SEDRIS Transmittal Format to Compact Terrain Database

(STF to CTDB)

STF to CTDB User Manual

Prepared by:

Kevin Wertman
Rong Wang

Table of Contents

1 Introduction

The STF to CTDB compiler produces a CTDB from environmental data stored in a STF file set. There are several ways for the user to tailor the STF to CTDB compilation process in order to produce CTDB from a wide variety of data sets. This document describes how to configure the STF to CTDB compiler and provides tips for converting your SEDRIS data into the CTDB format.

The STF to CTDB has a rather large set of command line options that are used to control the compilation process. These options and their default values will be described in the 2nd section of this document. In addition, the compiler uses several configuration files to govern exactly how it is to map the SEDRIS data into CTDB. These configuration files will be described at length in the 3rd section of this document. The 4th section will document where the STF to CTDB will attempt to find these configuration files in its environment, as well as which files and directories the compilation process will create and/or modify.

The final section contains FAQs.

2 STF to CTDB command line

Accessing a printout of the STF to CTDB compiler's command line arguments can be done by issuing the command:

```
stf_to_ctdb -help
```

This will print out the list of command line arguments and their meaning. Captured here is a list of all current arguments, their default value, and a description of their meaning. Options with a shaded background are required for operation. Default values with a shaded background indicate the option points to something in the environment (a file or directory that may be read or used).

Option	Default Value	Description
`-transmittals <string>`	""	Comma (or space) delimited list of SEDRIS transmittal files to be converted.
`-output_base_name <string>`	""	CTDB output base name (no extension or path)
`-output_path <string>`	"."	Set the output directory where CTDB database should be created.

Option	Default Value	Description
`-data_path <string>`	`null`	Points STF to CTDB to alternate data/ directory where the following files must be located: coord.rdr, attribution.rdr, wgs84grd file, and defaults.rdr.
`-temp_path <string>`	`temp`	Sets the output directory for the temporary files created during compilation. STF to CTDB will attempt to create this directory if it doesn't already exist.
`-features <string>`	`features.rdr`	Tells the compiler which features file to use during the compilation.
`-version <integer>`	`1`	Specifies CTDB version (not related to CTDB format)

`-gridded |`
`-nogridded`
	`-gridded`	Specifies whether the CTDB can have gridded patches.

`-tinned |`
`-notinned`
	`-tinned`	Specifies whether the CTDB can have tinned patches

`-level_of_detail <HIGH or LOW>`	`HIGH`	Specifies what level of detail information should be extracted from SEDRIS Transmittal at.
`-zone_number`	`0`	Specifies the UTM zone number for CTDB. Generally not needed, unless the dataset borders 2 UTM zones.
`-post_spacing <integer>`	`0`	Allows user to specify the grid post spacing in the CTDB. If 0, the STF to CTDB compiler will attempt to determine the post spacing from the SEDRIS data provided.

`-fill_empty_posts |`
`-no_fill_empty_posts`
	`-no_fill_empty_posts`	Specifies whether empty posts should be filled in with data interpolated from the surrounding posts.

`-geoid_data_file <string>`	""	Specifies name of file containing NIMA geoid data.
`-gcs_mode <simnet, single, or multi>`	`simnet`	Specifies what type of CTDB to compile. "simnet" generates a flat earth SIMNET style CTDB. "single" and "multi" create single and multi-cell GCS CTDBs.

Option	Default Value	Description

`-use_gtrs |`
`-no_gtrs`
	`-no_gtrs`	Specifies whether a multi-cell CTDB creation should create GTRS based CTDBs (rather than old-style GCS based CTDBs)

`-cell_id <integer>`	`0`	Specifies GCS cell number to compile. Use only if -no_gtrs command line switch is set.
`-geotile_id <string>`	`NULL`	Specifies GTRS geotile to compile. Use only if -use_gtrs switch is also set.

`-gen_mc_header |`
`-no_mc_hdr`
	`-no_mc_hdr`	Instructs STF to CTDB to generate a multi-cell CTDB header file for the extents of the transmittals specified with the –transmittals option. No other processing is done.

`-list_cells |`
`-no_list`
	`-no_list`	Instructs STF to CTDB to print out a list of cell IDs for the given set of SEDRIS transmittals specified with the –transmittals option. If –use_gtrs is specified, the cell IDs will be GTRS based. No other processing is done when this option is specified.

`-list_holes |`
`-no_hole_list`
	`-no_hole_list`	Print the location of each hole found in terrain skin.

