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 example use cases to demonstrate the capabilities included.

Components

- **Data Injection (cdb-inject):** populates imagery and elevation from source data.
- **Smart LOD Generator (cdb-lod):** creates lower level of detail imagery and elevation tiles for existing data.
- **Validation (cdbinfo):** provides information on missing geotypical and geospecific models and textures.
- **Attribute Translator:** converts feature attribution between different data dictionaries.
- **3D Visualization (Inception):** visualizes content using the Unity3D game engine.

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.

CDB Data Injection

Overview

The CDB Data Injection tool populates CDB imagery and elevation from source data. By default, it selected level of detail based on the resolution of the source rasters. 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, overwriting pixels in the existing content only where source data overlap.

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 **cdb-inject.exe**.

```
cdb-inject [options] <CDB>
```

Where `<CDB>` is the path to an existing CDB.

Options:

`-logfile <filename>`	Filename for log output
`-workers <N>`	Number of worker threads (default: 8)
`-bounds <s> <w> <n> <e>`	Bounds for area of interest
`-elevation <file/path>`	Source elevation filename or path
`-imagery <file/path>`	Source imagery filename or path
`-build-overviews`	Perform LOD downsampling
`-lod <lod>`	Target LOD

Example

Suppose there is a set of imagery files in a source directory (D:\imagery) and a separate elevation file (D:\elevation\elev.tif) for the same area. To create a CDB (D:\MyCDB) for this source data:

```
cdb-inject -elevation D:\elevation\elev.tif -imagery D:\imagery
D:\MyCDB
```

This will create D:\MyCDB with imagery and elevation tiles at LODs matching the source raster resolution.

A new higher level imagery subset becomes available (D:\new_imagery.tif). This file overlaps a part of the first imagery and matches the LOD. To add that to an existing CDB:

```
cdb-inject -imagery D:\new_imagery.tif D:\MyCDB
```

This automatically detects the target LOD and injects it into the imagery already in the CDB. Only pixels that are covered by the new data are overwritten while existing pixels are preserved.

CDB Smart LOD Generator

Overview

The CDB Smart LOD Generator is a tool for creating 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

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 **cdb-lod.exe**.

```
cdb-lod [options] <CDB>
```

Where `<CDB>` is the path to an existing CDB.

Options:

`-logfile <filename>`	Filename for log output
`-workers <N>`	Number of worker threads (default: 8)

Example

Suppose there are two source imagery files that have been used to generate a new CDB (D:\MyCDB). One is a lower resolution (LOD 2 equivalent) and one is high resolution (LOD 9 equivalent) occupying a subset of the lower resolution area. The high resolution data is from 2019 and the low resolution data is from 2020.

To run the smart lod generator, run cdb-lod from the command line as follows:

```
cdb-lod D:\MyCDB
```

In the target CDB, tiles will be generated for LODs 3-8 for the bounds of the high resolution data. However, since the lower resolution data is newer, LODs 2 and lower will not be overwritten.

CDB Validation

Overview

The CDB Validation tool 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

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 **cdbinfo.exe**.

```
cdbinfo [options] <CDB>
```

Where `<CDB>` is the path to an existing CDB.

Options:

`-logfile <filename>`	Filename for log output
`-bounds <s> <w> <n> <e>`	Bounds for area of interest
`-gsfeatures`	Test GSFeature dataset
`-gtfeatures`	Test GTFeature dataset

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”>
```