results Folder

The Results folder contains all model outputs from the both the current and previous runs of the model. To better organize the results, all output files are stored in folder with the date in which the model was run using the format yyyymmdd. The following table lists and describes the result files generated by the model.

File Name	Description
LogFile	The log file from the model run that generated the enclosed boundary files
ModelSettings	A MapInfo table containing the user definable variables used for the model run.
BoundaryMergeLog	A MapInfo table containing a list of boundaries that were merged with another because of their boundaries overlapped
BoundaryUpgradeLog	A MapInfo table containing a list of boundaries which had their urban classification upgraded due to their proximity of a higher-class boundary
UrbanAreas _AB	Urban boundaries generated for Alberta
UrbanAreas _BC	Urban boundaries generated for British Columbia
UrbanAreas _MB	Urban boundaries generated for Manitoba
UrbanAreas _NB	Urban boundaries generated for New Brunswick
UrbanAreas _NL	Urban boundaries generated for Newfoundland
UrbanAreas _NS	Urban boundaries generated for Nova Scotia
UrbanAreas _ON	Urban boundaries generated for Ontario
UrbanAreas _PE	Urban boundaries generated for Prince Edward Island
UrbanAreas _QC	Urban boundaries generated for Quebec
UrbanAreas _SK	Urban boundaries generated for Saskatchewan
UrbanAreas_CAN	*National coverage of urban boundaries
SuppBoundsFINAL	A MapInfo boundary file containing supplementary boundaries. These boundaries represent communities which fall just short of the thresholds used by the model, however, there are a significant number of insured properties in the area to warrant investigation. Currently these boundaries are not included in the final result, however, they are generated for future consideration.
UrbanAreas_CAN_Prim	*National coverage of Primary urban boundaries
UrbanAreas_CAN_Sec	*National coverage of Secondary urban boundaries
UrbanAreas_CAN_Ter	*National coverage of Tertiary urban boundaries
UrbanAreas_CAN_NAD83	*National coverage of urban boundaries in a NAD83 projection system
UrbanAreas_CAN_Prim_NAD83	*National coverage of Primary urban boundaries in a NAD83 projection system
UrbanAreas_CAN_Sec_NAD83	*National coverage of Secondary urban boundaries in a NAD83 projection system
UrbanAreas_CAN_Ter_NAD83	*National coverage of Tertiary urban boundaries in a NAD83 projection system

*File generated only when all ten provinces are selected to be processed as defined in the Configuration file.

Configuration File

The configuration file is a MapInfo table containing key information required by the model such as file names and locations of input data as well as variable settings. All information in the table is subject to change over time which in the absence of the configuration file, would result in increase the total effort necessary to maintain the model. The following list outlines the types of information stored in this file.

- Root folder location
- Error flag
- Input data
- Threshold values for boundary creation
- Email
- Completion dates

The model's configuration file is located in the main program folder ...\UB_Model\1_Program, and has the following table structure. The name of this file is hard coded into the model and therefore cannot be change without resulting in a failure.

Field Name	Format	Description
ID	Char(8)	Unique ID for each row in the file that is utilized by the model. Allows the program to access specific information within the table
Var	Float	A numeric value required for certain components of the program
Description	Char(68)	A brief description
Date	Date	Applied to rows pertaining to input data. Refers to the file date of the data set that was used in the previous run. Allows the program to determine if the input data set is new.
File Status	Char(15)	Applied to rows pertaining to input data. Indicates if the input data is the same as previous run of the model or if it is new.
File Path	Char(106)	Applied to rows pertaining to input data. Indicates where the data set can be found within the model's folder structure.
File Name	Char(48)	Applied to rows pertaining to input data. Indicates the input file name that will be used by the model.
Instruction1	Char(254)	Any additional information. Typically this contains a Drop statement used by the program that removes fields from a data set.
Instruction2	Char(254)	Any additional information.

All fields in the configuration file are used by the model in some fashion and will result in the model's failure if absent. Some fields are automatically populated by the model and do not need any special attention. Other fields are updated manually and are specific to the model's location or contain names of the input data. If there was a need to recreate this file, it is critical that the ID field be preserved. The following table contains a record of the unique IDs used at the time of the model was created. A backup copy of the configuration file can be found at the end of this document.

ID	Var	Description
FP1	0	File path - root folder location for the Model
FP2	0	?
ERR	0	Trigger when an error is detected in any program component
GNW	0	Genworth property point file
RDS	0	Road network file
PP1	0	Postal Points Location
MUN	0	Municipality boundaries
POP	0	Manifoid population data
RoadCnt	25	Threshold of roads needed to build a boundary
BuffCnt1	20	Buf Count where density >= RoadCnt (Purpose 1)
RoadCnt2	15	Lowest Buf Count threshold
CertCnt	100	Number of GNW points required to keep a supplementary boundary
UpGrdDis	2	Upgrade Distance (KM) - Set to 0 for no upgrading
RUN_AB	1	Set Var = 1 to process this province
RUN_BC	1	Set Var = 1 to process this province
RUN_MB	1	Set Var = 1 to process this province
RUN_NB	1	Set Var = 1 to process this province
RUN_NL	1	Set Var = 1 to process this province
RUN_NS	1	Set Var = 1 to process this province
RUN_ON	1	Set Var = 1 to process this province
RUN_PE	1	Set Var = 1 to process this province
RUN_QC	1	Set Var = 1 to process this province
RUN_SK	1	Set Var = 1 to process this province
E_Frm	0	Emails will be sent from this address (no quotes)
E_To	0	Email recipient list (include quotes)

Although the configuration file contains nine fields of information not all fields are required by the model. The following provides a description of the differ types of information included in the configuration file and outlines any dependencies.

Root Model Location

The first record in the configuration file (FP1) contains the file path of the model's folder location. It should exclude the name of the model folder (UB_Model) and must have a back slash at the end of the character string.

Although the other fields may contain information, only FilePath field is used by the model. The field length has been set at 106 characters which can be expanded if required with no negative effects on the model. This should be done inside of MapInfo Professional. The maximum character length in MapInfo is 254. If more than 254 characters are required, then additional modifications to the program will be required.

ID	Var	Description	Date	FileStatus	FilePath
FP1	0	File path - root folder location for the Model	12/12/2010		Z:\

Error Flag

The error flag field (ERR) is used as a means of communication between each of the model's program components and the main program. Its purpose is to indicate whether an error has been encountered by a component. It does this by inserting a value of 1 into the Var field. The main program continually checks this field and if it detects a value of 1 it shuts down the program. Although other fields may have information, only the Var field is used by the model.

	ID	Var	Description	Date	FileStatus	FilePath	FileName
■	ERR	0	Trigger when an error is detected in any program con	12/12/2010			

Input Data

The next five records in the table contain information for each of the specific input data sets. The fields used by the model are ID, FileDate, FileStatus, and FileName.

- **ID** – Unique ID used by the model to identify that particular record in the table so that it can read in its specific information
- **Var** – Not applicable.
- **Description** – Brief description.
- **FileDate** – provides the date of input file last used by the model. This cannot be blank when the model is run, however, can be set arbitrarily to an earlier date. The date format is mm/dd/yyyy.
- **File Status** – indicates if the input file has not changed since the model was last run or if it is new. This field is updated in the Check Files stage.
- **File Path** – Not applicable.
- **File Name** – identifies the name of the input file the model should access. The model expects a certain file type (.TAB, .TXT, or .CSV) and will fail if something different is encountered. Refer to the Check Files section for details.
- **Information1** – Blank.
- **Information2** – Blank.

ID	Var	Description	FileDate	FileStatus	FilePath	FileName	Instruction1	Instruction2
GNW	0	GNW property point file	05/09/2018	NEW	n.a.	GNWPropertyPoints01102018.TAB		
RDS	0	Road network file	06/08/2016	NEW	n.a.	RoadsLine.TAB		
MUN	0	Municipality boundaries	11/08/2016	NEW	n.a.	MunicipalitiesRegion.TAB		
EPC	0	Enhanced postal code points	06/15/2016	NEW	n.a.	EnhancedPostalPoint.TAB		
POP	0	Manifold population data	02/05/2016	NEW	n.a.	PopTotal.TXT		

Email Recipients

The configuration file contains two records for the email addresses of the sender and the recipients, E_Frm and E_To. The model expects to find these email addresses in the Instruction1 field. No other fields are referenced by the model. The sender's email address must not be included in quotations. The list of recipients must be in quotations with each email address separated by a semicolon, for example, "james.harvie@genworth.com; soheil.barkhodaee@genworth.com".

	ID	Var	Description	Date	File	File	FileN	Instruction1	Ins
☐	E_Frm	0	Emails will be sent from this address (no quotes)					soheil.barkhodaee@genworth.com	
☐	E_To	0	Email recipient list (include quotes)					"GNWMORTCANITPropertyHUBMapInfoAdministrators@genworth.com"	

It is expected that the recipients will be part of the following distribution list set up by Genworth using the email address as follows.

Email Address: GNWMORTCANITPropertyHUBMapInfoAdministrators@genworth.com

The display name for this distribution list within the Genworth contact list is "@GNW MORT CAN IT Property HUB Map Info Administrators"

Log file

Each time the model is run a log file is created in the model's program folder called LogFile.TAB. This log file is continuously updated while the model is running, recording significant milestones during its progress.

For convenience, each time the file is appended with new information, a copy of it is saved as a text file with the named LogFile.CSV. This enables you to view the results in a text editor of your choosing as well provides a means of tracking changes as they occur. This can be done in a program like TextPad which will beep each time a new version of the file has been saved.

Each time the model is run both the .TAB and .CSV versions of the log file are overwritten; however, a copy of the log file is saved to the results folder for future reference. The following table is small sample of what the log file looks like.

StartDate	StartTime	ProcName	Prov	Message1	Message2	Message3
7/11/2018	14:32.0	Model Start	-			
7/11/2018	14:35.0	Model Start	-	Email Notification Sent	Error flag in Config File is reset to zero	Date/Time format initialized
7/11/2018	14:35.0	Result Folder	-	Does Not Exists	Creating new folder	
7/11/2018	14:35.0	Model Start	-	Result folder created with today's date		
7/11/2018	14:35.0	Model Start	-	Deleting and re-creating XTemp folder from last run		
7/11/2018	14:37.0	Model Start	-	Picking Provinces	Provinces to be run: BC,	
7/11/2018	14:37.0	Data Check 1		Road Network: Found		
7/11/2018	14:37.0	Data Check 1		Muni Boundaries: Found		
7/11/2018	14:37.0	Data Check 1		Postal Points: Found		
7/11/2018	14:37.0	Data Check 1		Population Data: Found		
7/11/2018	14:51.0	Data Check 1		GNW Property Points: Found		
7/11/2018	14:52.0	Data Check 2		Road Network: NEW FILE		
7/11/2018	19:56.0	Data Check 2		Muni Boundaries: NEW FILE		
7/11/2018	19:56.0	Data Check 2		Postal Points: NEW FILE		
7/11/2018	20:16.0	Data Check 2		Population Data: NEW FILE		
7/11/2018	20:32.0	Data Check 2		GNW Property Points: NEW FILE		
7/11/2018	20:34.0	Data Preparation		Appending province to PopData	Detected new file: Processing will continue	Provincial PopData files saved
7/11/2018	26:27.0					
7/11/2018	26:27.0	Folder Management	BC	Deleting and replacing all provincial folders from last run	Complete	Creating model area folders
7/11/2018	26:39.0	Data Preparation	BC	Municiplaities	Detected new file - Creating Munis/Model Area lookup table	
7/11/2018	26:49.0	Data Preparation	BC	Roads		Provincial road files saved
7/11/2018	11:23.0	Data Preparation	BC	Pop Data	Dividing by province	Complete
7/11/2018	11:55.0	Data Preparation	BC	Postal Points	Detected new file: Processing will continue	Postal Point table saved

Email Notifications

Email notifications are sent out at specific times during the running of the model. The following describes when each message will be sent and any related conditions.

1. Model start – notifies when the model was initiated
2. Anytime an error is trapped causing the model to fail. In most cases the message will include the error message produced by MapInfo Professional indicating the failure point.
3. Model completion – notifies when model is finished

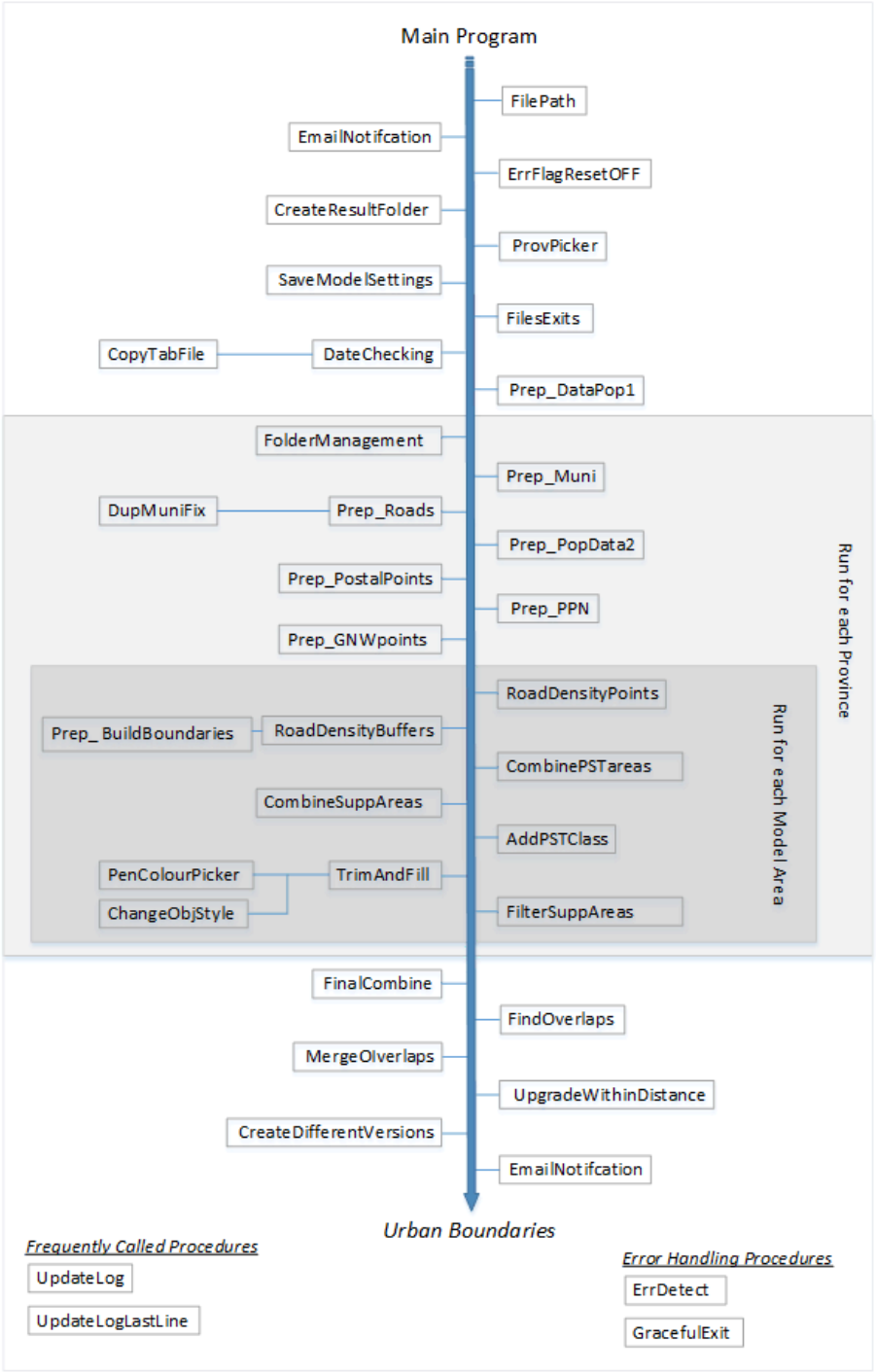
The sender's email is can be found in the Instruction1 field of the configuration file for the record where ID = E_Frm. The recipient is a distribution list that is administered by Soheil Barkhodaee. It can be found in the Instruction1 field of the configuration file for the record where ID = E_To.

Email Address: GNWMORTCANITPropertyHUBMapInfoAdministrators@genworth.com

The display name for this distribution list within the Genworth contact list is "@GNW MORT CAN IT Property HUB Map Info Administrators"

Main Program Description

All program components are built around a single program called UBM_Main. The Main program acts a framework that controls when procedures are called and how many times depending if it's a Provincial or Model Area specific procedure. The diagram shown below illustrates the order in which procedures are called including which are provincial or model area level procedures. The remainder of this section provides a high-level description of the Main program as well as some insights about each procedure.



Model Preparations

- Sets the size and position of Message window within MapInfo Professional. This is only apparent when the model is run directly from an open instance of MapInfo. The Message window is updated periodically providing feedback on the model's progress.

- Progress bars are turned off. This only has impact when the model runs in an open instance of MapInfo. With progress bars turned off, considerably time can be saved because the program does not have to continually asses how much work is left to be done.

- The Startime and LastCheckTime variables are initialized using the timer() function in MapBasic. This effectively identifies when the model began running and is used for determining how long it takes for the model as well as its procedures to run. These variables are used in the TimeCheck procedure

- Creates a new log file for the current run of the model. The Log file is created in the MapInfo's native .TAB file format which is updated at key points during processing providing feedback to the model's progress. If errors occur, then information about the error is also entered. This file is initially stored in the 1_Program folder with all other program components, however, when the model has completed all processing, a copy of the Log file is stored in the 4_Results folder. The model also generates a .CSV version of the Log file which allows for easier access to its contents while the model is running. The following is a description of the fields included in the file.
    - `StartDate Char(10)`
    - `StartTime Char(10)`
    - `ProcName Char(25)`
    - `Prov Char (2)`
    - `Message1 Char(254)`
    - `Message2 Char(254)`
    - `Message3 Char(254)`
    - `Message4 Char(254)`
    - `Message5 Char(254)`
    - `Message6 Char(254)`

- The FilePath procedure is run. This code loads into memory all file paths used by the model to open and save data files. The first operation performed is to open the configuration file where the file path of the root directory for the model is recorded. Using this file path, separate variables can be built for each folder location the model uses. The added benefit to this approach is that it makes the model more portable since there no specific file path references in the code. If the model is to be moved then the only change required is the root directory information stored in the configuration file.

- A "Start" email notification is sent to all stake holders that the model has begun. The specific type of message is defined by setting the EmlMsgType to "Start" before calling the EmailNotification procedure. The body of the message is hard coded and located in the EmailNotification procedure. The sender and recipients of the email are stored in the configuration file.
- Next, the error flag field in the configuration file is set to zero by running the ErrFlagResetOFF procedure. Whenever the model detects a processing error the program sets this field to 1 which initiates a small number of operations before it shuts itself down. These operations include capturing the error message provided by MapInfo and sending an email notification to stake holders. If the previous run of the model had triggered an error, then the error field will still be set to 1. This procedure simply resets it back to zero.

- Next, the start date and time is captured.

