CDB Productivity Suite User Manual

Introduction

The CDB Productivity Suite combines several tools for the generation, validation, and visualization of CDB content. This manual describes each of these tools and presents examples to demonstrate the capabilities included.

Components

- **Command-Line Tool (cdb):** provides a command-line interface to various functionality of the productivity suite libraries.
- **Attribute Translator:** converts feature attribution between different data dictionaries.
- **3D Visualization (Inception):** visualizes content using the Unity3D game engine.

Command-Line Tool (cdb.exe)

The `cdb` command-line tool incorporates various functionality of the productivity suite libraries into a single executable. The basic usage is as follows:

```
cdb [options] <cdb_path> <command> [cmd_options] [cmd_parameters]
```

The first part of the usage specifies global options and the target CDB. The only global option is the ability to specify a log filename for output messages. For example:

```
cdb -logfile MyLog.log c:\cdb\CDB_Yemen_4.0.0 <command> ...
```

In this case, we are interested in doing operations on the Yemen 4 CDB and writing debug information out to the MyLog.log file.

The command parameter specifies the desired operation. For example, the INJECT command is used to inject data into an existing CDB. Details for the available commands are provided in separate sections below.

The cmd_options and cmd_parameters arguments vary based on the command selected. However, commands generally require a target component with three parameters: the dataset, component selector 1, and component selector 2. For example, to select the Yearly VSTI Representation from the Imagery dataset, the parameters will be "`Imagery 001 001`". The dataset may also be specified numerically (e.g. "`004 001 001`" or "`4 1 1`").

Data Injection (INJECT)

The Data Injection command populates a CDB component from source data. If the CDB does not exist, a new one is created with the new data. If the CDB does exist, source data is injected into existing tiles. For raster components, this overwrites pixels in the existing content only where source data overlap. For vector components, this removes existing tiles and replaces them with the new data.

Usage

```
cdb <cdb> INJECT <cmd_options> <dataset> <cs1> <cs2> <sources>
```

Where `cmd_options` may contain:

`-bounds <n> <s> <e> <w>`	Bounds for area of interest
`-workers <N>`	Number of worker threads (default: 8)
`-lod <lod>`	Target LOD

Examples

```
cdb C:\MyCDB INJECT Elevation 001 001 MyDEM.tif
cdb C:\MyCDB INJECT Imagery 001 001 MyImage1.png MyImage2.png
cdb C:\MyCDB INJECT -lod 0 GTFeature 001 001 MyModelPoints.shp
cdb C:\MyCDB INJECT -lod 4 RoadNetwork 002 003 MyRoads.shp
```

Smart LOD Generation (LOD)

The Smart LOD Generator creates lower level of detail elevation and imagery based on existing content. Rather than simply downsampling existing data, it finds the highest detail data and selectively injects it into lower levels of detail based on file timestamps. If no lower level exists, it is created as would be expected in typical downsampling.

Usage

```
cdb <cdb> LOD <cmd_options> <dataset> <cs1> <cs2>
```

Where `cmd_options` may contain:

`-workers <N>`	Number of worker threads (default: 8)

Examples

```
cdb C:\MyCDB LOD Imagery 001 001
cdb C:\MyCDB LOD Elevation 001 001
```

Sampling (SAMPLE)

The Sampler generates a subset of source data for a given component. Based on the bounding area and dimensions provided, it selects an ideal level of detail, loads tiles covering the region, and samples them into the output file.

Usage

```
cdb <cdb> SAMPLE <cmd_options> <dataset> <cs1> <cs2> <outfile>
```

Where `cmd_options` may contain:

`-bounds <n> <s> <e> <w>`	Bounds for area of interest
`-width <N>`	Width (x) dimension (default: 1024)
`-height <N>`	Height (y) dimension (default: 1024)

Examples

```
cdb C:\MyCDB SAMPLE -bounds 13 12 46 45 Imagery 001 001 out.jp2
```

Validation (VALIDATE)

The Validator provides information on missing geotypical and geospecific models and textures. It searches the GTFeature and GSFeature datasets to identify matching models and reports if the associated model files are missing from the CDB. If the model exists, it inventories the textures referenced by the model and reports if the associated texture files are missing from the CDB.