`-short_soil |`
`-long_soil`
	`-long_soil`	-short_soil assign soil values to entire patches rather than individual grid posts (-long soil).

`-use_metadata |`
`-no_metadata`
	`-no_metadata`	Use metadata file created with previous run of the compiler.

`-use_object_data |`
`-no_object_data`
	`-no_object_data`	Use temporary object data created with previous run of the compiler.

`-area_of_interest (utm or gd)`	`NULL`	When -area_of_interest is set then the created CTDB will have the extents specified via the min/max boundary arguments. This option is meaningless during multi-cell CTDB creation.
`-minimum_x_bound <float>`	`0.0`	Specifies minimum easting/longitude when -area_of_interest is set to utm/gd, respectively
`-minimum_y_bound <float>`	`0.0`	Specifies minimum northing/latitude when -area_of_interest is set to utm/gd, respectively.

Option	Default Value	Description
-maximum_x_bound <float>	0.0	Specifies maximum easting/longitude when -area_of_interest is set to utm/gd, respectively.
-maximum_y_bound <float>	0.0	Specifies maximum northing/latitude when -area_of_interest is set to utm/gd, respectively.
-memory_model (SMALL, MEDIUM, or LARGE)	MEDIUM	Specifies the memory model to be used by the SEDRIS API when extracting SEDRIS data.
-custom_pat_column <string>	NULL	Comma delimited list of EDCS 4.0 attribute labels for which columns will be added to the PAT and information will be stored when that code is found on a polygon in the STF. The specified attributes must map successfully to a FACC 2.1 code with coded values.
-transpose_world_3x3 | -no_transpose_world_3x3	-no_transpose_world_3x3	Specifies whether the compiler should transpose the world 3x3 transformation matrix found on certain models. In some older SEDRIS transmittals these matrices were produced incorrectly.
-keep_temp_files | -no_keep_temp_files	-no_keep_temp_files	Instructs the compiler to preserve temporary files used during the compilation process. If the temporary files are not preserved, then the –use_object_data and –use_metadata options will not work.
-help	-	Prints the list of command line arguments.

While these command line options may be specified on the command line every time, they can also be set using the environment variable STF_TO_CTDB_ARGS. Any argument stored within the STF_TO_CTDB_ARGS string will be read by the compiler. Any argument given on the command line overrides a matching argument in the environment variable.

3 STF to CTDB configuration files

3.1 Features File

STF to CTDB has several user editable configuration files. The most important file is the features file. It is this file that dictates the discovery of mappable SEDRIS DRM objects. This done by laying out all the combination of ECCs, EACs and DRM classes that the compiler will need to map into CTDB. If a combination is found in the transmittal, but not in the features file, that data will not be compiled into the CTDB. The compiler will log the combination in such situations. For more information on the mapping process see the STC 2002 STF to CTDB presentation. The STF to CTDB ships with a default features file named features.rdr that contains most mappings to be encountered when creating CTDBs. It may be that this file is enough to convert your STF. However in most cases some tweaking is required. The features file consists of the following sections:

Features File Section	Applicable DRM Classes	Description
`gridded_terrain`	<Property Grids>	Mapping to gridded terrain.
`polygonal_terrain`	<Polygons>	Mapping to polygonal terrain.
`water_terrain`	<Polygons>	Mapping to microwater.
`building`	<Geometry Models>, <Point Features>, <Areal Features>	Mapping to volume models (uses roofline algorithm for <Geometry Models>)
`obstacle`	<Geometry Models>	Mapping to volume models (uses bounding box algorithm for <Geometry Models>)
`bridge`	<Linear Features>	Mapping to bridges.
`river`	<Linear Features>	Mapping to laid linear rivers,
`road`	<Linear Features>	Mapping to laid linear roads.
`railroad`	<Linear Features>	Mapping to abstract railroads.
`trees`	<Point Features>, <Linear Features>	Mapping to trees and tree lines
`canopy_edge`	<Areal Features>	Mapping to abstract canopy edges.
`canopy_roof`	<Polygons>	Mapping to physical canopy roofs
`soil_defrag`	<Areal Features>	Mapping to soil defragmentation areas.
`powerline`	<Linear Features>	Mapping to abstract powerlines
`pipeline`	<Linear Features>	Mapping to abstract pipelines.
`political_boundary`	<Linear Features>, <Areal Features>	Mapping to political boundaries.

