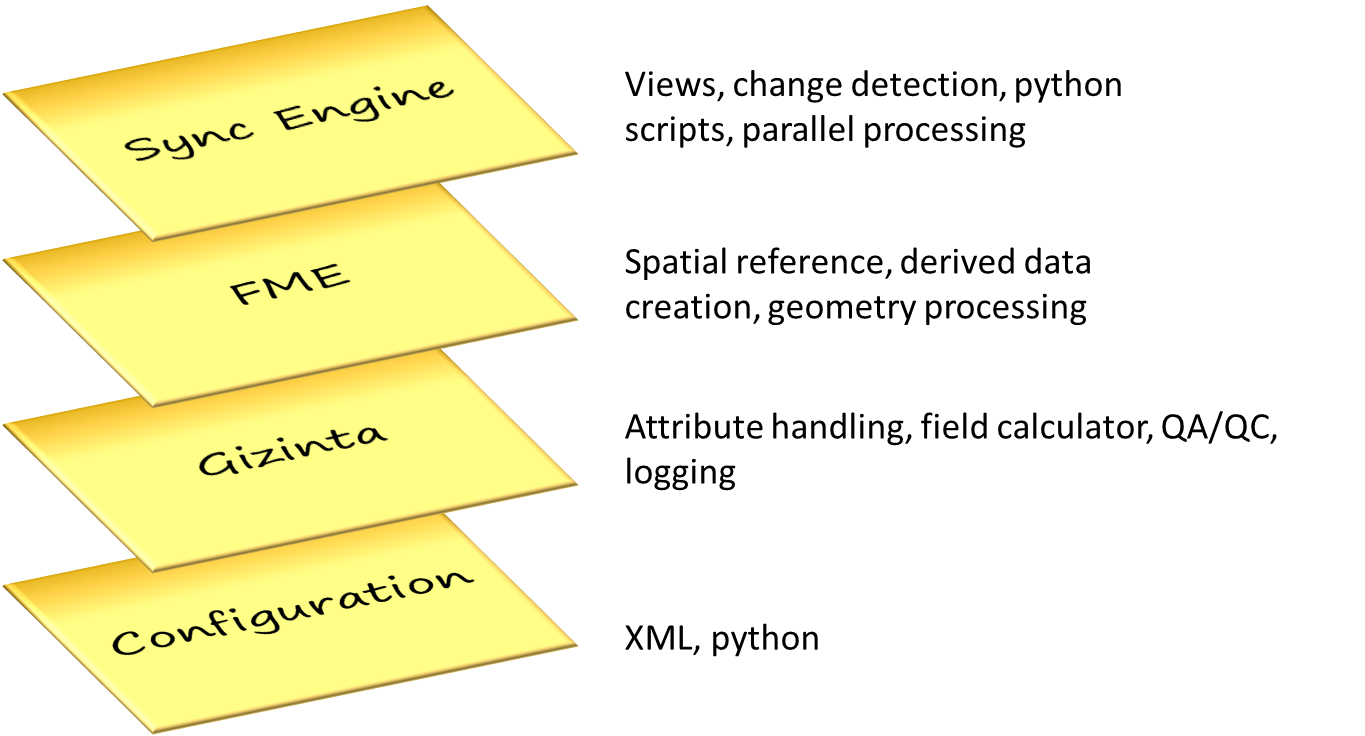
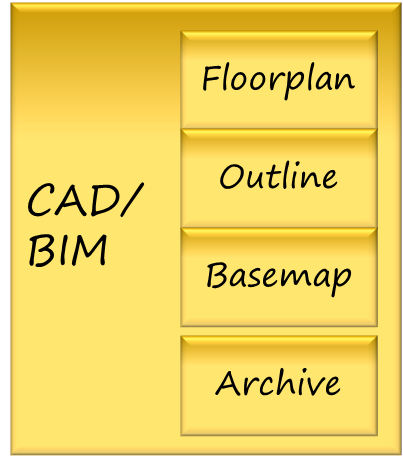
# Floorplan Sync Engine

The Floorplan Sync Engine is a sophisticated solution that automates the process of moving data from CAD drawings to Geospatial databases. There are 4 main parts to the system:



Generally speaking the Sync Engine could be set up to automate a number of ongoing data movement processes. The Floorplan Sync Engine comes with a specific configuration for an industry-leading data model for Interior Spaces called the Facilities Information Spatial Data Model (<http://FISDM.org>).



