# PlaneSmith - Coding

# Overview

PlaneSmith allows you to specify what form of output you get. This is done through writing code for the elements of your project: the level, definitions and objects. This is a documentation on coding in PlaneSmith. It is assumed you've read the Quick Guide.

A piece of code is a set of statements. These statements are bordered with **«** and **»**, which you can quickly insert with **Ctrl+B**. Everything outside of these brackets is considered a terminal statement and will be used as a simple string. (Newlines are interpreted as statements separate from terminals.) The output generator will interpret every statement and write the output into a file you choose.

Interpretation of statements depends on the context in which the generator is currently found. Each context is nested in its parent context. If a statement cannot be resolved in the current context, it will iterate through parenting contexts until it finds one which can resolve it.

Generating starts from your level code, this being its base context. In case you make a coding error, the generator will stop and a message will be shown.

**Important:** Keep in mind that it is possible to create an infinite loop. If this happens, you will have to halt the program. For this reason, PlaneSmith will be saving your work before generating starts.

# FECH statement

One of the statements is FECH. The keyword is a mix between fetch and for each. What it does is it collects a set of contexts - definition's or object's depending on what you specify. It then iterates through them and executes statements found in its body.

This is an example of a FECH statement:

«FECH.OBJ. , »name:«NAME»«END»

The first statement is the FECH statement. It is denoted by the keyword FECH, followed by a dot. It is specified we want to iterate through objects by OBJ (DEF is for definitions). After the second dot is a split terminal. This is a string that will be inserted between every two iterations (but not before first nor after last). A split terminal is optional (including the second dot) and it can only include terminal and newline statements.

The last statement is the END statement. Everything between FECH and END is the body of a FECH statement and is what its iterations base their output on.

This particular FECH will gather a set of objects, output **name:** followed bythe name of the object in that iteration. It will also separate all of these with a comma.

As you can see, FECH is a way to change into new subcontexts. FECH in the example above will iteratively go into the context of each object in the dictionary.

Documentation on each context and the statements they interpret follows.

# Context - Level

This is the starting context, as the generator starts from level code.

terminal

newline

Terminal and newline statements are interpreted in the standard way.

OBJ\_CNT

Number of objects in the level.

DEF\_CNT

Number of objects in the dictionary.

«FECH.DEF»*body*«END»

«FECH.DEF.*split*»*body*«END»

Gathers all definitions from the dictionary and iterates through them. Body is resolved in the context of each definition. Split terminal is inserted between each two iterations.

«FECH.OBJ»*body*«END»

«FECH.OBJ.*split*»*body*«END»

Gathers all objects from the level and iterates through them. Body is resolved in the context of each object. Split terminal is inserted between each two iterations.

# Context - Definition

This is the context of a definition.

terminal

newline

Terminal and newline statements are interpreted in the standard way.

«ID»

ID of the definition.

«NAME»

Name of the definition.

«IMG\_PATH»

Full path to the image file of the definition, including the file name.

«IMG\_NAME»

Name of the image file of the definition.

OBJ\_CNT

Number of objects in the level belonging to this definition.

«CODE»

Creates a subcontext belonging to this definition and interprets its code within that context.

«FECH.OBJ»*body*«END»

«FECH.OBJ.*split*»*body*«END»

Gathers all objects which belong to this definition and iterates through them. Body is resolved in the context of each object. Split terminal is inserted between each two iterations.

«AUTO\_CODE»

Creates a subcontext belonging to this definition and interprets its object-code within that context.

Definition's object-code is the code that is automatically assigned to new objects in a level belonging to that definition. Keep in mind that, when invoked like this, the object-code will be interpreted in the context of the definition, not an object!

# Context - Object

This is the context of an object.

terminal

newline

Terminal and newline statements are interpreted in the standard way.

«ID»

ID of the object.

«X»

X coordinate of the object.

«Y»

Y coordinate of the object.

«CODE»

Creates a subcontext belonging to this object and interprets its code within that context.

«DEF.*attr*»

Creates a subcontext belonging to the object's definition and attempts to resolve **attr** in that context.

# Conclusion

You are now prepared to utilize PlaneSmith to its full limits. Good luck!