See the STF to CTDB mapping document for more in-depth information regarding the mappings above. Here is the brief sample of the syntax of the features file:

```
(
  ( polygonal_terrain
     ( SE_DRM_CLS_POLYGON
        ( TERRAIN ROAD VEHICLE_LOT WATERBODY PARK ))
  )
  ( gridded_terrain
    ( SE_DRM_CLS_PROPERTY_GRID
          ( TERRAIN TERRAIN_ELEVATION_PROPERTY_SET ))
  )
  ( building
    ( SE_DRM_CLS_GEOMETRY_MODEL_INSTANCE
          ( BUILDING ADMINISTRATION_BUILDING AERODROME AERODROME_TERMINAL ))
    ( SE_DRM_CLS_AREAL_FEATURE
          ( AIRCRAFT_HANGAR BUILDING BARRACK BARN ))
;
; this section represents a comment in a features file.
;   ( SE_DRM_CLS_POINT_FEATURE
;         ( BUILDING ADMINISTRATION_BUILDING AERODROME AERODROME_TERMINAL ))
  )
  ( bridge
    ( SE_DRM_CLS_LINEAR_FEATURE
          ( BRIDGE ENGINEER_BRIDGE OVERPASS BRIDGE_SPAN ))
  )
  …
)
```

Each section maps to zero or more DRM classes, each of which in turn specifies the EDCS Classification labels that must be attached to the DRM class to complete the mapping. So in the first example, any polygon found with a <Classification Data> with the EDCS concept "TERRAIN" will map to polygonal terrain.   In the case of buildings, there are two sections, one for <Geometry Model Instances> and one for <Areal Features>.   In this example, the section for <Point Feature> buildings has been commented out and will not be read by the compiler.

Any DRM class / EDCS label combination that does not have an entry in the features file, but could possibly be mapped, is logged in the file (output_base_name)_unmapped.log. All mapping that successfully take place are logged in the file (output_base_name)_mapped.log. In both cases, output_base_name is the same string that is provided with the command line option –output_base_name.

3.2 Attribution and Default Files

The next two files are used to help with attribution of specific CTDB objects.   The first file specifies what EDCS Attributes can be used to provide CTDB attribution. These are universal for any mapping that needs them, for example the "width" attribution section will be used to find attribution for bridges as well as trees.   It is not expected that the user will need to edit this file, but it is documented here in the case it is needed.   Here is an example of the syntax in the attribution.rdr file:

```
(
  ( width
    ( WIDTH BOTTOM_WIDTH TOP_WIDTH ROAD_TOTAL_USABLE_WIDTH
      ROAD_MINIMUM_TRAVELLED_WAY_WIDTH ROAD_SECOND_TRAVELLED_WAY_WIDTH )
  )
  ( height
    ( HEIGHT_ABOVE_SURFACE_LEVEL OVERALL_BRIDGE_HEIGHT PREDOMINANT_HEIGHT
      PREDOMINANT_HEIGHT_WITHIN_OBJECT PREDOMINANT_VEGETATION_HEIGHT
      MAXIMUM_OBSTACLE_HEIGHT )
  )
  ( density
    ( BRUSH_DENSITY WOODY_VEGETATION_DENSITY )
  )
  …
)
```

So to follow this example, any <Property Value> found with the EDCS Attribute label of "BOTTOM_WIDTH" will be used to determine the width of the object it is attached to. The following values are expected to be defined in the attribution.rdr file:

- density
- height
- foliage_height
- diameter
- opacity
- width
- length
- angle

The following entries in the attribution.rdr file are required for elevation grid data table processing and simply tells the compiler which EDCS attribute labels to use to determine the diagonalization, elevation, trafficability, etc. The default values for these entries should be sufficient for any regular STF to CTDB usage.

- diagonalization
- elevation
- trafficability
- x_axis
- y_axis

The second attribution related file is the defaults.rdr file. It is in this file that a user can compensate for missing or unknown data in the STF being converted. Its syntax is similar to the features file syntax. We'll use the following snippet as an example:

```
(
  ( building
    ( height 10 )
    ( width 10 )
    ( length 10 )
    ( angle 0 )
  )

 ( bridge
    ( height 10 )
    ( width 10 )
    ( length 10 )
    ( angle 0 )
 )
 ( trees
    ( height 10 )
    ( foliage_height 8 )
    ( diameter 2 )
    ( opacity .60 )
 )
 ( canopy_edge
    ( density .60 )
 )
  …
)
```

Each of the main sections defines the CTDB type whose defaults are being defined. The list is the same as those that appear in the features file. Under that are attribute / value pairs that specify the attribute and the default value. The attributes are the same as in the attribution.rdr file. For this example, if a <Point Feature> classified as a tree does not have a <Property Value> that gives the "foilage_height" attribution, then the default value of 8 meters is used instead. If that is not the desired foliage height, then the user can change it to a value more representative of the trees they desire.