- The CreateResultFolder procedure is initiated. This procedure reformats the current date string so that two digits are used for both the month and day components, four digits for the year, and then re-arranges the components in the form yyyymmdd. Once complete, a new folder is created in the 4_Results folder with the name "UrbanAreas_yyyymmdd". The purpose of rearranging the date is so that the results of each run of the model are stored in chronological order.

- The ProvPicker procedure is initiated. This procedure determines which provinces the model will run. It does this by opening the configuration file and selects those fields where the User has entered a value of 1 in the "Var" field. An array is then populated with the province abbreviations for these records. The model then reiterates code for each of the provinces contained in the array.

- The SaveModelSettings procedure is initiated. This procedure creates a new table containing a listing of provinces included in the model run,

Data Checking Procedures

- The FilesExits procedure checks each of the input data sets to determine if the correct file exist and if it can be opened in MapInfo Professional. If the model encounters and error, this error is captured and saved in the configuration file until all input files have been evaluated. Once complete, the model checks the configuration file if there were any errors. If errors were found, then the model shuts down and reports the errors to the Log file.

- Next, the DateChecking procedure checks the file date of all input data and determines if it is new or the same as the previous run of the model. The date and the new/old determination are then stored in the in the configuration file. If the input data is the native MapInfo file format, then only the .DAT file is evaluated as this is considered to be the most accurate than the other file components .TAB, .MAP, .ID, and .IND as these can have different dates. If the file is found to be new then all file components are copied to the model's working folder, \3_Working\xDataStore\ using the CopyTabFile procedure.

Data Preparation

- There are seven procedures that prepare input data for use by the model, only the Prep_DataPop1 procedure is run prior to dividing the data into the respective provinces. The main purpose of this procedure is to add a new field to the Population data and update it with the two-letter province abbreviation so that it can be divided by province later in the program.

- Before any additional data preparations are performed, a For…Next loop is set up so that remaining process steps described in this section repeat for each province as specified by the configuration file.

- The FolderManagement procedure is called. This procedure deletes all "working" data files and folders for the target province and then recreates the folder structure for the current run of the model. This ensures that any files in this folder are from the current run of the model. Please note that only the target province is affected. If the model is not running on all provinces, then files for those provinces not included in the current run of the model will be from a previous run.

- As part of recreating the "working" folder structure for the target province, it creates a separate folder for each model area. Each province is divided into several smaller geographic areas called Model Areas. All processing of urban boundaries is performed at this geographic level, and when all areas have been completed, they are combined to form provincial and national coverages. The model area folders are located in the …\3_Working*prov*\xAreas\ folder. For more information on Model Areas refer to Section 4 Description of Model Areas.

- Prep_Muni – Opens the Municipality data set and selects all records for the target province. The selection only includes those fields required by the model. The resulting file is saved to the provincial folder with the name ModelAreaLUT. The following tasks are then performed.

    - If the target province is BC then some modifications are made to specific polygons in this boundary file that rectify future errors.
    - Two fields are added, one for the name of the model area in which it located and the other for the number of objects make up each municipality.
    - A check is performed to determine if a municipality is made up of more than one region and if any of those regions fall inside a different model area. If this found to be true, then the boundary is disaggregated and the different parts are then reassigned to the respective model areas.

- Prep_Roads – Opens the Roads data set and selects all records in the target province. The selection only includes those fields required by the model and limits the road type to those with a CARTO class between 1 and 5. The results are saved to a file called Roads_Carto1to5.TAB and is saved in a projection system that uses metres as its distance unit. A check is performed to determine if there are multiple municipalities with the same name. If any are found, then several steps are performed to change the model area name for those roads affected so that they get assigned to the appropriate area.

- Prep_PopData2 – Opens the Population data and selects all records for the target province and saves a copy to the province's working folder.

- Prep_PostalPoints – Opens the Postal Point data sets which contains a point object for each six-digit postal code. All records for the target province are selected and saved to the province's working folder. Only required fields are included in the selection. A new field is added and updated with the population amount contained in the Population data set. Finally, the postal points are divided into their respective model areas and saved to the working folder.

- Prep_PPN – Two custom built data sets are divided into the model areas for the target province. The first data set is a Population Place Name (PPN) file containing a point object for communities across the country. This table has been manually edited to reduce the number of points in a manner so that an even distribution of points is located in each model area. The second custom file is a Voronoi polygon layer generated from the PPN file mentioned above. Both custom files are stored in the 1_Program\SupportFiles\ PPNVoronoiFiles folder.

- Once the input data has been prepared for the target province, the process of generating urban boundaries begins. This done by setting up a Do ...Loop that cycles through each model area, running a series of procedures that create the boundaries. The following is a description of those procedures in the order they are run.

- Prep_GNWpoints – Opens the national coverage of the Genworth data set containing property points. Selects all records for the target province and saves them to the provincial working folder. A new field is added to the provincial version of the table call ModelAreaName which is updated with the model area name the property falls within. Finally, the point file is divided into their respective model areas.

- A Fetch...Next loop is set up so that the following procedures are run for each model area in the province. The loop retrieves each model area name from the ModelAreaTable.TAB file for the target province.

- RoadDensityPoints – The main purpose of this procedure is to create a single point file representing the centroid of all roads in the model area from which density buffers can be generated. This is divided into four steps. The first optimizes the short roads point file by aggregating together those which are within 100m of each other. The number of points aggregated at each location is retained. The second step generates a point file representing long roads, those determined to be greater than 1km in length. Because most of these roads are quite long, points are placed at an interval of 500 metres. The third step aggregates the long road point file at a 500m distance. This removes duplication that occurs at the beginning and end of each line segment. The fourth and final step is to

combine the short and long road points into a single data file (RoadDensityPoints.TAB), saving it to the working folder for the model area.

- RoadDensityBuffers – This procedure generates two separate urban boundary files. The first being the main output file containing urban boundaries that eventually get classified into primary, secondary, and tertiary (PST). The second output file is a boundary layer consisting of those boundaries which failed the threshold levels that would have given it a PST classification but contain a large number of properties insured by Genworth. These areas are referred to as being supplementary boundaries.
- In either case the methodology for building the boundaries is similar. A circular buffer is created around each of the road density points previously generated, where each buffer has an area of approximately 1 square kilometer. Next, buffers are updated with the number of roads found within it, those which contain fewer roads than the threshold amount are then removed and considered to be supplementary boundaries. For either output file (PST or supplementary) the BuildBoundaries procedure is called. This procedure is responsible for several operations which include merging buffers together to form boundaries, updating fields with pertinent information, removes holes within the boundary, and adding community names and population amounts.

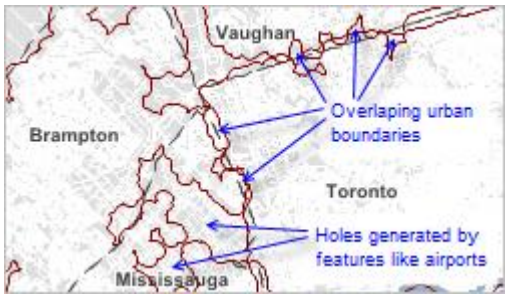
- CombinePSTAreas – This procedure collects the urban boundaries for each model area and adds them to a provincial level data layer that is stored in the province's working folder. The output file name is "UrbanAreas_ *prov*.TAB".

- CombineSuppAreas – This procedure collects the supplementary boundaries for each model area and adds them to a provincial level data layer that is stored in the province's working folder. The output file name is "SuppBoundsALL_*prov*.TAB".

- AddPSTclass – Once all models areas have been processed for the target province then the urban classification (Primary, Secondary, and Tertiary) is applied to each boundary. The thresholds that defined these classes are not controlled by the configuration file but hard-coded into the program. The object style is also modified to reflect the classification.

    - Primary: `TotalPop>=10000 and BuffCount >=100`
    - Secondary: `(TotalPop>=4000 or TotalPop=0) and BuffCount>=60`
    - Tertiary: `All remaining boundaries`

- TrimAndFill – Because urban boundaries are generated at the model area level, artifacts can be introduced when results are merged together. These typically show as overlapping boundaries from two or more adjacent model areas, or as a hole. The TrimAndFill procedure edits those areas so that overlaps are removed, and holes are filled where appropriate. To guide this process, a custom data layer was created that identifies where a boundary needs to be trimmed and where holes need to be fill.



The model can create overlapping urban boundaries where two or more Model Areas meet. Holes can be generated inside an urbanized area where the road density is low.

Results after trimming overlapping areas and filling holes in the urban boundaries creating a cleaner looking final product.

- A copy of the urban boundaries for the target province are then saved to the Results folder. Any unneeded fields in this version of the data layer are removed.

- FilterSuppBoundaries – This procedure identifies those supplementary boundaries that contain a specified number of GNW properties. Once these boundaries have been identified, they are further analyzed to determine if any are overlapping or contained fully within another. If any are found, then the model removes the largest region leaving the smaller one(s) intact. The GNW property count used as the selection criteria is stored in the configuration file (CertCnt).

MOVES TO NEXT PROVINCE

If there are additional provinces to be processed, then the model moves to the next province in the process list as defined by the configuration file. If all provinces are complete, then the model begins the final process steps.

- FinalCombine – Once all provinces have been processed then the urban boundaries for all provinces are combined into a single data layer. If all 10 provinces were selected, then the file name would be “UrbanAreas_CAN.TAB”. If a smaller number of provinces was selected, then resulting file name would be “UrbanAreas_PROV.TAB”. Please note that results are stored in a folder named with the date in which the model was initiated. This folder will contain separate data layer for each province as well as one containing all provinces.

- A copy of the log file is then copied to the results folder

- Finally, an email is sent to all stake holders that the model has successfully completed. The email list for these stake holders is stored in the Configuration file.

Procedure Description

ErrDetect

This procedure is run after each of the major function calls. Its main purpose to check if an error has occurred in the program or if the User has manually triggered a model shut down. The model was designed so that if any error occurs it is trapped so that the program does not crash. Once trapped then any error message are captured and stored in the Configuration and Log files, and then the program is shut down gracefully.

- Checks if the Configuration file is open, if it is then the program continues. If it is not, then it is opened.
- Checks the error flag field in the configuration file to determine its value. The value is stored in a variable called ErrFlag1 and will be either 0 or 1.

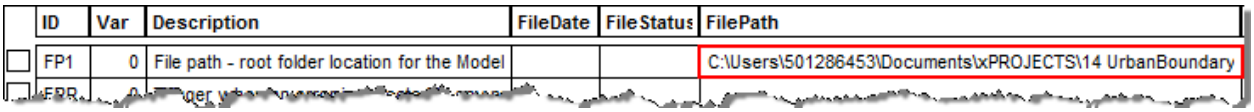
- Next, the ForcedShutdown_Table.TAB is opened. This table is used to manually shut down the model. See also section 0 ErrFlagResetOFF
- The error flag field is queried to determine its value. The value is stored in a variable called ErrFlag2 and will be either 0 or 1.

- Both ErrFlag variables are evaluated, beginning with ErrFlag2 and then ErrFlag1. If either of these variables have a value of 1, then the error message is captured and stored in both the Configuration and Log Files. The program is then gracefully shutdown. If both these variables have a value of zero, then there are no errors detected and the model continues

FilePath

The FilePath procedure is the first to be called when the model is initiated. Its purpose is to read into a variable the specific location the model is located on the computer system as well as other important locations used frequently by the model. The use of a variables to store file path locations has several advantages, the main one being the convenience of having a short and more descriptive name making it easier to read and understand the program's code. The following are some important points to know about the FilePath procedure.

- All variables that represent a file path have "FP_" at the beginning of the name. For example, the variable for all result data files is call "FP_Results"
- The only input the procedure needs is the location of the main program folder. This is retrieved from the "FilePath" field, in the first row of the configuration file.
- The file path should include the drive letter and the file path but exclude backslash and the model's folder name as they are included when the variables are defined, see image below.



ID	Var	Description	FileDate	File Status	FilePath
FP1	0	File path - root folder location for the Model			C:\Users\501286453\Documents\xPROJECTS\14 UrbanBoundary

- The file path is retrieved by first selecting the record in the configuration file with an ID = "FP1". Once this record is isolated then the text in the "FilePath" field is retrieved and stored in a variable call FP_Root, see below.

```
Select * from ConfigFile where ID="FP1" into TT1
    FP_Root = TT1.FilePath
Close Table ConfigFile
```

- Once the location of model's folder is known then all other locations can be defined by simply building upon text represented by FP_Root. For example, the variable locating the Results folder would be as follows.

```
FP_Results = FP_Root + "\UB_Model\4_Results\"
```

- Please note that changing the names of those folders used by the model will cause a failure.

Because this is the first procedure to open and read the configuration file, additional code has been added to determine the value of the SKIP flag. If this value is equal to 1, then the model runs the SkipToFinalCombine procedure. This procedure skips all the steps which build the urban boundaries for each province and moves directly to those procedures which combine results from each province and generates the final output of the boundaries.

```
Select * from ConfigFile where ID="SKIP" into TT1
    vSKIP = TT1.ID
```

EmailNotification

Email notifications are sent out at specific times during the running of the model. The following describes when each message will be sent and any related conditions.

1. Model start – notifies when the model was initiated
2. Anytime an error is trapped causing the model to fail. In most cases the message will include the error message produced by MapInfo Professional indicating the failure point.
3. Model completion – notifies when model is finished

The sender's email is can be found in the Instruction1 field of the configuration file for the record where ID = E_Frm. The recipient is a distribution list that is administered by Soheil Barkhodaee. It can be found in the Instruction1 field of the configuration file for the record where ID = E_To.

Email Address: GNWMORTCANITPropertyHUBMapInfoAdministrators@genworth.com

The display name for this distribution list within the Genworth contact list is "@GNW MORT CAN IT Property HUB Map Info Administrators"

Notes:

- The model leverages an external .EXE built by Soheil Barkhodaee. Two files must be present in the program folder email.config and email.exe
- The executable expects to have a .TXT file as an input. It must contain the necessary information in a specified order.
- Each time a message is sent from the model a text file is built with the required information

Process Steps building Email Command Line

- Creates and opens a .BAT file called DeletemeEmail.BAT. This file is stored in the main program folder.

- Next, the first part of the command line containing the file path of the Email.exe file and the network location are concatenated

```
LocAndCmd = FP_Master + "email.exe smtp.genworth.net 25"
```

- Retrieves the sender and recipient email addresses from the configuration file. Each is stored in separate variables that will be later inserted into the final command line.

```
Select * from Config where ID="E_Frm" into TT1
EmlFrm = TT1.Instruction1

Select * from Config where ID="E_To" into TT1
EmlTo = TT1.Instruction1
```

- The subject and the message body are then defined based on the type of message specified by the program. There are three message types, Start, End, and Custom. For the Start and End message types the subject and message body are predefined and are automatically assigned to variables (EmlSubj and EmlMsg). For the Custom type the EmlSubj and EmlMsg variables are defined at a place outside of the EmailNotification procedure where a specific error is trapped.

- Once all command line variables have been populated, then the command line is built

```
EmailCmd=LocAndCmd + EmlFrm +" "+ EmlTo +" "+ EmlSubj +" "+ EmlMsg
```

The command line is then inserted into the DeletemeEmail.BAT file. The .BAT file is then run resulting in the message being sent.

ErrFlagResetOFF

This procedure simply resets the error flag the model uses to shut down the model when an error has occurred in the previous run of the model. The reset is applied to two locations since there are two sources from which the shutdown can be initiated.

The first location is in the configuration file, specifically, the record where ID = ”ERR”. When an error is encountered, the program captures the error message produced by MapInfo and copies it to the Instruction2 field for this record. Next, it updates the Var field with a value of 1, changing it from a value of 0. Each time the ErrDetect procedure is called it checks to see if this value has changed, if a 1 is found then the model is shutdown.

The second location is in the ForcedShutdown_Table. This table provides a means for the User to manually shut down the program by simply changing the ErrorFlag field from a 0 to a 1. The ErrDetect procedure also checks this table to see if a shutdown is required.

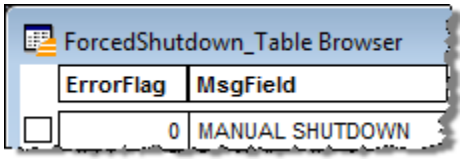


Image shows the single record contained in the ForcedShutdown_Table. When the ErrorFlag field is set to 1 the model will shut down all processing.

TimeCheck

The TimeCheck procedure is called numerous times throughout the model. Its main purpose is to post to MapInfo's Message window the elapsed time used to run different sections of the model. These postings include minutes and seconds and are only seen if the model is run in an open MapInfo window.

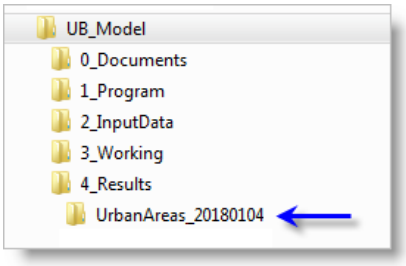
Two different elapsed time spans are calculated, the first is the time between when the model was started and the time the TimeCheck procedure is called. The second, is the elapsed time from when the TimeCheck was last called. To specify which of the two "elapsed times" to display, the TimeCheckType variable must be defined before the CheckTime procedure is called, see below.

```
TimeCheckType=1    Call TimeCheck
```

To return the elapsed time from when the model was started then set the TimeCheckType =0. To return the time from when the CheckTime was last called then set the TimeCheckType to any value, the model consistently uses 1.

CreateResultFolder

This procedure creates a folder location where all results from the current run of the model are stored. The folder name always contains the text string "UrbanAreas_" followed by the date in which the model was initiated and saved to the "\4_Results\" folder.



An example of an output folder created for a model run performed on January 4, 2018

This procedure is divided into two sections, the first formats the date string used in the folder name and the second creates the folder.

It is necessary to reformat the date string because the MapBasic returns the date as M/D/YYYY which results in confusion for the User when several of these result folders are sorted within a File Explorer window. The solution is to parse the date string into its three components, insuring that two digits are used for both the Month and Day, and then rebuild the date string as YYYYMMDD. For example, "4/10/2018" would be transformed to "20180410"

The reformatting of the date string tests for four different possibilities depending on where the slashes "/" occur within the string.

1. If the string has a slash in the second and fourth position, then the format provided is m/d/yyyy
2. If the string has a slash in the third and fifth position, then the format provided is mm/d/yyyy
3. If the string has a slash in the third and sixth position, then the format provided is mm/dd/yyyy
4. If the string has a slash in the second and fifth position, then the format provided is m/dd/yyyy

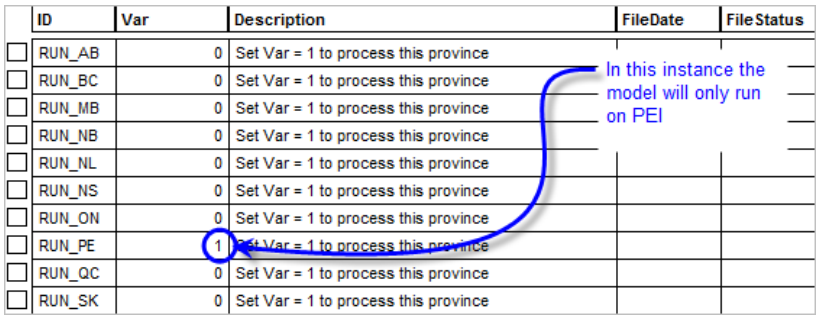
After detecting the specific date then the specific month/day/year components can be isolated and where necessary an extra zero appended. The three date components are then rearranged so that the date string reads "yyymmdd".

