Content Formats

**By Vinnie Simonetti**

This document is intended to explain the file formats used by SGDe.

## Hierarchy

* [Game](#_Game)
  + [Sprite Sheet](#_Sprite_Sheet)
  + [Map](#_Map)
    - [Entity](#_Entity)

## General format

For the format, XML is the primary data format and comprises of XML elements and attributes.

## \*.sgde

An SGDE is a XML document that links defines different document types together.

### Types

#### Sprite Sheet

A sprite sheet defines all the textures used by the game. This must always exist otherwise the content project will throw an exception. The XML element name is SpriteMaps.

##### Format

SpriteMaps contain the XML Elements: *SpriteMap*. A “SpriteMap” defines a sprite sheet/map that contains one image that will be used to draw an entity on screen.

###### SpriteMap

Attributes:

* ID: A positive integer that will be associated with this sprite sheet.
* Name (Optional): The name of the sprite sheet.

Elements:

* Source: A relative file path to the texture used by the Sprite sheet.
* Animation (Optional): A “Local” [Animation type](#_Animation).

#### Entity

An entity is an onscreen component that has a visual, location, and physics. It can have other attributes as well.

##### Format

Entities contain a variety of XML elements and attributes:

###### Attributes

* Name (Optional): A string that defines an entity’s name.

###### Elements

* Node: A node defines the transformation attributes of an entity.
  + Attributes:
    - [DID](#_DID) (Optional)
  + Elements:
    - Vector2: A two dimensional vector that contains two floating points.
      * Attributes:
        + X (Optional): Defines the value for the x-coordinate.
        + Y (Optional): Defines the value for the y-coordinate.
        + ID: What the Vector2 is used for. Supported value: “Translation” for the node’s position (default: 0, 0), “Scale” for the node’s scale (default: 1, 1)
    - Float: A floating point number.
      * Attributes:
        + Value (Optional): Defines the value that is used.
        + ID: What the floating point number is used for. Supported values: “Rotation” for the node’s rotation (default: 0)
* Sprite: A sprite defines the visuals used by the entity.
  + Attributes:
    - SID: This defines a positive integer that references the texture to use from the ID in the [Sprite Sheet](#_Sprite_Sheet). If this entity is being defined in a [Map](#_Map_1) then this is an optional attribute.
    - [DID](#_DID) (Optional)
    - Region (optional): This defines the region of animation frames to display. (Default: “Full”). The value “Full” means the full animation should be sued. If “Full” is not used then the value is two dash separated positive integers that define the start and end of the animation region. If one or the both values are left empty then the animation’s value will go in its place. So if “2-4” is used it will play frames 2, 3, and 4. If “-4” is used then it starts at the beginning of the animation and goes to frame 4. If “2-“ is used then it goes from frame 2 to the end of the animation.
    - Color (optional): The tint that the sprite will have when it is displayed. A hexadecimal color value in the format of AARRGGBB. (Default: FFFFFFFF)
    - Override (optional): Defines what an animation can override on a sprite. Detail: A sprite can have values assigned to it (such as tint) but the animation can also have the same attributes defined. This attribute defines what attributes of an animation can override the values of the sprite. An animation won’t always have a value for the sprite, in which case the sprite’s values are used until the animation has a value that (with this attribute) can override the sprite’s value. Supported values:
      * None: No override values (Default)
      * Tint: The sprite’s tint can be overridden.
      * RotationAbs/Rel: The rotation of a sprite can be adjusted by the animation. “Abs” is absolute rotation; “Rel” is relative rotation to the Node’s defined rotation. If a relative and absolute rotation exist in the same override then the content system defaults to absolute.
      * ScaleAbs/Rel: The scale of a sprite can be adjusted by the animation. “Abs” is absolute scale; “Rel” is relative scale to the Node’s defined scale. If a relative and absolute scale exist in the same override then the content system defaults to absolute.
      * FPS: When the entity is loaded the animation’s FPS is assigned to it. The developer can change this. If the developer changes the animation then and this override exists then the new animation’s FPS is assigned to the sprite and the developer’s FPS is replaced.
  + Elements:
    - Animation (optional): Any sprite specific animation can be defined with the [Animation](#_Animation) type (which this element is the parent element). The default animation can be set in the animation element.
* Physics: The entity specific physics to use. Components (everything is optional):
  + Attributes:
    - Enabled: Boolean value that enables physics. (Default: false)
    - EnableOnEnable: Boolean value that defines that if enabled is false then the physics system should not be enabled. (Default: false)
    - Collision: Boolean value if collisions can occur with this object. (Default: false)
    - Static: Boolean value if the entity is static. (Default: false)
    - PostSetup: Boolean value defining if the physics unit should be created after the physics system is enabled.
  + Elements:
    - Velocity: a two dimensional vector defining the entity’s velocity. It has an attribute “X” that defines a floating point value for the x-axis and “Y” that defines a floating point value for the Y-axis. (Default: 0, 0)
    - Forces: A list of forces, each element (name ignored) can contain two attributes: “X” that defines a floating point value for the x-axis and “Y” that defines a floating point value for the Y-axis. (Default: 0, 0)
* CustomEntity: This is a very specific element that defines what type the entity defines.
  + Attribute:
    - Base: The type that the custom entity is based off of. Done for type safety checks. This must be an [Assembly Qualified Name](#_Assembly_Qualified_Name). One important note is if the base type is not based off of SGDE.Entity then the custom type will not be saved. The exception to this rule is if the type is not able to be loaded (such as it being in a custom assembly and not in SGDE). If this is the case then the content processor doesn’t do any safety checks on the base type so if the type is not based off SGDE.Entity then a runtime exception will get thrown when the game tries to load.
  + Text: The inner text must be an [Assembly Qualified Name](#_Assembly_Qualified_Name) for the actual custom type.
  + Elements:
    - Constructor (Optional): A custom constructor for creating the entity.
      * <Child node, name ignored>: An argument to be passed into the constructor.
        + Attributes:

Index: Positive integer defining the argument’s position. This is primarily for use in out of order arguments as well as for assigning only partial arguments in a constructor.

Type: The Assembly Qualified Name for the argument.

Value: The value for the argument (based on the argument type). If the type contains a “Parse(String)” function then that is used to convert this value into the data type, as long as the type can be loaded.

#### Map

A map defines a game map.

##### Format

A Map comprises of two XML Elements: *Map* and *Resources*. “Map” contains the actual map of all the entities and the map physics. “Resources” lists all resources used by this map. “Map” must always precede “Resources.”

###### Map

* Layout: Defines the map layout. Child elements:
  + EntityPos: This positions the entity in a specific location on the map.
    - Attributes:
      * ID: Identifies the entity from [resources](#_Resources) that represents the base type for the entity.
      * [DID](#_DID) (Optional)
    - Elements:
      * Layout: This defines the actual entity-specific components of the map, from physics and position to texture. Everything is inherited from the base entity unless specified in the element. If the inner-text is “EntityDefined” then this element is ignored and all attributes and values that define an entity are defined in base entity. If “Defined” is specified then the entity defines components that override the base entity’s values. If “Defined” is used then Layout is the parent element for an [Entity](#_Entity) element.
* Physics: Defines the physics that are used on the map. Child elements:
  + Attributes:
    - Enabled (Optional): Boolean value defining if physics is enabled. The default value is “true” but if set to false the content system will ignore the rest of the physics element.
  + Elements:
    - Pharaoh: Defines the physics system for the entire map. It contains multiple elements to define how the map physics work:
      * Cell: A cell is a physics unit that the physics system uses to check collisions. If two physics cells overlap then the system checks collisions in them, else it ignores them and continues processing.
        + Attributes:

Width: A positive integer defining the width of the cell.

Height: A positive integer defining the height of the cell.

* + - * World: The size of the physics world, all physics interactions take place within this area.
        + Attributes:

Width: A positive integer defining the width of the world.

Height: A positive integer defining the height of the world.

* + - * Gravity (Optional): Gravity is self-explanatory as a force that affects all objects.
        + Attributes:

X: A floating point value defining the x-value of gravity.

Y: A floating point value defining the y-value of gravity.

###### Resources

* Entities: This defines a list of entities that are used by this map. The entire “Entities” list can have a [DID](#_DID) assigned to it.
  + Entity: An Entity defines what is internally called an EntityBuilder. An EntityBuilder is a class that is used to generate an Entity for the map. Components:
    - Attributes:
      * [DID](#_DID) (Optional)
    - Elements:
      * Entity: No relation to the parent element, this defines the actual entity type. This can be either a built in Entity type (see [Entity](#_Entity)) or a relative reference to the SGDE file that defines an Entity.

#### Game

This is the central file type and the only one required to be included in a game’s content project.