4 STF to CTDB environment

The STF to CTDB compiler expects to find certain files in certain places in the environment it is being run in. The files and directories are relative to the place where the compiler executable is located. Here are the files the compiler expects to find:

data/defaults.rdr	defaults configuration file described in section 3.
data/attribution.rdr	attribution configuration file described in section 3.
data/coord.rdr	file needed for backend CTDB creation libraries.
data/wgs84grd.(big or lit)	file needed by SRM library for certain SRF conversions.

The compiler creates the following files when run:

(basename)_unmapped.log Created in present working directory.	list of unmapped DRM classes and the classification codes they contain, where (basename) is the CTDB base name given in the –output_base_name argument.
(basename)_mapped.log Created in present working directory.	list of all mapped DRM classes and the classification used to map them, where (basename) is the CTDB base name given in the –output_base_name argument.
.stf2ctdb_gcache Created in data path directory.	Cache of the grid post spacing for transmittals so the second time the compiler is run on the same transmittal it can lookup the post spacing.

Also created are a series of temporary files used during the compilation. These are deleted by default, but can be kept using the –keep_temp_files option so that they may be used to re-run certain portions of the compilation process. They can be found in the directory specified in the option –temp_path and all end with the suffix .tmp. These files can consume large amounts of space.

5 FAQ

5.1 How do I create a custom features.rdr file?

The best way to create a features.rdr file customized for a specific SEDRIS transmittal is to start with the features.rdr.empty file that can be found in the sample_features directory of the STF to CTDB compiler distribution. Running the STF to CTDB with –features features.rdr.empty as a command line option will cause the compiler to attempt to convert the STF to a CTDB with no mappings defined. Every primitive DRM class it encounters with a valid classification code will be logged in the file unmapped.log. With this unmapped.log file you will have a roadmap as to what DRM class / EDCS classification combinations exist within the transmittal. Copy the features.rdr.empty to a new features file and fill in the mappings based on what exists in the unmapped.log.

5.2 My terrain has large holes, or looks incorrect. What could be going wrong?

Some terrain generation processes classify the terrain based on what is lying on top of it, or integrate certain features right into the terrain. While this isn't necessarily the wrong way to generate terrain data, it tends to lead to some interesting mappings for the STF to CTDB compiler. For example, when an STF was encountered that had all terrain polygons that lay under roads classified as EDCS Classification ROAD, the features file did not pick up those polygons as terrain and left them out of the completed CTDB. This caused the CTDB to have holes in the terrain under the roads, and they weren't traversable at all by the vehicles in the SAF. If there seem to be holes in the created terrain, check the unmapped.log file for any <Polygon> DRM objects that have not been mapped. If the classifications for those DRM objects could be in any way considered as terrain, then add the mappings to the polygonal_terrain section of your features file. In many cases this cleans up the terrain holes.

5.3 How do I compile VPF and DTED datasets into a CTDB?

In order to create a CTDB from VPF and DTED data, first convert native datasets to STF using SEDRIS provided tools such as DTED to STF and VPF to STF, or other available conversion tools. Once the DTED and VPF data is in STF, the STF to CTDB can be used to create a CTDB.

The STF to CTDB can compile both the VPF and DTED STFs into a single CTDB using the multiple STF capability of the compiler. This can be accomplished by specifying both STFs in the –transmittals argument, for example:

```
-transmittals "my_vpf_data.stf my_dted_data.stf"
```

or

```
-transmittals my_vpf_data.stf,my_dted_data.stf
```

The extents of the created CTDB will be derived from the extents of the first transmittal specified.

Due to the nature of the VPF and DTED datasets, the following set of options is required to convert them into a CTDB using STF to CTDB in addition to the regular options:

- `-level_of_detail LOW`
- `-post_spacing 100`
- `-fill_empty_posts`
- `-notinned`

The post spacing must be specified because VPF and DTED datasets are stored in the geodetic spatial reference frame, and the STF to CTDB cannot correctly determine the post spacing of the SIMNET or GCS based CTDB from geodetic data. The value of 100 meters is just a recommended post spacing for DTED based datasets, and can be changed as required.

Since there is no soil or trafficability information stored in native DTED datasets, all terrain soil and trafficability in a CTDB created from a DTED STF will be set to default values for all terrain.