The final step is to create the new file folder. Before doing so, a test is performed to see if the folder already exists. This would occur if the model was initiated twice in the same day. If this is the case, then there is no need to create a new folder, the existing the output files are overwritten. If the folder does not already exist, then the program proceeds to create the folder.

The "Folder Exist" and "Folder Create" operations are included in a .NET library made available, royalty free, by Pitney Bowes (MapInfo).

ProvPicker

This procedure identifies which provinces will be analyzed by the model. It does this by opening the configuration file and selecting all records where the ID field contains the word “Run” and the “Var” field is equal to 1. The image below shows all records in the configuration that meet this condition, notice that there is one for each of the provinces that is of interest to Genworth. By manually editing the “Var” field the User can control which provinces an urban boundary is created.



ID	Var	Description	FileDate	FileStatus
RUN_AB	0	Set Var = 1 to process this province		
RUN_BC	0	Set Var = 1 to process this province		
RUN_MB	0	Set Var = 1 to process this province		
RUN_NB	0	Set Var = 1 to process this province		
RUN_NL	0	Set Var = 1 to process this province		
RUN_NS	0	Set Var = 1 to process this province		
RUN_ON	0	Set Var = 1 to process this province		
RUN_PE	1	Set Var = 1 to process this province		
RUN_QC	0	Set Var = 1 to process this province		
RUN_SK	0	Set Var = 1 to process this province		

The specific code used for selecting the provinces is shown below. TT1 is the name of the temporary table used to store the results of the query.

```
Select * from ConfigFile where ID like "Run%" and Var=1 into TT1
```

Once the selection is made, the number of records in the selection is determined and stored in a variable called ArraySize. This value is used to specify how large the ProvAbb() array needs to be, see below.

```
ArraySize = TableInfo(TT1,8)
Redim ProvAbb(ArraySize)
```

The final step is to populate the ProvAbb() array with the two letter abbreviation for each province included in the analysis. This is a critical step because the contents of this array control the flow of the program, but it is also involved in selecting and messaging input data. The basic approach is to step through each record in the TT1 temporary table and retrieve the last two characters in the string located in the ID field. These two characters is the two letter abbreviation for the province which is inserted into the ProvAbb() array. The code used to perform this operation is shown below.

```
y=1
If ArraySize >0 then
    For y = 1 to ArraySize
        Select * from TT1 where Rowid= y into TT2
        Prov = Right$(TT2.ID, 2)
        ProvAbb (y) = Prov
    Next
```

It should be noted that if the “ArraySize” equals zero (none of the provinces were selected) then an error is recorded, and the model shuts down gracefully.

SaveModelSettings

This procedure is one several that are executed shortly after the model is run and before the model begins checking/processing input data. Its purpose is to capture the user defined settings used for the current run of the model. Results are stored in the results folder in a MapInfo table format.

- A table is created called “ModelSettings” containing three generic character fields, Detail1 Char(15), Detail2 Char(10), and Detail3 Char(68). The table is stored in results folder.
- The Configuration file is opened

- In the first row ModelSettings table, the word “Processed” is entered into the Detail1 field. The number of provinces being processed is entered into the Detail2 field. This value is determined when then model is initiated and is stored in the “ArraySize” variable. The two letter abbreviation for each provinces being processed is entered into the Detail3 field. This information is stored in a string called ProvListString which is created when the model is first initiated.
- A blank row is inserted into the ModelSettings table.

- Next, the row in the configuration where ID=RoadCnt is selected
- From this selection the values contained in the ID field, the variable field (Var), and description field are retrieved and stored into variables.
- These three values are inserted into ModelSettings table

- The previous three steps are repeated for BuffCnt1, RoadCnt2, and CertCnt. The following is an example of the ModelSettings table.

Detail1	Detail2	Detail3
Processed	10	AB, BC, MB, NB, NL, NS, ON, PE, QC, SK,
RoadCnt	25	Threshold of roads needed to build a boundary
BuffCnt1	20	Buf Count where density >= RoadCnt
RoadCnt2	20	Lowest Buf Count threshold
CertCnt	100	Number of GNW points required to keep a supplimenta

FilesExists

The only purpose of the FilesExist procedure is to determine if all four of the input data is available for processing. After the four input files have been assessed and any are found to be missing, then the model is shut down. The basic logic is illustrated below in context to the Roads data.

1. Select from configuration file where the ID =”RDS”.
2. Retrieve the name of the input file from the FileName field.
3. Determine if the input file exists in the “2_InputData” folder
4. If the file is found then the FileStatus field is updated with the word “Found”, otherwise the field is updated with “Not Found”

	ID	Var	Description	FileDate	FileStatus	FilePath	FileName
☐	RDS	0	Road network file	12/12/2010	Found	n.a.	RoadsLine.TAB
☐	MUN	0	Municipality boundaries	12/12/2010	Found	n.a.	MunicipalitiesRegion.TAB
☐	EPC	0	Enhanced postal code points	12/12/2010	Found	n.a.	EnhancedPostalPoint.TAB
☐	POP	0	Manifold population data	12/12/2010	Found	n.a.	PopTotal.TXT

Screen shot of configuration file showing the four records for the input data. The red box identifies the specific file name the model expects. The blue box identifies those fields used to store the file status.

Once the four steps are performed for each of the input data files, a query is performed to determine if any of them had a field status equal to “Not Found”. If this is the case, then the “ERR” record in the configuration file is set to 1 which triggers the program to shut down.

	ID	Var	Description
☐	ERR	1	Trigger when an error is detected

If the program is shut down because one of the input files could not be found, inspecting the configuration file will reveal which file(s) were missing.

DateChecking

The only purpose of the DateChecking procedure is to determine if any of the input data sets are the same as those used in the previous run of the model. If they not the same, then it is presumed the data is new and therefore needs to be copied to the model's working folder. By copying the file to another location allows the model to make modifications while preserving the original. If the dates are the same, then it is not necessary to copy the file allowing the model to move on to the next step saving which saves processing time.

The determination of whether the data set is old or new is done by comparing the input file's modified date to the one the model encountered that last time it was run. The previous modified date for each file is stored in the FileDate field contained in the configuration file.

Date checking for MapInfo files is not as straight forward as it is for other file types. This is because the native MapInfo data file can consist of, between 2 and 5 different file components (.TAB, .DAT, .MAP, .ID, .IND), where each component could potentially have different modified date. To avoid the complexities of checking and comparing the dates of up to five different files it was decided to only perform these tasks on the .DAT file. The decision was based upon the nature of the data provider, when they make changes it always affects the .DAT file.

The model uses a function called GetFileModifiedDate() located in the Kernel32.DLL, provided by Microsoft Windows operating system, to determine the modified date of a data file. It is necessary to use Windows function since there is not an equivalent one available in MapBasic. Within the model, the MapBasic code that defines the GetFileModifiedDate function and allows access to this Kernek32.DLL is found in the 2b_xGetFileDate_v1.MB file.

As mentioned earlier, each of the five input data sets are queried separately but all follow the same logic as illustrated below. This description refers specifically to the road network file but pertains to all input data layers.

1. The record pertaining to roads data set is selected from the Configuration file (where ID="RDS") into a temporary table called TT1
2. The specific file name for the roads data is retrieved from the FileName field, the .TAB extension is dropped from the name.
3. A variable called FullFilePath is populated with the input data location, the file name and the .DAT extension. It is this text string that will be sent to the GetFileModifiedDate() function.
4. The file date from the previous run is retrieved from the FileDate field. This is stored in a variable called PrevFileDate
5. The GetFileModifiedDate() function is called using the FullFilePath variable. The result is stored in a variable called ExistFileDate
6. A query is performed to determine if the ExistFileDate <> PrevFileDate. If the two dates are not equal, then the input file is considered to be new and are copied to the model's working folder \UB_Model\3_Working\xDataStore folder.

The following is the MB code relating to the above description.

```
Select * from ConfigFile where ID="RDS" into TT1
    TrgtFileExt = TT1.FileName               'File name with extension
    StrLenth = Len(TrgtFileExt)-4
    TrgtFile = Left$(TrgtFileExt, StrLenth)   'File name without extension
    PrevFileDate = TT1.FileDate

FullFilePath = FP_Input + TrgtFile + ".DAT"
    If FileExists(FullFilePath) Then
        ExistFileDate = GetFileModifiedDate(FullFilePath)

            If ExistFileDate <> PrevFileDate Then
                FileChanged = TRUE
                Update TT1 Set FileStatus = "NEW"
                Update TT1 Set FileDate = ExistFileDate
                Call CopyTabFile(FP_Input, FP_xStore, TrgtFile)
                Msg1 = "Road Network: NEW FILE"
                MsgNumber = 1    Call UpdateLogLastLine
            Else
                Update TT1 Set FileStatus = "No Change"
                Print "  Roads - No Change"
                Msg1 = "Road Network: No Change"
                MsgNumber = 1    Call UpdateLogLastLine
            End if
    End if
```

If the input file is in the MapInfo file format and it has been determined to be new, then the CopyTabFile procedure is called to make the copy of the file in the working folder.

CopyTabFile

The CopyTabFile procedure is exclusively used by the DateChecking procedure when it is determined that the input data file is new and is in the native MapInfo file format. Its main purpose is to copy the input data file to working folder, specifically "\UB_Model\3_Working\xDataStore".

The procedure is a function that requires three parameters

1. The directory path and file name for the input data file
2. The file name without the extension
3. A text string containing the specific file extension

For each of the MapInfo file types, a query is performed to determine if the specific file component exists, if it does then it is copied to the working folder. The following is the code used to evaluate the .TAB file, this is repeated for each of the other four file components.

```
If FileExists(sourceDir + filename + ".tab") then
       Save File sourcedir + filename + ".tab" As targetdir + filename + ".TAB"
End if
```

Prep_PopData1

The purpose of this procedure is to add and update a province field to the population data. This is necessary because the vendor does not provide this information with their product releases and it is required by the model so that it can properly divide the data into smaller, more manageable groupings. The following is the outlines the workflow.

- Opens the population and the postal points data from the "\Working\xDataStore" folder
- Determines the column number for the field containing the postal code field in both data sets mentioned above. This is done because MapBasic uses a numeric column reference to link tables together which causes errors if the data vendor adds or remove fields from their data sets.
- A new field called "PROV" is added to the population data
- The PROV field is updated with the two-letter abbreviation for the province contained in the postal point data set using the 6-digit postal code field as the link between the tables. The update statement is shown below where ColNum1 is the column reference for the population data and ColNum2 for the postal point data

```
Add Column "PopData" (PROV )From PostalPoints Set To PROV
Where ColNum1 = ColNum2
```

FolderManagement

The purpose of this procedure is to prepare the working\provincial folders for files created in the current run of the model. This preparation involves deleting the previous provincial folder and all its contents and then recreating the folder structure. This procedure is run for each province prior to the model preparing each of the input data sets.

This procedure uses three operations included in a .NET library made available, royalty free, by Pitney Bowes (MapInfo). These operations are *Folder_Exist*, *Folder_Delete* and *Folder_Create* and are contained in the FileFunctionsLib.dll. For more information please see

- Checks if the provincial working folder is present using Folder_Exists function. The function returns either a 0 or 1, where a zero indicates that it does not exist and 1 where it does exist.
- If the folder exists, and for most of the time this will be the case, then the provincial folder is deleted along with all of its contents. A new provincial folder is then created along with the sub folder call “xAreas”
- If the provincial folder does not exist, then the model proceeds to create the provincial and xAreas folders.

Within the xAreas folder there is a separate folder for each model area in the province. Because area names are different for each province they cannot be generically created, the names must be retrieved from the ModelArea table.

- The Model Area table for the target province is opened. This table will be used as a lookup table for each model area name.
- A Do…Loop is set up where the name of each model area is retrieved. Each time a name is retrieved, a folder with the same name is created within the xAreas folder.

```
Result = FOLDER_Create(FP_Wrking + ProvAbb(i) + "\xAreas\" + AreaName, sErr)
```

Prep_Muni

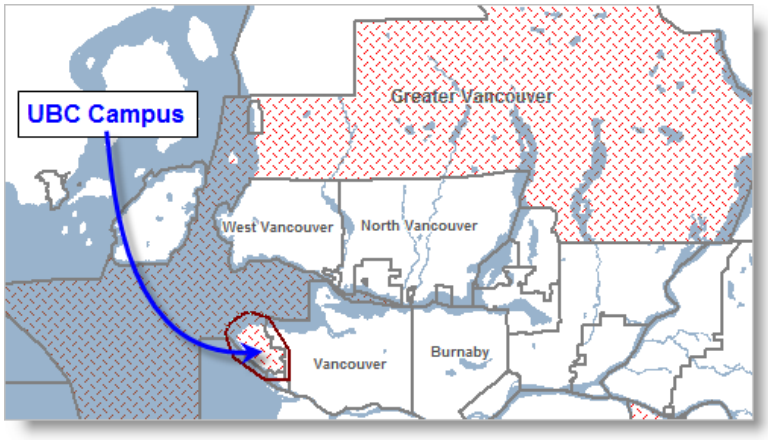
Creates LUT of municipalities and model areas, used for dividing roads into model areas. This procedure is run for each province included to be processed as determined by ProvPicker

- Opens the Configuration file and retrieves the name of the Municipalities data layer. Detects if the file is new or old. If the file is new, then the following steps are performed. If the file is old, then all steps are skipped
- Opens the national coverage of the Municipality data set
- Creates a listing of municipality names found within the target province by performing the following query. Only a small number of fields are required.

```
Select UNIQUE_ID, NAME, PROV from Muni where PROV= ProvAbb(i) order by NAME
into TT1
```

- The temporary table called TT1 is saved to the provinces working folder with the file name ModelAreaLUT.TAB

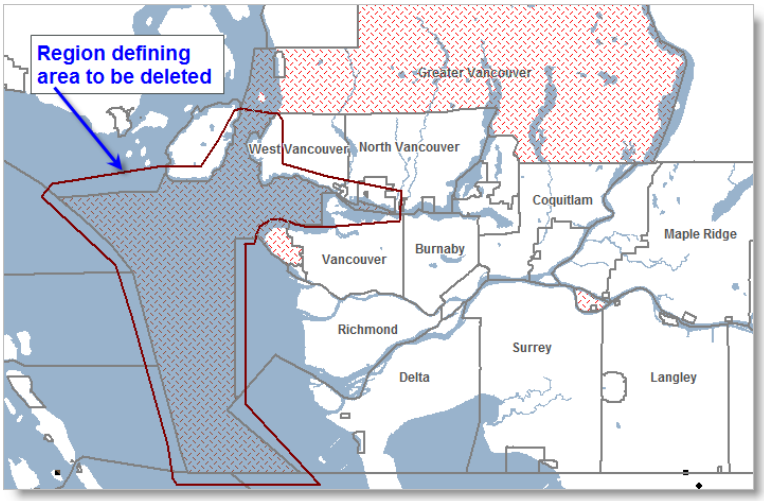
- If the target province is BC, then a small change is made to the boundaries of Vancouver and Greater Vancouver. Specifically, the area to the West of Vancouver, where the campus of the University of British Columbia is located, will be removed from the Greater Vancouver boundary and merged with Vancouver. This is performed because it is the only separated and highly populated area in Greater Vancouver and makes more sense geographically to have it merged with its closest municipality. The image below illustrates what portion of Greater Vancouver will be separated.



- The first operation performed is to export the ModelAreaLUT in MapInfo's native interchange format MIF/MID. This exported file is then opened and saved with the same name as a .TAB file, overwriting the original file. This is performed as workaround to a known issue with MapInfo's Split/Merge features. For some tables an error is encountered when using the Split operation that results in the deletion of the target object.
- The Greater Vancouver boundary is then selected and stored in a temporary table. Once selected the boundary is set as the Target. This is MapInfo terminology to refer to the object(s) that have been marked for a Split or Merge operation.

- Next, the UBC_TrimArea table is opened and the first record is selected. In this data layer the first record is a single boundary that encompasses the area to be removed from Greater Vancouver. The UBC_TrimArea table is stored in the "\UB_Model\1_Program\SupportFiles" folder.
- A Trim operation is then performed resulting in the portion of the Greater Vancouver boundary that is inside of the TrimArea boundary being separated. The Split is done in such a way that the new "UBC" region has the same tabular information as its parent.

Next, a portion of the Greater Vancouver boundary is deleted. This boundary is extremely large and the portion to be removed is entirely in the water. This is necessary because the lookup table (ModelArea_LUT) that assigns municipality boundaries to a model area, uses the centroid of the municipality region to make this decision. Because the Greater Vancouver boundary is so large, its default centroid located forces it to be assigned to the wrong model area. By deleting the portion that is in the water, and completely redundant for this purpose, the centroid falls in the proper location and there for receives the correct model area assignment. The region that defines the area to be deleted, is the second record in the UBC_TrimArea table.

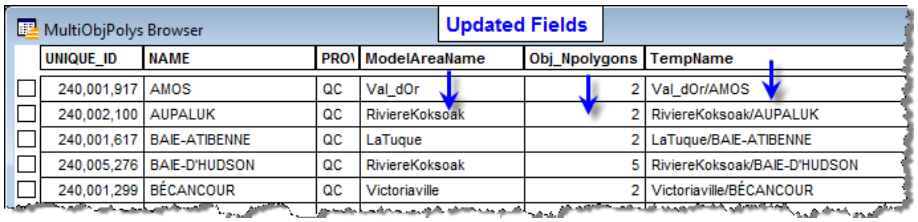


- The Vancouver boundary is selected, stored in a temporary table, and set as the Target.
- Next, the second record is selected from the UBC_TrimArea table. This defines the area that will be deleted from the Greater Vancouver region.
- An Erase operation is then performed.

The following process steps are applied to all provinces. Their purpose is to compensate for complex polygons where one or more of the object are in different Model Areas. This is important to resolve because it is necessary to properly divide the roads into the different areas and if not done properly then the urban boundaries will not be created properly. Effectively you are breaking apart all complex polygons and recombining them with respect to the model area in which they are located.