Floorplan Sync Engine

Facilities Database (FISDM)

The tools and techniques in the Floorplan Sync Engine have evolved over the course of several years. Vertex3 Inc., a small firm with significant expertise in this area, has been developing the tools while delivering to multiple project teams. The technologies are highly flexible and configurable, and just about any source drawing style and target database can be accommodated.

## How it Works

The Sync Engine is actually a simple concept:

* Gizinta provides the ability to have “playlists”. These are a set of source and target datasets with configuration settings defined in XML files.
* For each playlist
  + Read from one or more source drawings and write to a staging database
  + Each step typically involves an FME process that performs some of the complex geometry work
  + Perform QA and track the success of each step
* If all steps complete successfully, sync changes to a production database
* If there are errors, do not sync the drawing(s) and optionally continue
* Optionally delete the source CAD file if the load was successful

### The gseLoaderFME Script

The main script in the processing engine is called gseLoaderFME.py. Typically it is called from a .bat file with a few parameters. One example is shown below:

py\\gseLoaderFME.py fpLoadPlaylist.xml,fpDerivePlaylist.xml gseDataConfig.xml True False

In this case, the loader is called with the floorplan load playlist (fpLoadPlaylist.xml) and the floorplan Derive Playlist (fpDerivePlaylist.xml). The first step basically loads the data from CAD drawings with minimal processing, and the second step creates a number of derived features.

The next parameter is gseDataConfig.xml. This is a machine/server-specific file that provides a number of settings for the load process.

The next parameter is whether to auto-sync from the staging database with either True/False as the value.

The final parameter tells the process whether to stop after the first error (True) or continue if there are errors (False).

### The Floorplan Load Playlist (fpLoadPlaylist.xml)

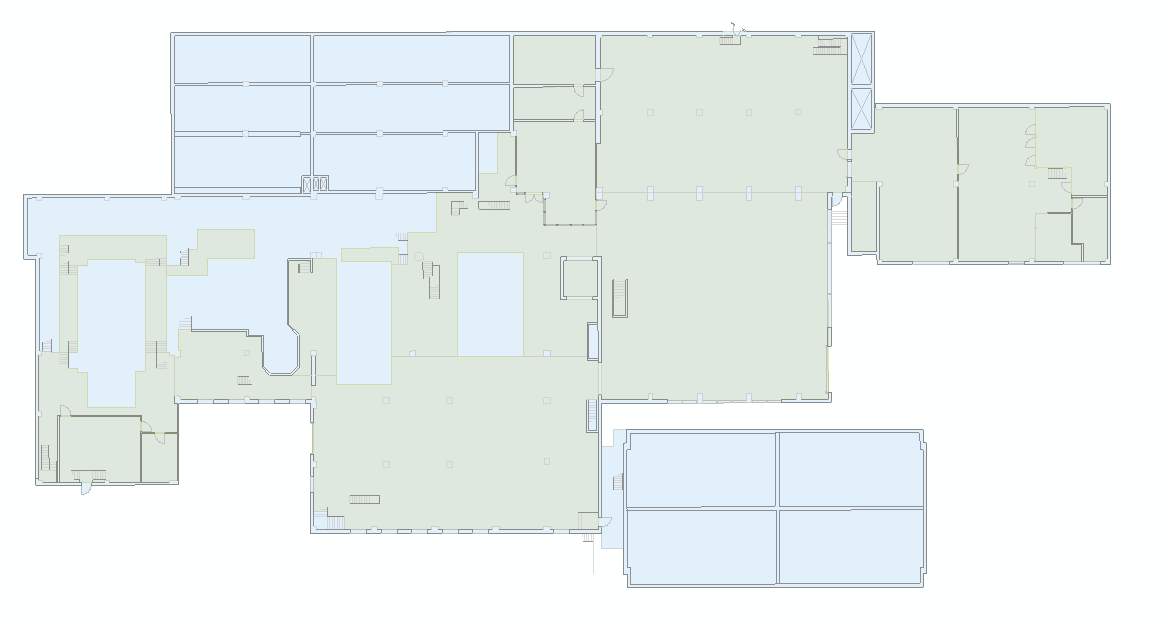
The load process performs the basic translation from CAD-GIS. Typically the configuration will load from a set of ArcGIS for AutoCAD feature classes but there are other options since the tools can read anything that FME can read. Processing is performed for:

* InteriorSpaces – the basic polygon representation of interior spaces
* Floor Polys – one or more polygons for the floor area/gross square footage area by floor
* Floorplan Lines – A set of lines aggregated into 1 feature by CAD layer. This typically results in about 5-10 features per floor rather than thousands of small features.