Usage

```
cdb <cdb> VALIDATE <cmd_options> <dataset> <cs1> <cs2>
```

Where `cmd_options` may contain:

`-bounds <n> <s> <e> <w>`	Bounds for area of interest

Examples

```
cdb C:\MyCDB VALIDATE GTFeature 001 001
cdb C:\MyCDB VALIDATE GSFeature 001 001
```

3D Visualization (Inception)

Inception is a Unity3D tool for visualizing a CDB database. It contains a dedicated README in the main menu of the application.

Attribute Translator

Overview

The attribute translator is a tool for converting feature attributes from a source dataset into the CDB data dictionary. It also supports translation to other data dictionaries.

Usage

This tool is run from the command line via the *Command Prompt* menu item in the *CDB Productivity Suite* program group. The executable is named **AttributeTranslator.exe**. A GUI wrapper is also available.

```
AttributeTranslator <RuleSet.xml> <input> <output> [layer1] [layer2] ... [layerN]
```

Where:

`<RuleSet.xml>`	Path and filename to XML translation rules
`<input>`	Input vector path and filename. Any GDAL supported vector format (such as Shapefile, GeoPackage, ESRI File Geodatabase) is supported.
`<output>`	Output vector path and filename. Supported formats are: Shapefile, GeoPackage, ESRI File Geodatabase. If the file exists, features will be appended.
`[layer#]`	Zero or more layers separated by spaces. If no layers are specified, all layers will be translated.

Rulesets

Concepts

Rulesets are defined as a hierarchy of conditionals (**if** tags in the xml file), structured as a tree. Each feature is passed through the tree, descending further as long as each condition is evaluated as ‘true’. Each conditional may contain tags that set attributes on the features. Once a conditional evaluates to ‘true’, **all** set tags directly inside the scope of the ‘ **if** ’ tag are executed.

Each conditional at that level are next evaluated. Once a condition evaluates to false, none of the set or if commands below are processed.

It is important to be aware that tags in XML are not guaranteed to be executed in the same order.

Layers

Features processed through the ruleset can be directed to a different layer in the output file by setting the **output-layer** attribute. For example:

```
<set attr="output-layer" literal="translated_linear_road"
type="string" />
```

The above will cause the feature to be placed in the translated_linear_road layer in the output File.

All features will have the **source-layer** attribute indicating the layer name that the input feature originated from.

Enumerations

Enumerations provide the capability to associate a symbol to a different value. This functionality exists to simplify the process of creating rules, as well as making the rules more readable. Some data dictionaries use numeric values to represent different semantic values. For example, Concrete may be specified as the integer **1**, Dirt as **2**, etc. In this example, the following enumeration can be specified:

```
<enum name="surface_material" type="int">
     <symbol name="Concrete" value="1"/>
     <symbol name="Dirt" value="2"/>
     <symbol name="Asphalt" value="3"/>
</enum>
```

Enumeration values are specified by using the literal value in a set tag to a value starting with the @ symbol, then the enum name, a colon, and then the symbolic name. Using the above enum example, if you wanted to set a Concrete value, you would use an **if** tag such as this:

```
<set attr="SMC" literal=”@surface_material:Concrete” />
```

In the above case, the SMC attribute would be set to a value of ‘1’ in the output feature.

Mapping Tables

Mapping tables provide a quick way to map a set of values to a different set of values. The same functionality could be processed with a series of **if** and **set** values, but using a map table simplifies this process.

```
<map-table name="yyy" default="1" type="int">
     <lookup source-value="Concrete"
map-to="@surface_material:Concrete" />
     <lookup source-value="Asphalt"
map-to="@surface_material:Asphalt"/>
     <lookup source-value="Rock" map-to="999" />
</map-table>
```

Conditionals

The type of evaluations used is specified with the **op** attribute in an **if** tag. Supported operations currently include:

- Always True: (always)
    - Always is a convenience operation that allows the user to organize a block of conditions and **sets** that are always processed.
- Equals: (eq)
- Not Equals: (neq)
- Less Than (lt)
- Greater Than (gt)
- In Set (in)
    - When the in operation is used, a set of values are specified as child **<Value>** tags. If the source attribute matches any of the child **Value** tags, the condition is **true**.

```
<if attr="land_route_type" op="in">
     <value>tank_trail</value>value>
     <value>trail</value>value>
     <set attr="FACC" literal="AP050"/>
</if>
```

Setting Attributes

‘set’ syntax:

```
<set attr="LNAM" source-attr='name' type="string" />
```

‘map’ syntax:

```
<map source-attr=”xyz” dest-attr=”abc” table=”yyy”>
```