- The ModelAreas boundary table for the target province is opened. This is a custom boundary file stored "\UB_Model\1_Program\SupportFiles" folder.
- Two additional fields are added to the ModelAreaLUT, one called ModelAreaName and the other Obj_Npolygons
- The Obj_Npolygons field is update with the number of polygons that make up each object in the ModelAreLUT. A query is then performed to determine if there are any records where this value is greater than 1. If any are found, then this indicates multiple polygons are used to define the municipality boundary and therefore could result in future processing errors.
- A separate table is created called MultiObjPolys.TAB. This file is saved to the provincial working folder. These same records are deleted from the ModelAreaLUT file. They will be added back later after some processing.
- All objects in the MultiObjPoly table are then disaggregated. This means that regions made up of multiple polygons are broken up so that each polygon is a separate record in the table. For this operation each polygon has identical tabular information. The table is saved and packed when the operation is complete
- An additional field is added to the MultiObjPoly table called TempName. This field will contain a unique name that identifies the municipality and Model Area the polygon is located.
- The ModelAreaName field is updated with the name of the model area in which the polygon is located.
- Next, the TempName field is updated with a text string that is combination of the model area name and the municipality name.

```
Update MultiObjPolys Set TempName = ModelAreaName+"/"+NAME
```



UNIQUE_ID	NAME	PRO\	ModelAreaName	Obj_Npolygons	TempName
240,001,917	AMOS	QC	Val_dOr	2	Val_dOr/AMOS
240,002,100	AUPALUK	QC	RiviereKoksoak	2	RiviereKoksoak/AUPALUK
240,001,617	BAIE-ATIBENNE	QC	LaTuque	2	LaTuque/BAIE-ATIBENNE
240,005,276	BAIE-D'HUDSON	QC	RiviereKoksoak	5	RiviereKoksoak/BAIE-D'HUDSON
240,001,299	BÉCANCOUR	QC	Victoriaville	2	Victoriaville/BÉCANCOUR

- A new table is created called MultiObjPolys2 using the same field names and projection system as the MultiObjPolys table. This new table is being created to store the results for the next step.
- Objects from the MultiObjPolys table are now combined using the unique name found in the TempName field and inserted into the MultiObjPolys2 table which was previously empty. The following code is used to perform this operation.

```
Create Object As Union From MultiObjPolys
      Into Table MultiObjPolys2
```

```
Group by TempName
Data   UNIQUE_ID=UNIQUE_ID,NAME=NAME,PROV=PROV,
       ModelAreaName=ModelAreaName,Obj_Npolygons=Obj_Npolygons,
       TempName=TempName
```

- Next, the MultiObjPolys2 table is appended to the ModelAreaLUT table using the following line of code. Notice that only five of the six fields are appended. The TempName field is not need and so it is not retained.

```
Insert Into ModelAreaLUT
      (COL1, COL2, COL3, COL4, COL5) Select COL1, COL2, COL3, COL4, COL5
      From MultiObjPolys2
```

- Finally, the Obj_Npolygons field is deleted from the ModelAreaLUT table as it is no longer needed.

Prep_Roads

Dealing with municipality boundaries which have the same name but located in different areas of the province

- Opens the Configuration file and retrieves the name of the Roads data layer. Detects if the file is new or old. If the file is new, then the following steps are performed. If the file is old, then all steps are skipped
- Opens the national coverage of the Roads data set
- Creates a new table consisting of roads located in the target province by performing the following query.

```
Select UID, STREET, CARTO, LEFT_MAF, LEFT_PRV from Roads
      where LEFT_PRV= ProvAbb(i)and CARTO<=5 and SUFTYPE<>"Ramp" into TT1
```

- Only roads with a CARTO classification between 1 and 5 are included. This represent all highways, major and local roads, and excludes features like hiking trails, alleys, ferry routes, and logging roads. The query also excludes roads with a surface type classification as “Ramp” since this will artificially increase road density in undesirable locations. Only required fields are included in the query.
- The resulting table is saved to the provincial working folder with the name Roads_Carto1to5.TAB . During the Save process the projection of the file is changed to a Conformal Projection for North America which uses metres as its distance unit. This is necessary for so that future process steps can be performed more accurately.

- Four additional fields are added to the new roads table,

```
Xcoord Float, Ycoord Float, RoadCount Float, ModelAreaName Char(70)
```

- The LEFT_MAF field is indexed allowing for quitter processing in future steps
- The ColNum1 variable is populated with the column reference of the LEFT_MAF field.
- The ModelAreaLUT for the target province is opened and the ColNum2 variable is populated with the column reference of the “Name” field.

- The DupMuniFix procedure is run. This procedure detects if the there are multiple municipality boundaries with the same name. If any are detected, then a check is performed to see if all boundaries are in the same model area. If they are in different model areas, then the corresponding roads are modified to reflect the model areas they are in. See the procedure description for more detail.

- (ProvRoads) A query is performed to select all roads which have not yet assigned to a model area. A small number records have already had their assignments by the DupMuniFix procedure and this query ensures that those assignments are not overwritten. Results are stored in a temporary table called TT2.
- (TT2) The ModelAreaName field is updated with the name of the model area located in the ModelAreaLUT table using the municipality name in each table as the link.

```
Add Column "TT2" (ModelAreaName) From ModelAreaLUT Set To ModelAreaName Where
ColNum1 = ColNum2
```

The roads are now divided into separate tables, one for each model area. All output files are saved to the working folder specific to the target model area being processed.

- The ModelAreas table for the target province is opened. This table is in the SupportFiles folder and will be used as a lookup table to process each model area.

- A Fetch...Next loop is set up to evaluate each record in the ModelAreaTable table.
    - The name of the target model area is retrieved from the table
    - (ProvRoads) All roads in the target model area are selected and saved to a table called Roads_Carto1to5.TAB.
    - (Roads_Carto1to5) Table is opened and a query is performed to select all records that have CARTO>=2. The results are saved to a table called Roads_Carto1to2.TAB.
    - (Roads_Carto1to2) Table is opened and a query is performed to select all road segments that are <=1km in length. The results are saved to two different tables, the first is called ShortRoads.TAB and the second ShortRoadPts.TAB.
    - The current selection is inverted (Run Menu Command 311) and the results saved to a table called LongRoads.TAB.
    - (ShortRoadPts) Table is opened and the Xcoord and Ycoord fields are updated with the coordinate values of each road's centroid.
    - (ShortRoadPts) All map objects are deleted from the table using a Drop Map statement. The table information is retained.
    - (ShortRoadPts) A new map is added to the table using a specific projection system.
    - (ShortRoadPts) A specific symbol style and colour is specified before performing a Create Points operation which inserts a symbol on to the map for each record in the table using the x,y coordinate contained in the Xcoord and Ycoord fields.
    - (ShortRoadPts) The RoadCount field is updated with a value of 1. This will be used later in the road density calculation.
- Then next record in the ModelAreaTable table is selected and processed using the steps described above until all records have been evaluated.

DupMuniFix

The model assumes that all municipality boundaries are constructed using a single polygon object and that each boundary is entirely inside of a single model area. In reality, there are many municipalities that are comprised of several separate boundaries and that some of them may be located in different model areas. In addition, it is common to find multiple municipalities with the exact same name located in different parts of a province. The purpose of this procedure is to identify those municipalities that exhibit the characteristics described, determine which ones pose a problem for the model, and then rectify both the model area lookup table and the road network file so that the model operates as intended. This procedure is called by the Prep_Roads procedure.

STEP 1 – Creates a list of municipalities that appear more than once in the ModelAreaLUT

- (ModelAreaLUT) Performs a query that groups records by Name field and includes a record count in the result, query is shown below. The ModelAreaLUT is derived from the municipality table and so contains their names.

```
Select NAME, Count(*) from ModelAreaLUT group by NAME order by Col2 desc into
TT1
```

- If duplicates are found, then these names are saved to a table called xDupMunis_LUT1. Duplicates are determined by the Count field >1.
- Another table is created called xDupMunis_Problems with the same table structure as the ModelAreaLUT. This table will contain the duplicates that will cause problems with the model.

STEP 2 – The model now has the names of the duplicate municipalities, it now needs to check each one to determine if the corresponding regions are in the same model area or in different ones.

- A Fetch…Next loop is set up to evaluate each record in the xDupMunis_LUT1 table.
  - The municipality name for the target record is retrieved and stored in vMuniName
  - (ModelAreaLUT) Selects all records with the same name as the target. To make this selection a text string is built with the name of the target municipality. This text string is then run like a line of code. This is the only way in MapBasic to include a variable in a section that changes for each iteration of the loop.
  - Results of the previous selection is saved to a table call TempTable
  - (TempTable) A query is performed using a GroupBy on the ModelAreaName field. This creates a list of the different model areas for each of the target municipality regions.
  - If only one record is found in the result, then all of the target municipality regions are in the same model area and therefore will not cause a problem with the model. The TempTable is closed and the model moves on.
  - If more than one record is found in the result, then the target municipality regions fall in more than model area and so do the corresponding roads. These municipality regions are then added to the xDupMunis_Problem table.
- Then next record in the xDupMunis_LUT1 table is selected and processed using the steps described above until all records have been evaluated.

STEP 3 – Performs a check to see if there were any problem municipalities, and if there are, prepares for further processing.

- (xDupMunis_Problem) Determines how many records are in the table. If there are none then there is no need for further processing and the procedure is exited.
- (xDupMunis_Problem) If one or more records are found in then steps are performed to create an identical copy of the table, sorted from the smallest region to the largest. In the next step this table is used as a lookup table to control which regions are processed first.
- (xDupMunis_Problem) Four additional fields are created, one for each coordinate of the minimum bounding rectangle (MBR) for the region of each record. These fields are then updated with the appropriate coordinate value.
- (TempTable) This table is opened and then deleted as it is no longer needed and will just cause confusion later if left hanging around the working folder.

STEP 4 – For each record in the xDupMunis_Problem the corresponding roads are selected and the ModelAreaName field is updated with the correct name.

- A Fetch...Next loop is set up to evaluate each record in the xDupMunis_Problem table.
    - For the target record, six different values are retrieved and stored into variables, they are the municipality name (NAME), model area name (ModelAreaName), the coordinates for the MBR (MinX, MinY, MaxX, MaxY).
    - (ProvRoads) A query is performed to select all roads in the target municipality. These records are stored in a table called TT_RoadList.
    - (TT_RoadList) Another query is performed to select all roads located within the bounds of the MBR of the target record. Because the query includes variables that change for each iteration of the loop, the query statement is built using a series of text strings which is executed using a Run Command as shown below.

```
cmdSelect = "Select * from TT_RoadList where CentroidX(obj)>= " + vMinX +
                                        " And CentroidX(obj)<= " + vMaxX +
                                        " And CentroidY(obj)>= " + vMinY +
                                        " And CentroidY(obj)<= " + vMaxY +
                                        " into TT1"
Run Command cmdSelect
```

    - (TT1) For all records retrieved by the query, the ModelAreaName field is updated with the proper model area name.
- Then next record in the xDupMunis_Problem table is selected and processed using the steps described above until all records have been evaluated.

Prep_PopData2

The main purpose of this procedure is to simply select those records assigned to the target province. This data will later be added to the 6-digit postal code data before being divided into the various model areas for the target province.

- Opens the Configuration file and retrieves the name of the Population data layer. Detects if the file is new or old. If the file is new, then the following steps are performed. If the file is old, then all steps are skipped
- Opens the national coverage of the Population data set
- Queries those records assigned to the target province and saves the result in the provincial working folder with the name PopTotal.TAB

Prep_PostalPoints

- Opens the Configuration file and retrieves the name of the Postal Points data layer. Detects if the file is new or old. If the file is new, then the following steps are performed. If the file is old, then all steps are skipped
- Opens the national coverage of the Postal Points data set
- Creates a new table consisting of points located in the target province by performing the following query.

```
Select POSTALCODE, MEP_ID, SLI, PROV, COMM_NAME, MUNICIPAL, LONGITUDE, LATITUDE
from PostalPoints where SLI=1 and PROV=ProvAbb(i) into TT1
```

- (TT1) The resulting data is saved to the provincial working folder as PostalPointsAll.TAB
- (PostalPointsAll) The ColNum1 variable is populated with the column reference of the POSTALCODE field.
- (PopTotal) The table for the target province is opened and the ColNum2 variable is populated with the column reference for the POSTCODE field.
- (PostalPointsAll) A new field is added, `PopTot Float`
- (PostalPointsAll, PopTotal) The PopTot field in the postal points table is updated with the population value contained in the PopTotal table. The tables are linked using the column reference for postal code fields, specifically ColNum1 and ColNum2

```
Add Column "PostalPointsAll" (PopTot )From PopData Set To PP_TOT
Where ColNum1 = ColNum2
```

The postal points are now divided into the separate model areas for the target province. The following steps are performed for each model area present in the ModelAreaTable

- The model area table for the target province is opened
- (ModelAreaTable) The first record is selected and the name of the model area is read into a variable called AreaName
- (PostalPointsAll, ModelAreaTable) A selection is performed for all postal codes found inside the model area boundary where the point and the boundary have the same model area name. The query is shown below.

```
Select * from PostalPointsAll, ModelAreaTable where PostalPointsAll.Obj Within
ModelAreaTable.Obj And ModelAreaTable.ModelAreaName= AreaName into TT1
```

- (TT1) Results are saved to the working folder for the target model area.
- (ModelAreaTable) The model name for the next record is retrieved and the query is performed on the next record in the table.

Prep_PPN

This procedure divides the populated place name (PPN) and Voronoi data layers into the separate model areas for the target province. These two data sets are custom built layers that are stored in the \UB_Model\1_Program\SupportFiles\PPNVoronoiFiles folder.

- The PPN table for the target province is opened
- The Voronoi table for the target province is opened
- The model area table for the target province is opened
- (ModelAreaTable) The first record is selected and the name of the model area is read into a variable called AreaName.
- (PPN, ModelAreaTable) A selection is performed for all PPN records found inside the model area boundary. The query is shown below.

```
Select * from PPN, ModelAreaTable where PPN.Obj Within ModelAreaTable.Obj And
ModelAreaTable.ModelAreaName= AreaName into TT1
```

- (TT1) Results are saved to the working folder for the target model area.

- (Voronoi, ModelAreaTable) A selection is performed for all Voronoi records found to intersect the model area boundary and have been assigned to the target model area. The query is shown below.

```
Select Voronoi.PPname, Voronoi.Pop2 from Voronoi, ModelAreaTable where
Voronoi.Obj Intersects ModelAreaTable.Obj and ModelAreaTable.ModelAreaName =
AreaName into TT1
```

- (TT1) Results are saved to the working folder for the target model area.
- (ModelAreaTable) The model area name for the next record is retrieved and the queries are performed again with this name.

Prep_GNWpoints

The purpose of this procedure is to divide the Genworth property point data set into their respective model areas. Property points are used to help identify a fourth class of urban boundaries which do not meet the criteria for the other three classes but contain many property points which makes these areas significant to the business.

- Opens the Configuration file and retrieves the name of the Genworth property point data layer.
- Opens the property point table as Points
- (Points) Selects all records in the target province and saves the result to the provincial working folder as GNWpoints.TAB
- (GNWpoints) Adds a new field called ModelAreaName
- (GNWpoints) Creates an index on Prov field. This will speed up any queries on this field.
- Opens the ModelAreaLUT for the target province
- (GNWpoints) updates the ModelAreaName field with the name of the model area in which it is located.

```
Add Column "GNWpoints" (ModelAreaName )From ModelAreaLUT Set To ModelAreaName
Where contains
```

The point table is now divided into the model areas for the target province using the ModelAreaLUT aas a lookup table.

- A Fetch...Next loop is set up to evaluate each record in the ModelAreaLUT table.
    - The name of the model area is retrieved from the first/next record
    - (GNWpoints) A query is perform to select all records in the target model area. The table is saved to the working folder for the model area.

- Then next record in the ModelAreaLUT table is selected and processed using the steps described above until all records have been evaluated.

RoadDensityPoints

The main purpose of this procedure is to create a point file representing each road that is to be included in the road density calculation. This is achieved by processing data layers created by the Prep_Roads procedure for both the short roads (<1km) and long roads (>1km). Processing is divided into four steps which are described below.

Two external functions were built specifically for this model which are used in the first three processing steps. The first is a point aggregation function which reduces the number of points in a point file by inserting a single point to represent all points within a specified aggregation distance. The second function plots points along a line or polyline at a predefined distance. Both functions are contained within the "sogGenworthApp.dll" located in the "\UB_Model\1_Program" folder.

Step 1 – Aggregates points contained in the ShortRoadPts data set at 100 metres. This data set was created by the Prep_Roads procedure when the roads were divided into the model areas. This data layer is a point file representing the centroid of all roads that have a length less than 1km. Aggregation is performed to minimize the number of points so that future processing steps can run more efficiently. The number of points aggregated at each location is calculated and stored in the result file and used later when calculating the road density. Results are save to a file called ShortRoads_Agg100.TAB.

Step 2 – Points are plotted every 500 metres along the line segments found in the LongRoads data set. This data layer was created when the road data layer was divided into the model areas and contains all roads that have a length greater than 1km. Results are saved to a file called LongRoadPts.TAB

Step 3 – Aggregates the LongRoadPts at a distance of 500 metres. This is necessary because the function automatically creates a point at the beginning and end of each line segment. In locations where two or more roads come together point stacking will occur which artificially increases the road density. Aggregation will reduce the number of points down to one. Results are saved to a file called LongRoads_Agg500.TAB

Step 4 – Combines the ShortRoads_Agg100 and the LongRoads_Agg500 data layers into a single data layer.

- Opens the ShortRoads_Agg100 data layer as ShortRoadPts. Changes the name of the "Sum" field to "AggVal_Sum"
- Opens the LongRoads_Agg500 data layer as LongRoadPts. Changes the name of the "Sum" field to "AggVal_Sum". This field is updated with a value of 1
- The LongRoadPts data layer is appended to ShortRoadPts. Each table consist of a single column therefore only one column reference is needed.

```
Insert Into ShortRoadPts (COL1) Select COL1 From LongRoadPts
```

- ShortRoadPts is saved to a new file called RoadDensityPoints
- ShortRoadPts is closed without saving. This preserves its original state before the LongRoadPts file was appended.

RoadDensityBuffers

The main purpose of this procedure is to generate the urban boundaries using a buffering methodology on the output file from the RoadDensityPoints procedure. In addition to generating the boundaries, other operations are performed such as adding community names and population amounts.

- Opens configuration file and retrieves three threshold values
- Retrieves the value in the Var field where ID=”RoadCnt”. This value is stored in the vRoadCount variable. The value specifies the number of roads needed inside of a 1kmsq area before it is included in future processes.
- Retrieves the value in the Var field where ID=”RoadCnt2”. This value is stored in the vRoadCount2 variable. This is a threshold value used for creating supplementary urban boundaries.
- Retrieves the value in the Var field where ID=”BuffCnt1”. This value is stored in the vBuffCount variable. The value represents the number of buffers needed to generate an urban area.

- Initializes two variables with the value of zero (vX1present, and vX2present). These variables are used in the process of deciding whether a boundary will be given the “supplementary’ classification. The “pool” of boundaries that gets evaluated for this classification has two possible sources. The variables mentioned above notify the model that one or both sources are available.

- Opens the RoadDensityPoints table for the target model area
- Creates an empty table called “RoadBuffers” that will be used to store the road density buffers created. File is saved to the model area folder and in a projection system that uses metres as its distance unit.
- (RoadDensityPoints) Creates a circular buffer around each point object with an area of approximately 1 sqkm.
- (RoadBuffers, RoadDensityPoints) The “RoadCount” field for each buffer is updated with the sum of all RoadDensityPoints that fall within each boundary. Effectively this is the number of roads within 1sqkm.