#### Load FME Process (gseFloorplanLoader.fmw)

The FME process for this playlist is gseFloorplanLoader.fmw. This process has several key functions:

* Reproject the data from the source coordinate system to the target coordinate system. This requires a World file for each CAD drawing that supports the transformation from 0,0 in paper space to x,y in a local coordinate system such as NAD83 State Plane or UTM Zone. By default data for the floorplan database is stored using Web Mercator, the standard for web mapping applications.
* Run the gzFieldCalculator. This calculates field names and values, and also sets the output dataset names for each record.
* Fix catwalks in Interior Spaces and Floor Polys. In CAD there is no way to represent rooms within rooms or donut holes in the polygons. For GIS use we want to remove those thin sets of lines from the outer boundary that define the donuts.
* Aggregate floorplan lines by floor number and autocad layer. This greatly reduces the processing time and also performance in GIS applications, while still providing the ability to color doors and stairs separately in applications.
* Run gzSourceTargetQA to check for QA / processing errors
* Run gzWriteData to log processing results and load to the dynamic database writer for the staging database.



Typical results of the floorplan load process – lines, spaces, and floor poly (rotated)

### The Floorplan Derive Playlist (fpDerivePlaylist.xml)

The derive playlist reads from the 3 feature classes in the source drawings, and it also reads basic building and floor information from the target database that will be used in processing. If there is no matching record in the Active\_Floor, Floor\_Level, and Building Outline feature classes this process will end with an error.

Several derived datasets are created:

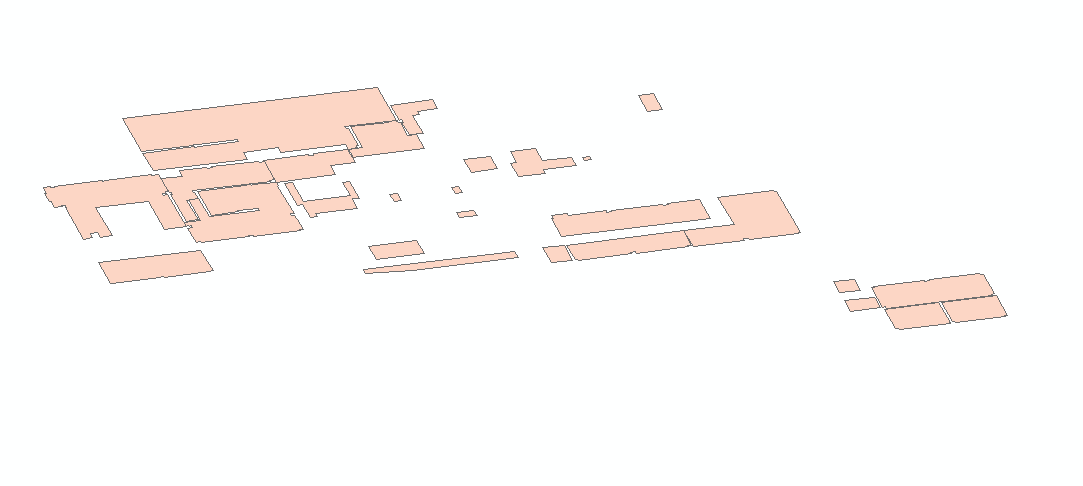
* Floor Outlines – polygons that are a convex hull around floorplan line features for each floor. The main purpose for this dataset is with the Esri Attribute Assistant and Data Driven Pages to simplify the process of adding other data to floors such as smoke alarms, fire extinguishers, and other assets using GIS applications.
* Floor Areas – this creates a single record, often with multi-part polygons, to represent floor area.
* Floor Points – for each Floor, a point is created from the location of the Building Outline Point. This provides a consistent, default point location for each floor. This is helpful for keeping a consistent extent/scale in a building when flipping between floors using Data Driven Pages.
* Interior Space Points – a point at the center of each Interior Space to be used for locating, labeling and routing purposes.
* Interior Spaces Tilted – derived features that provide an isometric view in 2D GIS applications and on paper to support visualization and mapping needs.