5.4 How can I validate and inspect my SEDRIS data?

It is always recommended that SEDRIS transmittals being converted using the STF to CTDB compiler be validated first. There are two tools provided by SEDRIS called Rules Checker and Syntax Checker that will validate the transmittal against the rules that govern the SEDRIS DRM. There are some extreme cases that if a transmittal fails the Syntax Checker then it could cause the STF to CTDB to crash because it expects the data it is reading the have valid syntax.

Another tool that is extremely helpful is the SEE-IT tool. This tool can find holes in the terrain represented in the STF as well as other problems that can cause an invalid CTDB to be created. Notably it can detect when road features don't connect properly, which would lead to disconnected road and river networks in the CTDB created by STF to CTDB.

The final tool to help with STF to CTDB conversions is the Side-By-Side viewer by AcuSoft. This tool possesses the capability to load the STF you converted and the CTDB just created and view them side by side to look for differences. It also allows you to investigate individual polygons by using the "polygon select" mode and viewing the corresponding SEDRIS data structures underneath the selected polygon. This is particularly useful for debugging possible soil and trafficability issues.

Directions for obtaining all tools mentioned above can be found at the http://tools.sedris.org website.

5.5 How can I create the single cells of a multi-cell CTDB in separate processes?

In order to create an individual cell or geotile for a large multi-cell CTDB, the user must specify the cell or geotile id on the command line in conjunction with the -gcs_mode multi argument. In order to find out which cells/geotiles the STF datasets covers, the -list_cells option can be used. Also, when compiling cells of a multi-cell CTDB individual, the multi-cell header must be created in a separate step. Here's an example of all necessary steps to create a multi-cell GCS based CTDB:

```
C:\stf_to_ctdb>stf_to_ctdb.exe -transmittals my_multi_cell_dataset.stf -no_gtrs -
list_cells
43080 43081 43440 43441
C:\stf_to_ctdb>stf_to_ctdb.exe -transmittals my_multi_cell_dataset.stf -no_gtrs -gcs_mode
multi -cell_id 43080 -output_base_name my_multicell_ctdb
…
my_multicell_ctdb.dir/cell43080.c7l is created.

C:\stf_to_ctdb>stf_to_ctdb.exe -transmittals my_multi_cell_dataset.stf -no_gtrs -gcs_mode
multi -cell_id 43081 -output_base_name my_multicell_ctdb
```

```
…
my_multicell_ctdb.dir/cell43081.c7l is created.

C:\stf_to_ctdb>stf_to_ctdb.exe -transmittals my_multi_cell_dataset.stf -no_gtrs -gcs_mode
multi -cell_id 43440 -output_base_name my_multicell_ctdb
…
my_multicell_ctdb.dir/cell43440.c7l is created

C:\stf_to_ctdb>stf_to_ctdb.exe -transmittals my_multi_cell_dataset.stf -no_gtrs -gcs_mode
multi -cell_id 43441 -output_base_name my_multicell_ctdb
…
my_multicell_ctdb.dir/cell43441.c7l is created

C:\stf_to_ctdb>stf_to_ctdb.exe -transmittala my_multi_cell_dataset.stf -no_gtrs -
output_base_name my_multicell_ctdb -gen_mc_header
…
Wrote multi-cell header file my_multicell_ctdb.dir/my_multicell_ctdb.m7l with the
following extents:

        SW Corner: (50.000000, 10.000000)
        NE Corner: (52.000000, 12.000000)

C:\stf_to_ctdb>
```

The process detailed above results in the very same multi-cell CTDB that would have been created with the following single command:

```
C:\stf_to_ctdb>stf_to_ctdb.exe -transmittals my_multi_cell_dataset.stf -no_gtrs -gcs_mode
multi -output_base_name my_multicell_ctdb
…
my_multicell_ctdb.dir/cell43080.c7l is created.
…
my_multicell_ctdb.dir/cell43081.c7l is created.
…
my_multicell_ctdb.dir/cell43440.c7l is created
…
my_multicell_ctdb.dir/cell43441.c7l is created
…
Wrote multi-cell header file my_multicell_ctdb.dir/my_multicell_ctdb.m7l with the
following extents:

        SW Corner: (50.000000, 10.000000)
        NE Corner: (52.000000, 12.000000)

C:\stf_to_ctdb>
```

The process is the same for GTRS based multi-cell CTDBs, with the exceptions that the -use_gtrs option is used instead of -no_gtrs and the -geotile_id option is used instead of the -cell_id option. The -list_cells argument will output geotile id strings if the -use_gtrs option is specified.