- (RoadBuffers) Buffers that have a “RoundCount” value greater and equal the User defined threshold value (vRoadCount) are selected and saved to a separate file called “UrbanBoundaries”. A copy of the same data layer is saved to file called “BuffsXXplus” which is used in a later process step.
- Opens the UrbanAreas table with the alias name “TargetFeatures” This is necessary because the BuildBoundaries procedure requires the input file have that name.

- Calls the BuildBoundaries procedure. This procedure completes the build process and append model area names and population amounts among other tasks.

Updates each urban area (TargetFeatures) with the number of buffers that were needed to create them, then proceeds to remove those which are below the threshold

- Opens the BuffsXXplus table
- (TargetFeatures, BuffsXXplus) Updates the BuffCount field with the number of BuffsXXplus objects that falls inside each boundary
- (TargetFeatures) All records where BuffCount <= vBuffCount are selected.
- If there are records selected then the following operations are performed.
    - Selected records are first saved to a table called SuppBoundaries_x1.Tab and then deleted from the TargetFeatures table.
    - The TargetFeatures table is then saved and packed
    - The SuppBoundaries_x1 table is opened.
    - (SuppBoundaries_x1) A new field called SupFlag is added to the table and is then updated with a value of 1.
    - The vX1present variable is set to 1. This lets the model know that supplementary boundaries are available from this source and for this model area.

- The TargetFeatures table is closed. This is the alias for the UrbanAreas table. Alias was used because the BuildBoundaries procedure requires the input file having this name,
- The PSTExistFlag is set to 1 indicating that PST boundaries were created for this model area

The process of building Supplimentary urban boundaries begins.

- Opens the UrbanAreas table. This will be used later in the procedure

- (RoadBuffers) Selects records using the following expression. This selects those buffers with a threshold slightly below that which was used to generate the PST urban boundaries. The variablse cRoadCount and vRoadCount2 were retrieved from the configuration file at the beginning of this procedure.

```
Select * from RoadBuffers where RoadCount<vRoadCount and
RoadCount>=vRoadCount2 into TT1
```

- If one or more records were selected in the previous selection then the following operations are performed.
    - (TT1) Saved to a file called SuppBoundaries_x2
    - Opens SuppBoundaries_x2 with the alias name TargetFeatures
    - (TargetFeatures, UrbanAreas) Selects all objects that intersect with those boundaries in the UrbanAreas table. The result is stored in a table called TT1.
    - (TT1) The Carto12Count field is updated with a value of 1. This field is no longer needed for supplementary boundaries and so it is being repurposed to flag those records which were included in the previous selection.
    - (TT1) All records with a value of 1 in the Carto12Field are selected and deleted

Flagging records to be deleted in this case is necessary because these records were selected using a geographical join between two tables. The delete operation is not available when this occurs.

- (TargetFeatures) table is saved and packed.
- Calls the BuildBoundaries procedure. This procedure completes the build process for the supplementary boundaries.

- (TargetFeatures) A new field is added to the table called SupFlag (Float). This flag field indicates that the boundary was created as part of the PST build process. Since this is not what is being performed this field should contain all zeros. This field is added so that it has the same table structure as the one created in the PST build process. These tables will be later merged.
- The vX2present variable is set to 1. This lets the model know that supplementary boundaries are available from this source and for this model area.

Evaluates the presents of the SuppBoundaries_x1 and ...x2 tables. If present, then they are combined where appropriate. These files are determined to be present if the vX1 present and vX2present variables have been set to a value of 1.

- If the vX1present and vX2present variables are both set to 1 then the following operations are performed.
  - The tables SuppBoundaries_x1 and ...x2 are opened as SuppBoundx1 and SuppBoundx2
  - The SuppBoundx2 table is appended to SuppBoundx1
  - The Row_ID field is updated with the rowid, the table is then saved
  - A check is performed to see if there are multiple boundaries with the same name. The following query is run. It performs a GroupBy on the CommName field and counts the number of occurrences. It also determines the Max value of the row IDs.

```
Select CommName, Count(*) "Cnt", Max(Row_ID)"MaxValue" from SuppBoundx1
group by CommName into Lut1
```

  - (Lut1) All records are select where the Cnt field has a value greater than 1
  - (Lut2) If any records are selected then the following operations are performed.
    - The first record from the Lut2 table is selected
    - The community name and MaxValue are read into variables vCommName and vRowID respectfully
    - (SuppBoundx1) Selects all records where the community name is the same as the one selected from the look-up table Lut2.
    - (TT1) All

- If the vX1present variable is to 0 and vX2present is set to 1 then the following operations are performed. This means there were no boundaries left over from the PST analysis but there was from the supplementary boundary analysis.
    - The SuppBoundaries_x2 is opened with the alias SuppBoundx2
    - The GNWpoints table is opened
    - (SuppBoundx2) The GNWcount field is updated with the number of property points found within each boundary.

```
Add Column "SuppBoundx2" (GNWcount )From GNWpoints Set To Count(*) Where
within
```

BuildBoundaries

- Opens the Roads_Carto1to2 table with the alias name “Carto12”.
- (TargetFeatures) All buffers are combined into a single complex object (multiple polygons). This object is then disaggregated so that each region is now a separate object.
- (TargetFeatures) Several fields are added to the table as described below.

    - `Row_ID Integer`
    - `CommName Char(25)`
    - `PPNCount Integer`
    - `Pop1 Integer`
    - `Pop2 Integer`
    - `TotalPop Integer`
    - `ObjCount Integer`
    - `BuffCount Integer`
    - `ModAreaName Char(40)`

- (TargetFeatures) Updates the Row_ID field with the numeric row number for each record. This provides a unique ID to be used later.
- (TargetFeatures) Updates the ObjCount field with the number of polygons are used to create each urban area. Although complex objects have already been removed in an earlier step, boundaries that contain a hole will remain. This step identifies those with a hole.
- (TargetFeatures, RoadDensityPoints) Updates the RoadCount field for each urban boundary with the sum of the “AggVal_Sum” field of those point that are located within the area.
- (TargetFeatures, Carto12) Updates the Carto12Count field with the number of roads contained within each boundary

Removes holes from for any of the urban areas using the following steps

- (TargetFeatures) Selects all records where the ObjCount field is >1
- If any are found, then the result is saved to a file called Deleteme1. If none are found, then all steps are skipped.
- The Deleteme1 table is opened as “ProcList”. For each record in this table the following steps are performed.
- A Fetch...Next loop is set up where each record in the ProcList table is evaluated
    - The value in the Row_ID field is read into a variable called “ObjectID”
    - (TargetFeatures) Selects the specific object to disaggregated by selecting record where the Row_ID = ObjectID
    - (TargetFeatures) Disaggregates the selected object in such a way that that all components retain the same table information as the original object
    - (TargetFeatures) Selects all records where Row_ID = ObjectID and sorts them from by area from largest to smallest. Results are stores in a temporary table called TT2. This selection will include the all components of the original object before it was disaggregated.

    - (TT2) All but the first record is selected and deleted
- The program goes to the next record in the ProcList until all have been processed.
- (TargetFeatures) Table is saved and packed.

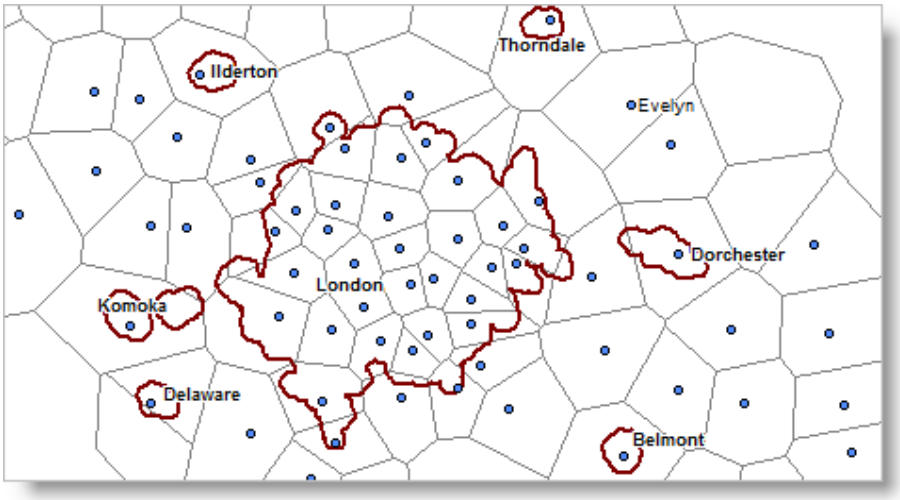
Adds Model Area name to boundaries

- (TargetFeatures) Updates the ModelAreaName field with the name of the target model area. This name is read into the "AreaName" variable earlier in the program.
- Opens the Voronoi data layer for the target model area.

Updates each urban area with the community name

- Open the PlaceNamePoints table as PPN. This is a customized point table representing the location of a community.
- (TargetFeatures, PPN) Updates the PPNcount field with the number of point objects from the PPN table that are located within each urban boundary.
- (TargetFeatures) A selection is made to identify any boundaries where the PPNcount <=1. Results are stored in a temporary table called TT1.
- (TT1, Voronoi) Updates the CommName field with the community name. The name is retrieved from the Voronoi table in which the TT1 object is located. Code is displayed below. This addresses most all rural communities which produce a single urban boundary for each community point.

```
Add Column "TT1" (CommName) From Voronoi Set To PPname Where Contains
```



Blue points represent community locations in and around London, ON. Communities like Thorndale, Dorchester, and Belmont, the name is retrieved from the Voronoi boundary layer since these urban boundaries contain only a single community point. The London urban boundary contains several community points and so a different strategy is needed which is described below.

In medium or large metropolitan areas, several community points are usually found within the urban boundary. Because of this and the complex geometries existing between this boundary and the multiple Voronoi polygons, a different approach is needed. For larger urban

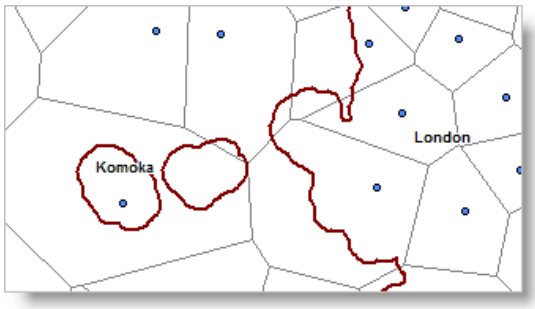
boundaries the name is retrieved from the community point within the boundary which has been flagged as the dominant community name. The following steps describe this process.

- (TargetFeatures) Select all records where PPNcount >=2. Saves the result to a temporary table called TT3. This is used as a lookup table.
- (TT3) determines if there were records selected, if yes then the program proceeds
- A Fetch...Next loop is set up to evaluate each record in TT3
    - (TT3) Determines the row id for the first record in the table
    - (TT3) Saves a copy of the table as “Deleteme”. The next step is an SQL query which cannot be performed with a temporary table therefore a base table is created called Deleteme. This file will be deleted later when it is no longer needed.
    - (PPN, Deleteme) Selects all of the community points that fall inside of the urban boundary, sorts the results in descending order by the dominant community flag (DomCommFlag) field. This ensures that community point you want to retain for the urban boundary name is positioned at the top of the TT4 table. The following is the query used.

      ```
      Select PPN.NAME, PPN.DomCommFlag from PPN, Deleteme
      where PPN.obj Within Deleteme.obj and Deleteme.Row_ID=RowNum
      order by PPN.DomCommFlag desc into TT4
      ```

    - (TT4) The first record is selected and the community name is retrieved and stored in a variable called “MasterName”
    - (TT3) Selects the specific row to update with the community name by using the row id that was retrieved earlier. Updates the CommName field with the contents of the MasterName variable.
    - (Deleteme) this table is deleted.
- The program goes to the next record in the TT3 table until all have been processed.
- (TargetFeatures) Table is saved and packed.

The next step is to check for multiple boundaries with the same community name. This is a rare occurrence but does happen when there are two small subdivisions that are over 2km apart. At this distance they typically fall within the same Voronoi region which explains why they receive the same community name. A good example of this is Komoka just on the outskirts of London, ON. When this happens, the polygons are combined together.



- (TargetFeatures) A query is performed that groups records by the community name and sorts them in descending order by the count field (Col2). Results are stored in a temporary table called TT1. The query is shown below.

```
Select CommName, Count(*) from TargetFeatures group by CommName order by col2
desc into TT1
```

- (TT1) determines if any records were selected by the previous query, if yes then the program proceeds.
- A Fetch...Next loop is set up to evaluate each record in TT2
    - (TT2) The first record is selected, and the community name found in the CommName field is retrieved and stored in a variable called “PlaceName”
    - (TargetFeatures) A selection is made for all records that have the same community name as the “PlaceName” variable. Results are stored in TT3
    - (TT3) All objects in the selection are combined together, see code below. Text fields retain the same information they did previously. The sum was calculated for numeric fields.

```
Objects Combine Data
RoadCount=sum(RoadCount),CommName=CommName,TotalPop=TotalPop,
BuffCount=sum(BuffCount)
```

- The program goes to the next record in the TT2 table until all have been processed.
- (TargetFeatures) Table is saved and packed.

Check for boundaries without a name is performed

- (TargetFeatures) A query is performed to select records with a blank in the CommName field. Results are stored in TT4
- (TT4) If there are any records selected then the TT4.CommName field is updated with the name of the Voronoi boundary the TT4 object is within.

The population for each community is estimated and appended to the corresponding boundary. This is a four-step process that requires the use of the Postal Points data set.

Step 1 – Data preparation

- Opens the PostalPoints table for the target model area
- (PostalPoints) Adds two fields to the table: `PPname Char(40), PopUpdateFlag Integer`
- (Voronoi) Adds two fields to the table: `SqArea Float, PctOverlap Integer`
- (PostalPoints) Updates the PPname field with the name of the Voronoi boundary in which it is located.
- (Voronoi) Updates the SqArea field with the square area of the Voronoi region, in square kilometres

Step 2 – Flags postal points that are inside of an urban boundary and updates the boundaries with population amount

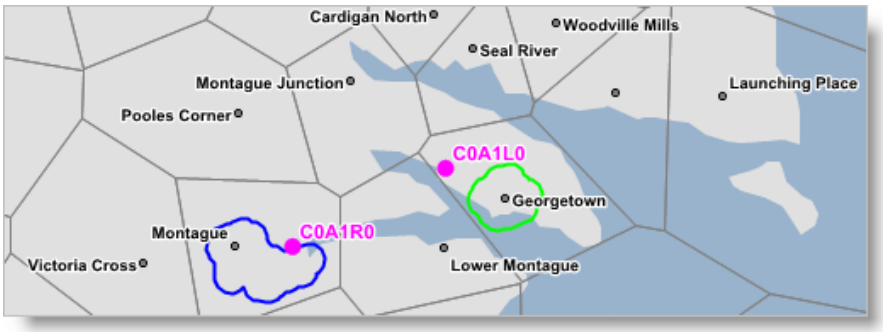
- (PostalPoints, TargetFeatures) Selects postal points that are inside of an urban boundary

```
Select PostalPoints.PPname, PostalPoints.PopTot, PostalPoints.PopUpdateFlag
from PostalPoints, CommAreasTable
where PostalPoints.obj Within CommAreasTable.obj into InBoundaryPoints
```

- (InBoundaryPoints) Saves this table to the working model area folder, by the same name. Used for future reference.
- (InBoundaryPoints) Updates the PopUpdateFlag to a value of 1. This indicates to the model that the populations associated with these points have already been used. Note, the table updated is the temporary table created from the above query, not the one that was just saved.
- (TargetFeatures, InBoundaryPoints) Updates the Pop1 field with the sum of population value of those InBoundaryPoints located within the urban area.

```
Add Column "CommAreasTable" (Pop1 )From InBoundaryPoints Set To sum(PopTot)
Where within
```

Step 3 – Identifies postal points that are outside of an urban boundary but inside of the Voronoi region that has the same name as the urban boundary. If any are found, then the population amounts are summed and applied to the urban boundary.



Postal codes C0A 1R0 and C0A 1L0 are located outside of their respective urban areas but are contained within the same Voronoi region as the community. In this situation the populations are captured and applied to the urban boundary.

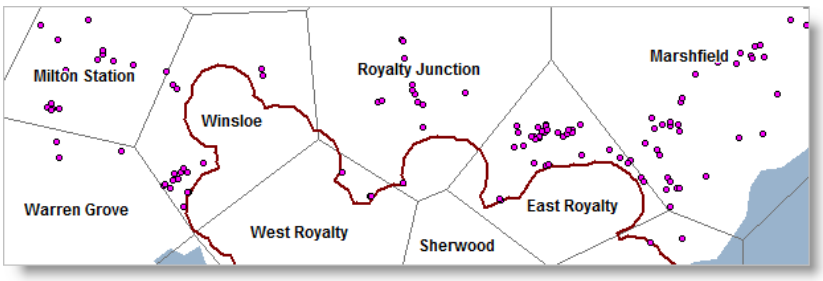
- (PostalPoints) Selects records where the PopUpdateFlag=0. Results are stored in a temporary table called OutBoundaryPoints.
- (OutBoundaryPoints) Saves this table to the working model area folder, by the same name. Used for future reference.

- (TargetFeatures) For each records in the TargetFeatures table the following procedures are performed.
    - First record is selected. The Community name is retrieved and stored in a variable called UBname. The record ID is Retrieved and store in a variable called TargetRec
    - For the target urban boundary all OutBoundaryPoints are selected which have the same community name as the target and has not been previously used.

```
Select * from OutBoundaryPoints where PPname = UBname and PopUpdateFlag =0
into TT1
```

    - If there are any records selected, then the following procedures are performed.
    - (TT1) The PopUpdateFlag is updated to a value of 1
    - (TT1) The sum of the PopTot field is determined and stored in a temporary, TT2
    - (TT2) The result of the previous query is retrieved and stored in a variable called PopValue.
    - (TargetFeatures) Selects the TargetFeatures with the specific record ID equal to the TargetRec variable. Results are store in TT3
    - (TT3) Updates the Pop2 field with PopValue
- The program goes to the next record in the TargetFeatures table until all have been processed.

Step 4 – In this step the postal points that are outside of an urban area are analyzed and those which are found to be in reasonable proximity to an urban area have their population values added to the population of the urban area. Proximity is defined by the amount of overlap the urban area has with respect to the Voronoi regions. If there is 25% overlap then the population of the postal points is added to the urban area. In the Charlottetown example below, the population from points contained within the Winsloe and East Royalty Voronoi regions are added to the urban area because the amount of overlap is 42% and 51% respectively. The population from points for Royality Junction are not added because the percent overlap is only 9%, well below the threshold of 25%



- Sets area units for MapInfo as “sq km”, this ensures all area calculations are done using this same unit of measure, specifically the AreaOverlap() function
- (Voronoi, OutBoundaryPoints) Updates the Voronoi.Pop2 with the sum of the population for all OutBoundaryPoints record that is located within the Voronoi region.