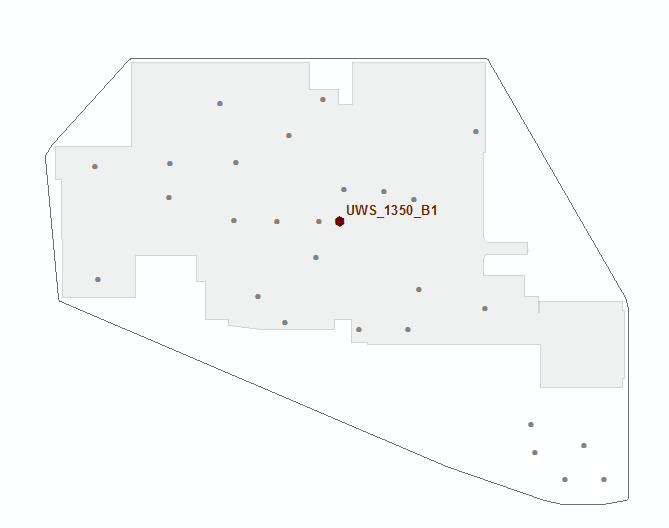
#### Derive FME Process (gseFloorplanLoader.fmw)

This process reads data from the staging database for the 3 main floorplan feature classes. It merges that data with Active Foor, Floor Level, and Building Outline data to construct the new feature classes. Most of the processing uses basic FME transformers, but there is one custom transformers to tilt the Interior Spaces. Key transformers are:

* HullAccumulator and Bufferer to create Floor outlines
* Aggregator by FloorID for the Floor Areas
* GeometryReplacer using the BuildingOutlinePoint for Floor Points
* InsidePointReplacer for the Interior Space Points

The gzTiltRooms custom transformer has an embedded python script that calculates new x and Y values for the tilt features. Basically the logic is to stack and spread floors by stacklevel and then reduce the Y values by a tilt factor to make them look tilted. The default tilt denominator is 2, which provides for basically a 50% tilt (1/tilt denominator).





The pictures above show a typical set of derived data for a drawing – floor outline, space points, floor point, tilted spaces, and floor area

### The Basemap Load Playlist (bmPlaylistAll.xml)

This process loads from source siteplan drawing and creates a number of features for the Basemap dataset. These features line up with the Esri ArcGIS for Local Government basemap feature classes in the Basemap feature class. This process is optional and is intended for organizations that manage their basemap data in siteplan drawings. If this is relevant for your organization more specific documentation will be provided for your implementation.

### The Basemap Derive Playlist (bmBldgDerivePlaylist.xml)

The Building derive playlist essentially aggregates the building features by BuildingID and then adds a point inside the polygon for each building. This ensures there is one row per building in the database. This is a follow-on for the basemap load process and is only relevant if you are loading basemap features from CAD or have special need to create these building features.

### The Building Outline Load Playlist (bldgOutlinePlaylist.xml)

This process loads a source polygon (rectangle) and point for each building. It also calculates the tilt angle of the rectangle relative to the world file/projection values. These 2 datasets are used in the floorplan processing for the rotation angle (Data Driven Pages and Tilt), and the Point feature geometry is used for each Floor Point.

### Server Config Settings (gseDataConfig.xml and .sde files)

In the ETL\serverConfig folder there are a few .xml files for site/server-specific settings. Also the ArcGIS .sde connection files are located in this folder and are the single place that the tools reference to locate the staging and target databases. In gseDataConfig.xml there are several settings defined in XML attributes

* cadFolder – a folder location or string that can be eval’d in python to provide a path to the source CAD drawings.
* stagingWS – the .sde connection file for the staging database. The default is GIS Staging.sde and is almost always left unchanged.
* productionWS – the .sde connection file for the target/production database. The default is GIS Production.sde and is almost always left unchanged.
* minTime – the actual date time or an expression that can be eval’d in python to control the datetime for files that should be included in processing.
* fileExt – usually “.dwg” for AutoCAD drawings.
* fmeExe – the location of the FME executable (different versions can be installed on the same machine)
* sourceEPSG – the source coordinate system to be used. Note this means a separate data config file and load process for each campus that uses a different spatial reference.
* runas – Currently only FME is supported but a data interop version may be available in the future.