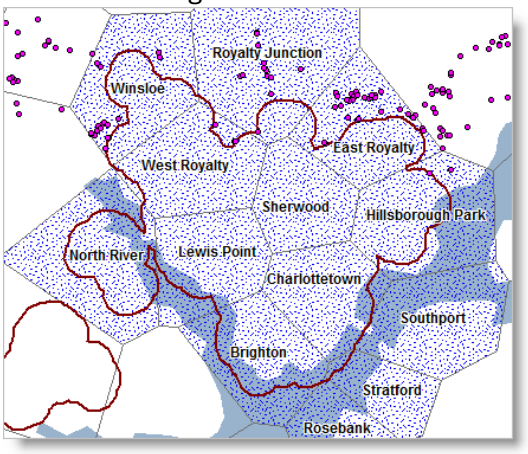
- A Fetch...Next loop is set up and each record in the TargetFeatures table the following procedures are performed.
    - First record is selected. The community name is retrieved and stored in a variable called UBname. The record ID is Retrieved and store in a variable called TargetRec
    - Creates a list of Voronoi regions that intersects (overlaps) the target urban area. Results are saved as Vor_LUT.

```
Select Voronoi.PPname, AreaOverlap(Voronoi.Obj,CommAreasTable.Obj)
"OverlapArea", Voronoi.SqArea "TotalArea", Voronoi.PctOverlap
from CommAreasTable, Voronoi
where CommAreasTable.Obj Intersects Voronoi.Obj
and CommAreasTable.CommName = UBname
into Vor_LUT
```

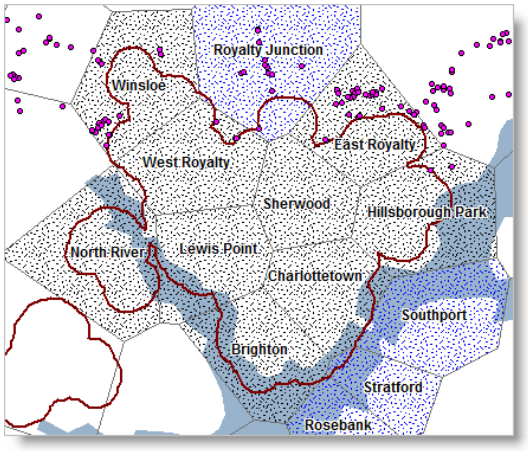
    - The results are saved as Vor_LUT to the model area working folder. The temporary table of these results is closed and the saved version of the same name is opened.
    - (Vor_LUT) The percentage of overlap is calculated using the OverlapArea value calculated in the previous query and the area of the Voronoi region. Results are stored in the PCTOverlap field.

PPname	OverlapArea	TotalArea	PctOverlap
Brighton	4.78363	7.35315	65
Charlottetown	5.86442	6.19288	95
East Royalty	3.64006	7.07268	51
Hillsborough Park		9.0097	48

    - (Vor_Lut) All Voronoi regions where the percent overlap is greater than 25% are selected and stored in a temporary table called Vor_Lut2. A copy of this file is saved to the model area working folder.
    - If no records are selected, then the program goes to the next record in the TargetFeatures table



Map showing the 13 Voronoi regions (blue) that overlap the Charlottetown urban area and the



After determining the percent of overlap between the Voronoi region and the Charlottetown urban area, 9 surpassed the threshold of 25% (gray). If

postal code points that have not yet been used in a population count (OutBoundaryPoints)

any postal points are found in these regions, then their population amounts are added to the Charlottetown urban area. Only regions labelled Winsloe and East Royalty contain postal points

- If no records are selected, then the program goes to the next record in the UrbanAreas table
- Each Voronoi region in the Vor_LUT2 table is queried to determine if there are any postal points contained within them. The following steps are performed.
  - (Vor_LUT2) The first record is selected. The Voronoi region name is retrieved and stored in a variable called VorName
  - (OutBoundaryPoints) All records that have a PPname = VorName

    ```
    Select * from OutBoundaryPoints where PPname = VorName and
    PopUpdateFlag=0 into TT1
    ```

  - If TT1 contains any records, then the PopUpdateFlag is updated with a value of 1
  - (TT1) The sum of PopTot field is calculated and stored in a table called TT2
  - The population summed value is added to a variable called PopSumValue
  - The next record is selected in Vor_LUT2 table and the steps described above are repeated until all records have been evaluated.
- The program goes to the next record in the Vor_LUT2 table until all have been processed.

- (TargetFeatures) The target urban area is selected using the row id which was retrieved earlier and stored in the TargetRec variable. Results are stored in a temporary table TT3
- (TT3) The value contained in the PopSumValue variable is added to the existing value of the Pop2 field.

  ```
  Update TT3 Set Pop2 = Pop2 + PopSumValue
  ```

- The PopSumValue variable is set to zero prior to starting the next urban area
- The Next record in the UrbanAreas table and the steps described above are repeated until all records have been processed.

- The program goes to the next record in the TargetFeatures table until all have been processed.

- The total population for each urban area is then calculated as the summation of the Pop1 and Pop2 fields. These represent the populations of postal points found inside the boundary and outside.

CombinePSTareas

This procedure is run after each model area in the target province has been processed. Its purpose is to collect the results from each model area and save it to a provincial coverage of the same data.

- Determines if the variable called LoopToggle1 is equal to zero. If it equals zero, then the model knows this is the first time the CombinePSTareas procedure was called. If it is greater than zero, then it knows it has been called more than once.

- If LoopToggle1 = 0 then the following steps are performed.
  - Opens the UrbanAreas file for the target model area and saves the table to the provincial working folder with the name "UrbanAreas_*PROV*.TAB"
  - LoopToggle1 is set to a value of 1

- I If LoopToggle1 > 0 then the following steps are performed.
  - Opens the provincial version of the file created earlier, "UrbanAreas_*PROV*.TAB". An alias of "ProvUrbAreas" is used to reference this data set.
  - Opens the UrbanAreas file for the target model area. An alias of "TT1" is used to reference this data set.
  - The data in the TT1 file is appended to ProvUrbAreas table using the operation shown below

```
Insert Into ProvUrbAreas ( COL1, COL2, COL3, COL4, COL5, COL6, COL7, COL8,
Col9, Col10, Col11) Select COL1, COL2, COL3, COL4, COL5, COL6, COL7, COL8,
Col9, Col10, Col11 From TT1
```

- The UrbanAreas table is saved

CombineSuppAreas

This procedure behaves in a similar fashion to the CombineUrbAreas procedure; however, its purpose is to create a provincial coverage layer of urban areas that did not meet threshold values for any of the three (PST) classes. These boundaries will be later evaluated to determine if they meet the criteria for becoming supplementary areas, areas below thresholds but contain many Genworth insured properties.

- Determines if the variable called LoopToggle2 is equal to zero. If it equals zero, then the model knows this is the first time the CombineDeletedAreas procedure was run. If it is greater than zero, then it knows it has been called more than once.

- If LoopToggle2 = 0 then the following steps are performed.
    - Determines if the supplementary boundaries for the target model area were created using the FileExist() function. Note, these boundaries are not always created.
        - If the boundary file exists, then the supplementary boundary file is opened and a copy of this file is saved to provincial working folder with the name "SuppBoundsALL_*PROV*.TAB"
        - LoopToggle2 is set to a value of 1.
    - If a supplementary boundary file does not exist, the model simply moves to the next model area

- I If LoopToggle2 > 0 then the following steps are performed.
    - Determines if the supplementary boundaries for the target model area were created using the FileExist() function. If the file exists, then the following then the following operations are performed.
        - Opens the supplementary table for the target model area with the name TT5
        - Opens the SuppBoundsALL table with the name ProvSuppAreas
        - Appends the TT5 table to the ProvSuppAreas table

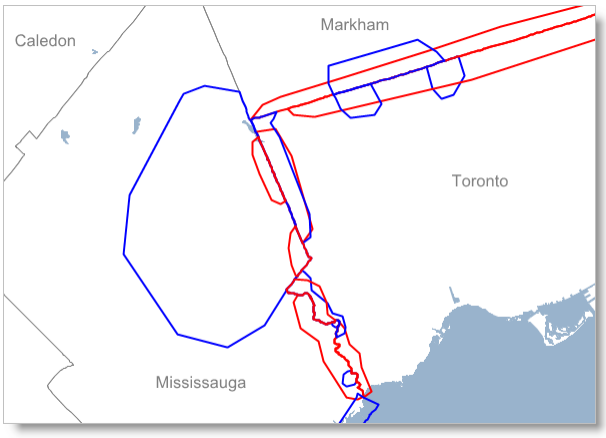
```
Insert Into ProvSuppAreas ( COL1, COL2, COL3, COL4, COL5, COL6, COL7, COL8,
Col9, Col10, Col11, Col12) Select COL1, COL2, COL3, COL4, COL5, COL6, COL7,
COL8, Col9, Col10, Col11, Col12 From TT5
```

        - The ProvSuppAreas table is saved

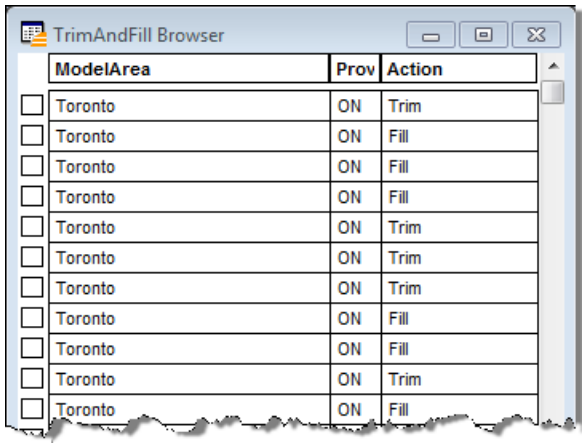
    - If a supplementary boundary file does not exist, the model simply moves to the next model area

TrimAndFill

This basic function of this procedure is to trim urban boundaries to the outer edge of the model area in which they are located and to fill-in any unwanted holes in the urban coverage. To help the model be more precise on deciding which areas are trimmed or filled, a support file is used to define which operation is performed at any given location. Although this procedure is performed after all urban boundaries for a province are generated, it performs its operation at the model area level. The following table provides a basic illustration of this support layer.



The map for the TrimAndFill table consist of a number of red a blue boundaries. The red areas indicate where urban boundaries will be trimmed, and the blue areas show where holes in the urban boundaries will be filled.



ModelArea	Prov	Action
Toronto	ON	Trim
Toronto	ON	Fill
Toronto	ON	Fill
Toronto	ON	Fill
Toronto	ON	Trim
Toronto	ON	Trim
Toronto	ON	Trim
Toronto	ON	Fill
Toronto	ON	Fill
Toronto	ON	Trim
Toronto	ON	Fill

View of the TrimAndFill table. This table specifies what type of operation should be applied to each model area.

- Opens the TrimAndFill table from the "\UB_Model\1_Program\SupportFiles" folder.
- (ResultTable) Adds a new field to the provincial urban boundary table called TempID. This table was already open with the alias name "ResultTable".

The Trim operation is executed first.

- (TrimAndFill) Creates a lookup table of all model areas in the province that require a Trim operation. This is done by selecting all records for the target province that have the word "Trim" in the Action field. Results are saved to a temporary table called Trim_LUT.

```
cmdSelect = "Select ModelArea from TrimAndFill where Prov="
             +Chr$(34)+ ProvAbb(i) +Chr$(34)+
             "and Action="+Chr$(34) +"Trim"+ Chr$(34)+ "
             group by ModelArea order by ModelArea into Trim_LUT"
 Run Command cmdSelect
```

Note: Because MapBasic does not permit a variable name be included within an SQL statement a workaround needs to be employed. A variable call "cmdSelect" is created and populated with a text string containing the SQL statement that includes the required variables. The cmdSelect variable is then executed using a Run Command statement. The code for this query is shown above. Please note that Chr$(34) is the ASCII character number for a double quote.

- (Trim_LUT) A check is performed to determine if there were any objects selected from the query. Remember this is a lookup table of model areas that require trimming. If objects are found then the model continues otherwise it moves the Fill operation.

- A Fetch…Next loop is set up so that the following operations are run for each model area listed in the Fill_LUT table.
    - The first record in the Trim_LUT is selected
    - (Trim_LUT) The name of the model area is retrieved and entered into a variable called “AreaName”

    - (ResultTable) Selects all urban boundaries from the ResultTable that have been assigned the same model area name as the one retrieved from the Trim_LUT (AreaName). Results are stored in a temporary table called ObjList.
    - (ObjList) Checks if there are any records returned by the selection. Value is stored in a variable called RowCount.
    - If RowCount is >0 then the model proceeds. The selected urban boundary is set as the Target. This is a MapInfo term identifying the selected object as the one to be edited, in this case it will be trimmed. Note, it is possible to have more than one urban boundary in a model area that could be selected.
    - (TrimAndFill) Selects all records in the TrimAndFill table that are to be used to trim the selected urban boundary object.

```
cmdSelect2 = "Select * from TrimAndFill where ModelArea="
         +Chr$(34)+  AreaName +Chr$(34)+ " and Action="
         +Chr$(34)+ "Trim" +Chr$(34)+ "into TrimWithObjects"
 Run Command cmdSelect2
```

    - The trim operation is performed in a way that all values in the table is preserved.

      Objects Erase Into Target
        Data RoadCount=RoadCount, Carto12Count=Carto12Count, Row_ID=Row_ID, CommName=CommName, PPNCount=PPNCount Pop1=Pop1, Pop2=Pop2, TotalPop=TotalPop, ObjCount=ObjCount, BuffCount=BuffCount, ModAreaName=ModAreaName, Prov=Prov, Class=Class, GNWcount=GNWcount

    - The Target is turned off in preparation for the next trim operation
- The next object in the Trim_LUT is selected and the operations described above are repeated until all records Trim_LUT have been processed.

The Fill operation is performed next

- (TrimAndFill) Creates a lookup table of all model area in the province that require a Fill operation. . This is done by selecting all records for the target province that have the word "Fill" in the Action field. Results are saved to a temporary table called Trim_LUT.

```
cmdSelect = "Select ModelArea from TrimAndFill where Prov="
             +Chr$(34)+ ProvAbb(i) +Chr$(34)+
             "and Action="+Chr$(34) +"Fill"+ Chr$(34)+ "
             group by ModelArea order by ModelArea into Fill_LUT"
 Run Command cmdSelect
```

- (Fill_LUT) A check is performed to determine if there were any objects selected from the query. If objects are found then a number of operations are performed. If none are found then these operations are skipped.

- A Fetch…Next loop is set up so that the following operations are run for each model area listed in the Fill_LUT table.
    - The first record in the Fill_LUT is selected
    - The name of the target model area is retrieved and entered into a variable called "AreaName"

    - (ResultTable) Selects all urban boundaries from the ResultTable that have been assigned the same model name as the one retrieved from Fill_LUT. The results are stored in a temporary table called ObjList.
    - (ObjList) Checks if there are any records returned by the selection. Value is stored in a variable called RowCount.
    - (ObjList) If there is only one object selected then the following operations are performed.
        - The urban boundary class for the object is retrieved and stored in a variable called vPSTClass
        - The PenColourPicker procedure is called. This sets the brush and pen styles for the specific class of the object.
        - The selected urban boundary is set as the Target

        - (TrimAndFill) All objects located in the target province and model area, and whose Action indicator is specified as "Fill", are selected. Results are stored in a temporary table called FillerObjects.
        - (FillerObjects) Performs a quick check if any objects were selected.
        - If a Fill object is found it is then Combined with Target urban boundary
        - The ChangeObjStyles procedure is called. This procedure contains two functions the first changes the brush style and the second changes the pen style of the boundary.

    Note: when a merge operation is performed in MapInfo the target object is deleted from the table and a new, modified version of the object is added. The new object is created with the default style settings, for polygon objects it is a solid white fill colour with a black border.

- (ObjList) If there are more than one object in this temporary table, then each object in the list is evaluated to determine which one(s), are subject to the Fill operation. To do this, a means of evaluating each object is performed.
- (ResultTable) Updates the TempID field with the rowid. This unique name will allow individual records to be referenced or selected.
- A variable "z" is initialized with a value of 1. This allows the program to refer to a particular row number
- A Fetch...Next loop is set up so that the following operations are performed for each object listed in the ObjList table.

    - (ObjList) The first record is selected using a query. Result is stored in a table called ObjectsToEdit. This is done because upcoming operations cannot be done on a records selected with a Fetch statement, it needs an actual table.

      ```
      Select * from ObjList where rowid = z into ObjectsToEdit
      ```

    - (ObjectsToEdit) The unique value in the TempID field is retrieved and stored in a variable called vTemID
    - (ObjectsToEdit) The urban boundary class for the object is retrieved and stored in a variable called vPSTClass
    - The PenColourPicker procedure is called. This sets the brush and pen styles for the specific class of the object.
    - The selected urban boundary is set as the Target

    - (TrimAndFill) Selects all objects in the model area whose Action is specified as "Fill" and also intersect with the target urban boundary (the same TempID). Saves results to a temporary table called FillObjects.
    - (FillObjects) Checks if there were any intersecting objects. If there were none then the model continues on to the next record in the ObjList table.

    - If objects were selected in the preceding query, then the objects are combined to the Target object
    - The ChangeObjStyle procedure is called
    - The loop counter "z" is incremented by one. This ensures the next record in the ObjList table is evaluated.
- (ObjList) The next record in the ObjList table is processed

- (Fill_LUT) The next model area in the Fill_LUT is processed.

PenColourPicker

This procedure is only called during the Fill operation contained within the TrimAndFill procedure. Its purpose is to ensure that the pen style used for each of the urban boundary types (primary, secondary, tertiary) is preserved. This is needed because the function in MapInfo the merges objects together tends to use its default styles which is entirely different from what the model uses.

- In the Fill operation, where this procedure is called, the urban boundary classification for the target urban area is determined and stored in a variable called vPSTclass. Depending on what is returned, the Brush and Pen styles are defined.

```
If vPSTclass="Primary" Then
    newBrush = MakeBrush(1,0,0)
    newPen = MakePen (2, 2, 8388608)

Elseif vPSTclass="Secondary" Then
    newBrush = MakeBrush(1,0,0)
    newPen = MakePen (2, 2, Blue)

Else    'vPSTclass="Tertiary" Then
    newBrush = MakeBrush(1,0,0)
    newPen = MakePen (2, 2, Green)
End If
```

- The description for the Brush and Pen functions are as follows.

  Brush (pattern, foreground colour, background colour)
  Pen (width, pattern, colour)

ChangeObjStyle

This procedure performs the style changes to those urban boundaries that have been modified by the Fill operation. This is necessary because the Merge function in MapInfo that the Fill operation uses, deletes the original object and creates a new one with the modifications. The new object does not retain the style of the original and so it needs to be restored manually.

- Initializes n=1. This controls the loop that will select records
- A For…Next loop is set up that will increment between n and RowCount. RowCount is the number of objects to edit determined in the Fill section of the TrimAndFill procedure
- (ObjectsToEdit) A selection is performed that selects the first record in the table

```
Select * from ObjectsToEdit where RowId = n
```

- A variable called objModify is populated with the actual object of the first record

```
objModify = selection.obj
```

- Initiates a Case statement based on the object type of the selected object

```
Do Case ObjectInfo(objModify, OBJ_INFO_TYPE)
```

- If the object style is a polygon then two functions are run. The first changes the outline of the polygon called AlterObjectPenStyle, and the second changes the fill pattern, called AlterObjectBrushStyle.
- Both of these functions are defined within the ChangeObjStyle procedure.

FinalCombine

This FinalCombine procedure combines the urban boundaries for all provinces included in the run, into a single output file. The name of the output file depends on how many provinces were included. If all provinces were included then the output filename will have "CAN" appended to the name, specifically "UrbanAreas_CAN.TAB". If only some of the provinces were included, then "PRV" is appended to the name. If only one province is included, then that province's abbreviation is used.

- Procedure begins by checking the value contained in the variable ArraySize. This is the number of provinces included in the model run which is determined when the program is first started.
- Next, it determines what suffix will be added to the output file name. If the array size is equal to 10, the maximum number of provinces that can be included, then the prefix "CAN is to be added. This is done by setting the variable "AddTxt" equal "CAN"
- If the array size is anything greater than 0 and less than 10, then the prefix will be "PRV" following by a numeric value set to the number of provinces included in the model run. This is done by setting the AddTxt variable is set to equal "PRV" + ArraySize.

- Initializes i=1. The "i" variable is used to identify individual provinces in the ProvAbb() array
- Initializes LoopToggle3=0. This is used to force the model to perform a particular set of operations on the first province it encounters while a different set of operations is performed on all other provinces. Basically, it makes a copy of the first result file and then appends the remaining result files to it.
- A For…Next loop is set up to once for each province included in the model run
    - LoopToggle3 is evaluated, if it equals zero then the following operations are performed.
    - Opens the result file for the target province. This is the first province listed in the ProvAbb() array.
    - This data set is saved to the Result folder with a new name "UrbanAreas_XXX.TAB" where XXX = AddTxt and is either CAN or PRV as explained earlier.

```
Commit Table TT1 as FP_Results + FolderName + "\UrbanAreas_" + AddTxt + ".Tab"
```

    - Note: The above code shows a variable called FolderName as part of the file path for the destination folder. This variable contains the actual name of the destination folder which is an eight-character date string that corresponds to when the model was started. This variable was defined by the CreateResultFolder procedure.
    - The value of LoopToggle3 is changed to a value of 1. This ensures this portion of code is <u>not</u> run again for other provinces.

    - If LoopToggle3 is greater than zero, then the following operations are performed.
    - Opens the output file that it has just created from the Result folder as TargetTable
    - Opens the result file for the next target province listed in the ProvAbb() array as TT1
    - The contents of the TT1 table are appended to the TargetTable

```
Insert Into TargetTable ( COL1, COL2, COL3, COL4, …COL14)
                    Select COL1, COL2, COL3, COL4, …COL14 From TT1
```

- The urban boundaries for all other province are appended until all elements contained in the ProvAbb() array have been processed.

FindOverlaps

The FindOverlaps procedure analyzes the urban boundary table compiled by the FinalCombine procedure to identify which boundaries overlap each other and by how much.

- Procedure begins by creating an output table to store the results of the analysis. The is table is called “OverlapCheck.Tab” and is stored in the results folder. The following outlines the structure of this table.
    - TargetName Char (25)
    - TargetUrbID Integer
    - TargetClass Char(15)
    - TargetModelArea Char(40)
    - IntersectName Char (25)
    - IntersectUrbID Integer
    - IntersectClass Char (15)
    - IntersectModelArea Char(40)
    - Prov Char(2)
    - TargetObjArea Float
    - OverlapArea Float
    - PCT Decimal (17,13)
    - PairCode Char(15)
    - ProcFlag Integer
    - Description Char(30)

- Next, the number of records in the “TargetTable” is determined. The name “TargetTable” is the alias used for the combined urban boundary table that was just created in the FinalCombine procedure.
- The Row_ID field is updated with the RowID. This is done to ensure there is a unique value assigned to each record. This field was used earlier for a different purpose and at this point in the model it is no longer holds a unique value
- The variable k is initialized at 1. This is a counter that is used to track the number of loops (records analyzed) performed in the next operation.

- A Loop is set up to evaluate each record in the TargetTable. The following describes what is performed for each record.
    - The first record is selected from the TargetTable (urban boundaries) and is saved as a separate table called “Deleteme.Tab”. This is done because an SQL query is performed next and temporary tables cannot be referenced query statement. This table is deleted after it is no longer needed.
    - The Deleteme table is opened and given the name “TargetObject.

- The following SQL query is performed. If the target object intersects another urban boundary object, then an 11 field table is created. This table contains information identifying the target object, the one it overlaps and how much area is overlapped.

```
Select TargetObject.CommName, TargetObject.Row_ID, TargetObject.Class,
       TargetObject.ModAreaName, TargetTable.CommName, TargetTable.Row_ID,
       TargetTable.Class, TargetTable.ModAreaName, TargetTable.Prov,
       Area(TargetObject.obj,"sq m"),
       Area(Overlap(TargetTable.Obj, TargetObject.Obj),"sq m")
from   TargetTable, TargetObject
where  TargetTable.Obj Intersects TargetObject.Obj
into   TT2
```

- If the Selection contains no records, then the model moves on to the next record in then TargetTable.
- If the Selection does contain records, which means an overlap was detected, then the results are inserted into the OverlapCheck table. The fields in this selection should correspond with the first 11 fields in the OverlapCheck table. The OverlapCheck table contains four additional fields which will be used later.
- The next record in the TargetTable is analyzed.

- Once all the urban areas have been analyzed, a check is performed to determine if there were any records inserted into the OverlapCheck table.
- If records are found, then the following operations are performed. Please note that the OverlapCheck table will contain several records that are of little value and are removed from the table. This occurs because the previous operation will add records to the OverlapCheck table in the following scenarios, When object A overlaps object B, when object B overlaps object A, when Object A overlaps itself, and when object B overlaps itself.

  - The PCT field in the OverlapCheck table is updated with the percent of overlapping area
  - The PairCode field is updated with a text string containing the RowID of the target object, followed by a comma, followed by the RowID of the intersecting object.
  - The OverlapCheck table is sorted alphabetically by the TargetClass field. This ensures Primary boundaries are listed first, Secondary boundaries next, and Tertiary boundaries last. In order to make this a permanent change, this query needs to be saved as a table, the current Overlap Check table is deleted, and the saved selection is opened and re-named.

- A selection is performed to identify all records in the OverlapCheck table that have a PCT = 0 or PCT>99. These records are removed from the table. A percent overlap of 0 indicates that the two boundaries are adjacent to each other, they share a common border. If the percent overlap is >99% then this indicates it is overlapping with itself. Both types of records provide no value and therefore removed.

TargetName	TargetI	TargetC	IntersectName	Interse	IntersectClass	Prov	PCT	PairCode	
Calgary	29	Primary	Calgary	29	Primary	AB	100.0000000000000	29,29	A
Calgary	29	Primary	De Winton	204	Secondary	AB	0.0003652246678	29,204	
Edmonton	312	Primary	Edmonton	312	Primary	AB	100.0000000000000	312,312	
Edmonton	312	Primary	Campbell	314	Primary	AB	0.0000000000000	312,314	B
Lloydminster	313	Primary	Lloydminster	313	Primary	AB	100.0000000000000	313,313	
Lloydminster	313	Primary	Lloydminster	2,406	Primary	SK	0.0000000000000	313,2406	
Campbell	314	Primary	Edmonton	312	Primary	AB	0.0000000000000	314,312	
Campbell	314	Primary	Campbell	314	Primary	AB	100.0000000000000	314,314	
Aldergrove	445	Primary	Aldergrove	316	Tertiary	BC	18.1818436889104	445,316	C

An example of the OverlapCheck table prior to modifications. Not all fields are displayed.

A. This record is a by-product of the methodology used to identify overlapping boundaries. It is simply saying that it overlaps itself as indicated by the percent of 100%, and the pair code containing two identical numbers. Since this does not provide useful information it will be removed from the table.

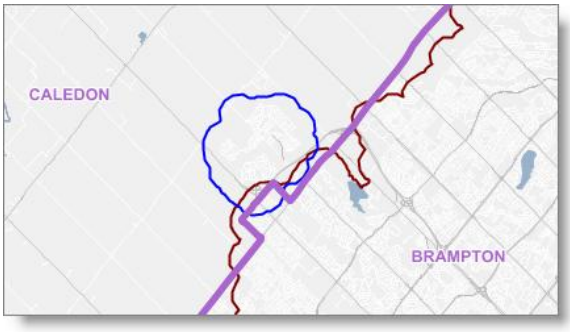
B. This record identifies two boundaries that intersect but there is no overlap. Since this does not provide useful information it will be removed from the table.

C. This record identifies two boundaries that overlap by 18% and therefore should be merged.

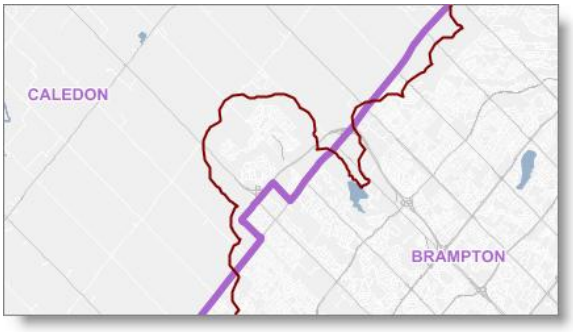
- The OverlapCheck table is saved and packed
- Next, any overlap duplication is removed from table. This is by examining each record's pair code, flagging it as either the first or second pair. When all records have been examined then those records flagged as the second pair are deleted.
  - A Fetch ...Next loop is set up to evaluate each record in the OverlapCheck table
  - For the first record, the row ID for the target object is read into a the variable vID1
  - For the first record, the row ID for the intersecting object is read into the variable vID2
  - For the first record, the value in the ProcFlag field is read into the variable vProcFlag
  - If vProcFlag = 0 then the following is performed
    - Selects all records where the PairCode field has "vID1, vID2" This should actually be the record "fetched". The ProvFlag field in this selection is updated with a value of 1.
    - Selects all records where the PairCode field has "vID2, vID1" This is the opposite pair of the one previously selected. The ProvFlag field in this selection is updated with a value of 2.
  - The next record is then fetched where the ProcFlag is zero.
- Selects and deletes all records where the ProcFlag =2.

MergeOverlaps

Due to the nature of how the model builds and assembles urban boundaries into a national coverage, it is common to have urban boundaries from different model areas overlap when the result files are combined. This is illustrated in the example below.



An example of a secondary boundary (blue) overlapping a primary boundary (red) at the border between the Caledon and Brampton model areas prior to the merge operation.



The result of the merge operation which detects overlapping boundaries and merges lower-class boundaries with high-class boundaries.

Because overlapping boundaries cause issues in other Genworth operations, they must be removed. This is done by physically combining boundaries together using the following rules.

1. A merge will only be performed when the overlap percent is greater than 0.005%
2. Primary boundaries that overlap another primary boundary are not combined. Overlaps are managed during the Trim and Fill procedure whereby the overlapping portions are physically removed.
3. Boundaries of the same class are merged together
    a. If a tertiary boundary intersects with another tertiary boundary, then they are merged together.
    b. If a secondary boundary intersects with another secondary boundary, then they are merged together
4. Boundaries with a lower class are always merged with the boundary of the higher class
    a. If a tertiary boundary intersected a secondary boundary, then tertiary boundary is merged with the secondary boundary
    b. If a tertiary boundary intersected a primary boundary, then the tertiary boundary is merged with the primary boundary
    c. If a secondary boundary intersected a primary boundary, then the secondary boundary is merged with the primary boundary

The methodology used for merging boundaries is to first select from the OverlapCheck table the specific type of overlap, for example, all occurrences where a tertiary boundary overlaps another tertiary boundary, see below. Once this subset has been created, it steps through each record merging the “Intersect” object with the “Target” object. Using the example shown below, boundary 596 is merged to boundary 563, boundary 691 is merged to 588, boundary 759 is merged to 737, and boundary 2248 is merged to 2147.

TargetName	TargetUrbID	TargetClass	IntersectName	IntersectUrbID	IntersectClass	Prov	PCT	PairCode
Shilo	563	Tertiary	Shilo	596	Tertiary	MB	34.8020483805419	563,596
Onanole	588	Tertiary	Onanole	691	Tertiary	MB	12.1724858526369	588,691
Silverwood	737	Tertiary	Silverwood	759	Tertiary	NB	0.6948087558776	737,759
Regina Beach	2,147	Tertiary	Regina Beach	2,248	Tertiary	SK	13.9382731378039	2147,2248

An example of the temporary table (TT1) identifying those tertiary boundaries that overlap another tertiary boundary.

The following describes the general merge operation as it applies to where tertiary boundaries overlap another tertiary boundary. It easily applies to other overlap types by replacing the blue text with a different class name.

- A query is performed on the OverlapCheck table to create a listing of all tertiary boundaries that overlap another tertiary boundary.

```
Select   * from OverlapCheck
where    TargetClass="Tertiary" and IntersectClass=" Tertiary" and PCT > .005
into     TT1
```

- If no records are selected by the query, then the program moves on to evaluating the next rule. If records are selected, then the following tasks are performed.
- A Fetch… Next loop is set up to evaluate each record in the TT1 table like what is shown above.
    - The first record in the TT1 is selected
    - The record ID for the target boundary is retrieved and stored in the variable vTargetRec
    - The record ID for the intersecting boundary is retrieved and stored in the variable vMergRec

    - The specific target record is selected from urban boundary table (alias TargetTable). Because the query involves a variable, a text string is built with the appropriate selection and a run command is executed to perform the query. See below.

```
cmdSelect = "Select * from TargetTable where RowID= " + vTargetRec + " into xx"
Run Command cmdSelect
```

    - If a record is selected, and this should be successful every time, then it is set as the target. This is a MapInfo Profession term for marking a specific object for editing.

    - Next, the specific merge record is selected from the urban boundary table (alias TargetTable). This is the intersecting boundary that will be merged.

```
cmdSelect = "Select * from TargetTable where RowID= " + vMergRec + " into zz"
Run Command cmdSelect
```

    - If a record is selected, and this should always be successful, then it is merged with the target object previously identified.
    - The next record in the TT1 table is selected and same steps are repeated.

- The contents of the TT1 table are then inserted into the BoundaryMergeLog table created earlier.

This code is repeated four more times, once for each of the following scenarios.

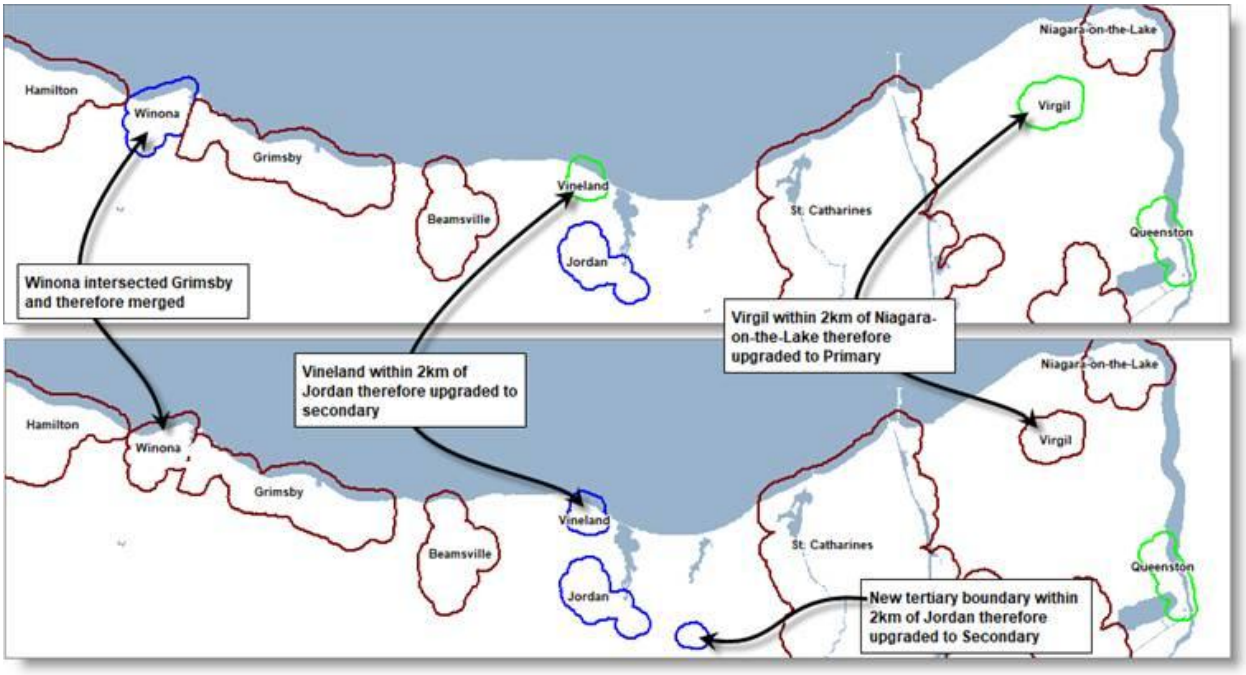
- Secondary merged with secondary
- Tertiary merged with secondary
- Tertiary merged with primary
- Secondary merged with primary

UpgradeWithinDistance

The process of upgrading boundaries based on the proximity to other boundaries is a three-step process. The first step is to generate buffers of the urban boundaries that are the subject of interest. Next, generate a list of boundaries that intersect those buffers. Finally, select boundaries in the list and upgrade their classification. These three steps are performed three times, once for upgrading secondary to primary, once for upgrading tertiary to primary, and once for upgrading tertiary to secondary.

The Upgrade operation first determines if a secondary or tertiary boundary is within a specified distance of a higher-class boundary. If any are detected, then the lower-class boundary is changed to match that of the higher-class boundary. The distance is defined by the “UpGrdDis” value in the configuration file. The following are the rules which govern a boundary upgrade.

1. If a secondary boundary was located within 2km of a primary boundary then its classification was upgraded to primary
2. If a tertiary boundary was located within 2km of a primary boundary then its classification was upgraded to primary
3. If a tertiary boundary was located within 2km of a secondary boundary then its classification was upgraded to secondary



Examples of both the merge and upgrade operations.

The following steps are performed to prepare for the update processes.

- Opens configuration file and retrieves the distance threshold value (buffer size). This value is located in the Var field for the "YpGrdDis" record.
- One fields is added to the urban boundary table, "TargetTable". The fields name is UpgradeInfo Char(40).
- Separate tables are created for each of the three boundary classifications. The files are called "TempPrime", "TempSec", and "TemTer". These tables are stored in the result folder and are deleted when processing is complete.
- The BoundaryMergeLog table is created. This table is created to contain a record of which boundaries were merged.