If you are not familiar with .sde connection files, these files are used to store server information and username/password for access to Esri Geodatabases. There is significant documentation in the ArcGIS help if you are new to connection files.

### Config Files

The attribute and dataset mapping is controlled through a set of Gizinta configuration files stored in the ETL\config folder. Specific documentation on these files is available at <http://gizinta.com>. In the Floorplan Sync Engine configuration, both the playlists and the config files are all stored in the \config folder. The playlist XML file defines the location/name of the FME process to run and the name of the log file to be created, and it also is the place where specific/required parts of the CAD drawing names is set. For example, the CAD drawings may contain “XP” or “FP” or “FM” in the files that are floorplan drawings.

#### Sync Engine-specific Gizinta Configuration

Beyond the basics of the Gizinta configuration, the Sync engine config also includes the definition of database Views to be used. Basically a section for each dataset that requires change detection is configured using a ChangeDetection element in the appropriate Gizinta XML file. There are several XML attributes to be defined:

* exceptProductionView – the name of the view that compares the staging database table minus the production table for a dataset.
* exceptStagingView – the name of the view that compares the production database table minus the staging table for a dataset.
* idField – the identifier field for the dataset, typically FLOORID or SPACEID depending on the dataset.
* viewFields – a comma separated list of fields to be used in the views. Note that only the fields included here will be used to compare datasets, and SHAPE/Geometry fields can be included.

This information is used by the CreateDiffViews tools that are specific to your implementation. They are located in the ETL\gse.tbx and ETL\gseLocal.tbx Toolboxes. See the setup documentation for your site for specifics.

## Files and Folder structure

This section provides additional detail on the files and folder structure for the Sync Engine.

* + Top level folder will be name c:\apps\gizinta\gse<organization code>
  + Subfolders:
    - **CAD** – note that most of the drawings are often located somewhere else, this generally contain a few drawings for initial testing purposes
    - **Data Models** – Xml Workspace document for Geodatabase, also can store support tables like floor\_levels and other datasets useful for setting up the database
    - **Docs** – the location of this file and other documentation.
    - **ETL** – Folder that contains the data loading tools. This folder contains
      * .bat files that are used to run the processing,
      * a gse.py file that is used to map the folder structure for processing
      * a few Geoprocessing toolboxes called gse.tbx (general tools) and gseLocal.tbx (local processing tools specific to a specific site)

The ETL subfolders are:

* + - * + **batch** – contains tools to batch load/reload your database using parallel processing techniques
        + **config** – Contains Gizinta Xml files that drive the data and field mapping from source to target datasets.
        + **fme** – Contains FME workbench files and associated python files.

gseDrawing.py is the single place where the specifics of CAD file and folder naming conventions are translated into keys for SITEID, BUILDINGID, FLOORID, SPACEID, and other values are located. The config files use these functions.

gzfme.py provides a set of Gizinta functions that are used in FME processing.

* + - * + Other folders:

**Customxfmr** – contains the set of FME custom transformers to be installed on sync engine computers

**embed** – stub code to be inserted as FME startup and shutdown scripts

**log** – the location of FME log files

**Utilities** – and tools that are generally useful that don’t fit anywhere else

* + - * + **py** – Contains the python scripts used to set up the database, load data, and sync changes between databases. Subfolders are:

**dwgManagement** – you may have files in this folder that are specific to your site to set up folders/file structure for processing

**log** – the location of log files for processes in the \py folder

* + - * + **serverConfig** – Contains sde Connection files, and also process-specific configuration settings.
        + **views** – SQL files to create the views and perform some administration tasks. There are 2 sets of views:

**Differences** – the diff views defined in the config files. After the script is run you may keep a copy of the .SQL file here

**Publishing** – the definition of views to be published to other users in the organization. This joins data from multiple tables and provides floor and building-level data for feature classes.