Upgrading Secondary to Primary

- **Step 1:** A temporary table is created to store buffers used in the analysis. Table is called TempBuffers.
- All primary boundaries are buffered using the value specified in the configuration file which is stored in a variable called BuffSize. Buffers are inserted into the TempBuffers table previously created.

- **Step 2:** Creates a list of all secondary boundaries that intersect the buffers

```
Select  TempSec.SecName, TempBuffers.CommName "PrimeName"
from    TempSec, TempBuffers
where   TempSec.Obj Intersects TempBuffers.Obj into TT3
```

- The results of this query are saved to table called “Deleteme” which is deleted later. This is necessary because the next SQL query can only be performed on a base table.
- A lookup table is then created using the Deleteme table. This query includes a GroupBy which ensures there is only a single listing of each secondary boundary name. This is necessary as the previous query included duplication when a secondary boundary intersected more than one primary boundary.

```
Select  SecName, PrimeName, Count(*)
from    Deleteme
group by SecName
order by Col3 desc into TT4
```

- The results of the query in TT4 are saved to a table called TempLUT2.Tab. The Deleteme table is permanently deleted.

- **Step 3:** All records in the TargetTable table that also exist in the TempLUT2 table are selected and stored in a temporary table called ObjectsToEdit. These are all secondary urban boundaries that are within in the specified buffer distance to a primary boundary.

```
Select  TargetTable.CommName, TargetTable.Class, TargetTable.UpgradeInfo
from    TargetTable,TempLUT2
where   TargetTable.CommName = TempLUT2.SecName
and     TargetTable.Class = "Secondary"
into    ObjectsToEdit
```

- Records in the ObjectsToEdit table have the Class field updated with the new class name “Primary”. The UpgradeInfo field is updated with a note to indicate what type of upgrade was performed, specifically “2 to 1 – Intersection *x* km”.

Upgrading Tertiary to Primary

- **Step 1:** The same buffers will be used, therefore, new buffers are not needed
- **Step 2:** A list of all tertiary boundaries that intersect primary boundaries is created and stored in table called TempLUT3.
- **Step 3:** All records in the TargetTable table that also exist in the TempLUT3 table are selected and stored in a temporary table called ObjectsToEdit. These are all tertiary urban boundaries that are within in the specified buffer distance to a primary boundary. The Class field is updated with the new class name “Primary”. The UpgradeInfo field is updated with “3 to 1 – Intersection *x* km”.

- A log file is created listing all urban boundaries where the classification was upgraded. This is done by selecting all records in the TargetTable that have an entry in the UpgradeInfo field. The results are saved to the results folder with the name BoundaryUpgradeLog.TAB.

Upgrading Tertiary to Secondary

- A new TempSec table is created. This is necessary because some of these records have already been up graded to primary and therefore should not be present in this table.

- **Step 1:** A temporary table is created to store secondary boundary buffers. Table is called TempBuffers2.
- All secondary boundaries are buffered using the value specified in the configuration file which is stored in a variable called BuffSize. Buffers are inserted into the TempBuffers2 table previously created.

- **Step 2:** Create a list of all tertiary boundaries that intersect the TempBuffers2. These records are store in the TempLUT4 table.

- **Step 3:** All records in the TargetTable table that also exist in the TempLUT4 table are selected and stored in a temporary table called ObjectsToEdit. These are all tertiary urban boundaries that are within in the specified buffer distance to a secondary boundary. The Class field is updated with the new class name “Secondary”. The UpgradeInfo field is updated with “3 to 2 – Intersection *x* km”.
- The records contained in the ObjectsToEdit table are appended to the BoundaryUpgradeLog

CreateDifferentVersions

Divides the national version of the urban boundaries into tables containing each of the three classes.

Divide National Version into PST Classes

If all 10 provinces were included in the model run then separate map layers are created for each of the three urban classifications, primary, secondary, tertiary. Additional versions are created in the NAD83 projection system.

- Checks if the ArraySize =10. If it does then the following is performed
  - Opens the national boundary table from the results folder (UrbanAreas_CAN.Tab).
  - Selects all records classified as “Primary”. Saves the result as “UrbanAreas_NAT_Prim.Tab”
  - Selects all records classified as “Secondary”. Saves the result as “UrbanAreas_NAT_Sec.Tab
  - Selects all records classified as “Tertiary”. Saves the result as “UrbanAreas_NAT_Ter.Tab

Combines Supplementary Boundaries

As the title states, combines all supplementary boundaries together. Before doing so the program check each province to see if any were created.

- Initializes i=1. The “i” variable is used to identify individual provinces in the ProvAbb() array
- Initializes LoopToggle3=0. This is used to force the model to perform a particular set of operations on the first province it encounters while a different set of operations is performed on all other provinces. Basically, it makes a copy of the first result file and then appends the remaining result files to it.
- A For…Next loop is set up to once for each province included in the model run
  - LoopToggle3 is evaluated, if it equals zero then the following operations are performed.
  - Opens the supplementary boundary file for the target province. This is the first province listed in the ProvAbb() array.
  - A check is performed to determine how many records are in the table. If there are more than zero, then a copy of the table is made and saved to the result folder with the name “SuppBoundsFINAL.Tab”.
  - The new table is then opened and given the alias SuppBounds. The table is opened so that it is ready and available for appending new records to it.
  - The LoopToggle3 variable is set to 1 indicating that the output table has been created and available.
  - If the previous check indicated there were no records in the supplementary table file then the program moves to the next province.

  - If LoopToggle3 is greater than 0, then the first part of the If statement has been run and must be skipped for all other provinces.

- The supplementary boundary table for the next province is opened and the number of records in this table is checked.
- If there are records in this table then they are appended to the SuppBounds table.
- If no records are found then the program moves to the next province.

Change Style for Supplementary Boundaries

- Checks to determine if the supplementary boundary table exists. If it does then it sets up a For…Next loop. If a table is not found, then nothing is performed.

- Before continuing variables are set defining the fill and stroke pattern that will be used in the region style change. The pattern will be a transparent fill with a thin black line for the border.

- The number of records in the supplementary boundary table is determined and stored in a variable called RowCount.
- A loop control variable called “n” is initialized with a value of 1
- The for…Next loop is set to run from n to RowCount. Essentially set to evaluate each record in the table.
  - Selects the first record in the table and inserts the object into a variable called objModify
  - A Case statement is set up that evaluates the object type and only attempts the modification if the object is a region.

```
Do Case ObjectInfo(objModify, OBJ_INFO_TYPE)
    Case OBJ_TYPE_REGION
        objModify = AlterObjectPenStyle(objModify, newPen)
        objModify = AlterObjectBrushStyle(objModify, newBrush)
    Case Else
        Note "The selected object is an unknown object type"
End Case
```

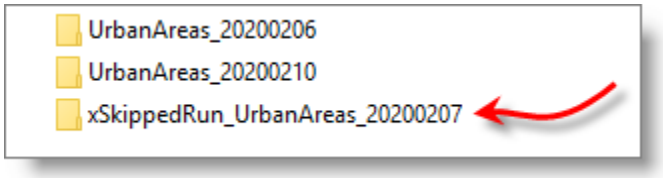
  - If the object is a region, which it will always be the case, then the update is performed.

SkipToFInalCombine

This procedure was created to help save time in the event the program crashes. It does this by performing all the final processing steps which combine the results for each province, merges and upgrades them where applicable, and saves different copies of the final version. Before running this procedure, make sure all provinces have been processed successfully. For example, if you were initially running all ten provinces and the program crashed during the seventh, then modify the configuration file and re-run the model on provinces 7 through 10. The results produced at the end of from this second run will only include provinces 7-10 and therefore can be ignored. Once the provinces have been processed, modify the configuration file so that the SKIP flag is to 1. Also make sure that the all provinces have been flagged as included in the process. Results will be stored in a special folder within the 4_Results folder.

Note: Most of the time crashes are a result of the program going faster then the hard drive can read and write information. This means the model can be rerun successfully without the same error occurring. Most of these types of errors have been rectified by inserting a two second pause in those location in the code where data is being saved and crashes tend to occur.

- This procedure does not get called until a small number of items get initiated, specifically, file paths are loaded, and the LogFile is created.
- When the file paths are loaded, the model checks the configuration file to determine if the SKIP flag has been set to 1. This check is performed at this time because it is the first time the configuration file is accessed. If it determines that the SKIP flag is set to zero, then the model runs normally. If it is set to 1, then it runs SkipToFinalCombine procedure.
- The SkipToFinalCombine procedure contains the following operations
    - Starts the timer
    - Sets the FolderName variable equal to “xSkippedRun”. This is done so the result folder will have a slightly different name than when the model runs normally. An example of this is shown in the image below.



    - The following procedures are called
        - CreateResultsFolder
        - FinalCombine
        - FindOverlaps
        - MergeOverlaps
        - UpgradeWithinDIstance
        - CreateDifferentVersions
    - Copies LogFIle to the output folder
    - The SKIP flag is reset to zero.

Description of Model Areas

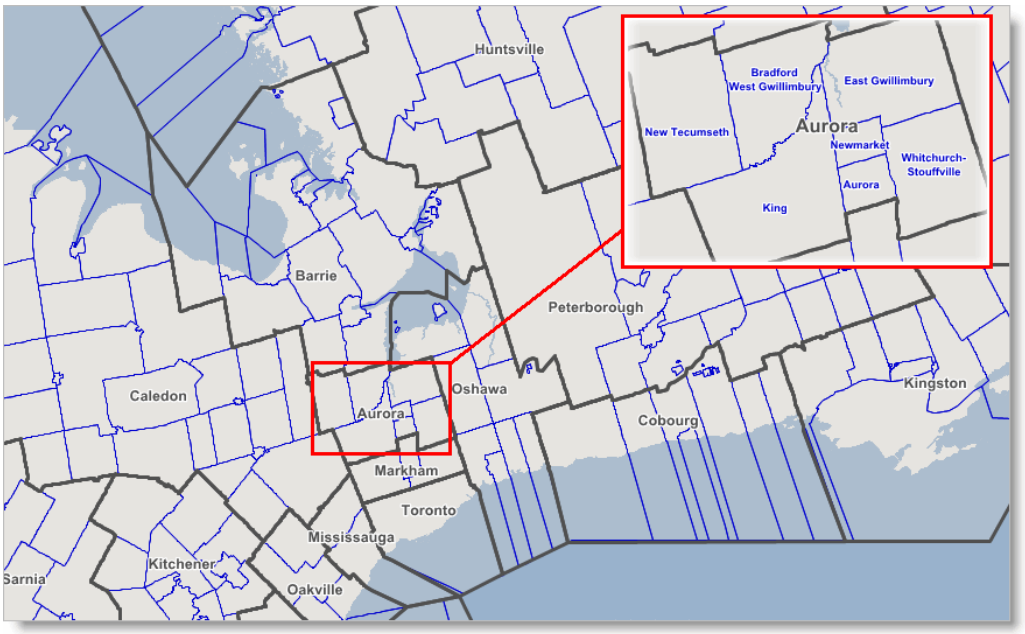
The most important unit of geography used by the model is the “Model Area”. This is because all urban boundaries are generated at this level which are merged together at the end of processing to form both provincial and national versions.

The sole purpose to creating model areas was to simply decrease processing time. In the early stages of model development, tests were performed to determine which level of geography would result in the best performance. At the provincial level, it was estimated that it would take two weeks to process all 10 provinces. This was due to the large quantity of queries of large data sets which would return only a small number of records. Testing at the municipal level proved to be much better, estimated at several days of processing. Although this was far better than the provincial level, it was noticed that many municipalities did not produce valid urban boundaries and therefore time spent processing those areas was wasted.

After additional testing with other geographic boundaries, it was concluded that a custom solution was in order. The criteria included the following.

- Must be large enough to ensure an urban boundary could be generated
- Must contain approximately 20-30,000 road segments. This was considered enough whereby urban boundaries could be generated.
- Extents of each boundary must coincide with existing municipal boundaries. This was critical since the vendor for the road data includes the municipality name in the road network file. This feature makes dividing the roads into another geographic unit much easier.

Based on these three criteria, municipal boundaries were merged into groupings that followed these criteria. The result was the merging of 5,098 municipal boundaries into 173 Model Areas. Running the model using these areas resulted in reducing processing time from several days to less than 17 hours.



Model Areas (thick grey outline) are the smallest geographic area used by the model for processing data. They consist of one or more municipality boundaries (thin blue outline), for example, the Aurora model area is made up of seven municipality boundaries as shown by the inset map above.

FileFunctionLib Liceense Agreement

FileFunctionsLib.dll License Agreement

http://communitydownloads.pbinsight.com/code-exchange/download/file-folder-and-drive-functions-library/

READ THE FOLLOWING TERMS AND CONDITIONS CAREFULLY BEFORE DOWNLOADING SAMPLE SOFTWARE FROM THE Pitney Bowes Software CODE EXCHANGE. BY CLICKING “I AGREE” BELOW AND DOWNLOADING THE SAMPLE SOFTWARE, YOU ACCEPT THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THESE TERMS AND CONDITIONS, PITNEY BOWES SOFTWARE IS NOT WILLING TO LICENSE THE SAMPLE SOFTWARE TO YOU.

1. License
    1. In this license agreement (“License Agreement”), you, the recipient of the license rights granted by this Agreement, are referred to as "Licensee" or "You." Pitney Bowes Software Inc. (“Licensor”) grants Licensee a perpetual, nonexclusive, non-transferable, royalty-free license to use the sample software and any related documentation downloaded from Licensor’s Code Exchange (“Licensed Product”). You may use, copy, modify and create derivative works of the Licensed Product, incorporate such derivative works within your products and redistribute such derivative works incorporated into your products to your customers, provided you do so under license terms as restrictive as those in this License Agreement.
    2. All rights to and in the Licensed Product, including, but not limited to, copyrights and trade secret rights, belong to Licensor or Licensor’s third party providers. Licensee shall not transfer or distribute the Licensed Product to others, except as permitted herein. Licensee shall ensure that any copyright and other proprietary notices contained in the Licensed Product are reproduced and included on any copies.
2. Term. This License Agreement is effective until terminated. Licensee may terminate this License Agreement by returning the Licensed Product to Licensor. Licensor may terminate this License Agreement if Licensee breaches any of the terms and conditions. Upon termination of this License Agreement for any reason, Licensee shall return the Licensed Product to Licensor. All provisions of this Agreement relating to disclaimers of warranties, limitation of liability, remedies, or damages, and Licensor's proprietary rights shall survive termination.
3. Disclaimer of Warranties. THE LICENSED PRODUCT IS PROVIDED “AS IS.” LICENSOR DISCLAIMS ALL WARRANTIES RELATING TO THE LICENSED PRODUCT, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
4. Limitation of Liability. LICENSOR SHALL NOT BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER, INCLUDING, WITHOUT LIMITATION, CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES, LOSS OF DATA, LOSS OF PROFITS OR LOST SAVINGS, ARISING OUT OF USE OF OR INABILITY TO USE THE LICENSED PRODUCT, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY.
5. General. Any software provided to Licensee by Licensor shall not be exported or re-exported in violation of any export provisions of the United States or any other applicable jurisdiction. Licensee may not assign or transfer any of the rights, duties or obligations hereunder. This Agreement shall be governed by and interpreted under the laws of the State of New York without regard to conflicts of law provisions. This Agreement constitutes the entire and only agreement between the parties relating to the subject matter hereof, and supersedes all prior or contemporaneous agreements and the terms in any purchase order or other document provided by Licensee. This Agreement may be modified only in a writing signed by the parties. Either party’s failure to enforce any provision of this Agreement will not constitute a waiver of the provision or of the party’s right to enforce the provision. If any provision of this Agreement is held invalid or unenforceable, the provision in question will apply with the modification necessary to make it valid and enforceable. If Licensee is the United States Government, the Licensed Products are provided with restricted rights. Use, duplication or disclosure by the United States is subject to restrictions as set forth in this Agreement, pursuant to DFARS 227.7202-3(a) (1995), or subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 (Oct. 1988) or subparagraphs (c)(i) and (2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable.

Backup of Configuration File

The following table is replica of the configuration file used by the model. In the event that the original file becomes lost or corrupt, this table can be used. Simply save the table as a .CSV with the name ConfigurationFile.CSV, Open it in MapInfo Professional and save it MapInfo's native .TAB file format. Make sure the output file is located in the /01_Program folder.

Update this table for final version.

ID	Var	Description	FileDate	FileStatus	FilePath	FileName	Instruction1	Instruction2
FP1	0	File path - root folder location for the Model			C:\Users\501286453\Documents\xPROJECTS\14 UrbanBoundary			
ERR	0	Trigger when an error is detected in any program component					If the Var field is 1, then and error has been detected in one of the program components. This forces the model to completely shutdown instead of running other program components unnecessarily	
SKIP	0	Set to 1 to run only the FinalCombine procedure. See User Guide						
GNW	0	GNW property point file	20180509	NEW	n.a.	GNWPropertyPoints01102018.TAB	Note: not used in current version of the model	Dates for testing below. Remove once model is done
RDS	0	Road network file	20160608	NEW	n.a.	RoadsLine.TAB		7/8/2013
MUN	0	Municipality boundaries	20161108	NEW	n.a.	MunicipalitiesRegion.TAB		9/8/2010
EPC	0	Enhanced postal code points	20160615	NEW	n.a.	EnhancedPostalPoint.TAB		9/8/2010
POP	0	Manifold population data	20160205	NEW	n.a.	PopTotal.TXT		
	0		20101212					
RoadCnt	25	Threshold of roads needed to build a boundary			Increase to shrink urban area size, reduce to increase the size			
BuffCnt1	20	Buf Count where density >= RoadCnt (Purpose 1)						
RoadCnt2	15	Lowest Buf Count threshold			Used to build supplementary areas that are below threshold but have many GNW properties			
CertCnt	100	Number of GNW points required to keep a supplementary boundary						
UpGrdDis	2	Upgrade Distance (KM) - Set to 0 for no upgrading			How close a 2nd or 3rd boundary needs to be to a 1st to have a class upgrade			
RUN_AB	1	Set Var = 1 to process this province						
RUN_BC	1	Set Var = 1 to process this province						
RUN_MB	1	Set Var = 1 to process this province						
RUN_NB	1	Set Var = 1 to process this province						
RUN_NL	1	Set Var = 1 to process this province						
RUN_NS	1	Set Var = 1 to process this province						
RUN_ON	1	Set Var = 1 to process this province						
RUN_PE	1	Set Var = 1 to process this province						
RUN_QC	1	Set Var = 1 to process this province						
RUN_SK	1	Set Var = 1 to process this province						
	0							
	0							
FP2	0	Filepath used for creating Model Area folders			C:\Users\501286453\Documents\xPROJECTS\"14 UrbanBoundary"			
	0							
E_Frm	0	Emails will be sent from this address (no quotes)					soheil.barkhodaee@genworth.com	
E_To	0	Email recipient list (include quotes)					"GNWMORTCANITPropertyHUBMapInfoAdministrators@genworth.com"