##### Format

A game comprises of two XML Elements: *Game*, and *Settings*. “Game” must always precede “Settings” otherwise the content system will not be able to process it.

A “Game” is the resource definition of what maps the game contains. “Settings” defines the game’s settings (screen, window, etc.) and map order.

###### Game

Child elements:

* Maps: This is a list of maps used by the game. Contains a list the following elements:
  + Map: An individual [game map](#_Map). This contains an ID attribute that contains a unique, positive integer that will be associated with the map. It contains a Layout element. A layout element can contain either a map or a map reference. A map reference specifies the relative SGDE file that is a Map.

###### Settings

Attributes:

* SpriteSheet: This is the entire game’s SpriteSheet and defines all textures and global animations. If this attribute doesn’t exist then the game will attempt to compile “SpriteSheet.sgde” which is the default file.

Child elements:

* MapList: A MapList defines the order of the maps and the initial map to load. It contains the following elements:
  + Map: This defines the map to use. Attributes:
    - Required:
      * ID: A positive integer that references the ID of the map to use.
    - Optional:
      * Name: The name of the map.
      * InitialMap: The first map to load. There can only be one InitialMap for a game.
* DefGameSettings: Default game settings for the game. The default value for all of these, match the default value of the settings used by the type SGDE.Game. Elements:
  + Screen: Defines the default settings for the game screen. Attributes:
    - Fullscreen (Windows/Windows Phone): Boolean value defining if the game is fullscreen or not. [Default: false]
    - VSync: Boolean value defining if VSync is enabled. [Default: false]
    - Multisample: Boolean value defining if multisample, or anti-aliasing, is enabled. [Default: true]
    - Width (Windows/Windows Phone): Positive integer defining the width of the game screen.
    - Height (Windows/Windows Phone): Positive integer defining the height of the game screen.
  + Game: Defines the default settings for the game itself. Attributes:
    - FixedTime: Boolean value defining if the game runs on a fixed time step or not. This means that Update is called on a specific interval instead of as-fast-as-it-can. [Default: true]
    - FrameTime: TimeSpan that defines the desired amount of time a single frame should last. The format of the time can be found [here](http://msdn.microsoft.com/en-us/library/3z48198e.aspx). [Default: “00:00:00.0166667”]
    - MouseVisible (Windows): Boolean value defining if the mouse cursor is visible on the screen.
    - Orientation (Windows Phone): A Microsoft.Xna.Framework.DisplayOrientation that defines the orientation of the game. As of XNA 4.0 the possible values are “Default”, “LandscapeLeft”, “LandscapeRight”, and “Portrait”. These can be separated by a pipe ‘|’ character to allow more than one orientation. [Default: Default]
    - WindowResizeable (Windows/Windows Phone): If the game window is resizeable. **Warning:** Games are not expected to be resized so appropriate measures must be taken if it occurs to prevent the game from “breaking” during execution. [Default: false]
    - Title (Windows/Windows Phone): The title that is displayed on the border of the window for the game. [Default: “”]

## Basic Types

### DID

A “DID” is a developer ID. It is an attribute that contains a String to reference an object by a developer.

### Animation

Animation defines a set of sprite animations that take place on a single sprite sheet. An animation can be used to define an actual, frame-by-frame animation, or a single static frame that defines a region of an image. When used with a Sprite, certain attributes of the Sprite might need to be overridden in order for the animation to work.

#### Format

An animation component contains multiple animation “sets” which define the actual animation. There are two types of animation: “Global” and “Local”. Global animations have already been defined and are simply referenced; Local animations are defined in this component. Data types:

##### Attributes

* ID (Global): Positive integer identifying the animation to be used for something like a Sprite. This is a 32bit value but has a specific format. The upper 16bits represents the sprite that contains the animation; the lower 16bits identifies the animation itself. The values are also one-based, so if the second animation on the first sprite is desired then the value would be 131073 or 0x00020001. The value of zero is reserved for the default animation (AKA no animation).

##### Elements

* AnimationSet (Local): This is a set of frames that define an animation.
  + Attributes:
    - Default (Optional): If this is the default animation to be used on a Sprite. This is ignored on a [Sprite Sheet](#_Sprite_Sheet). There can only be one default animation, if more than one exists then a warning is presented and only the first default animation set is considered default.
    - FPS (Optional): A floating point number defining how many frames per second the animation runs at.
    - [DID](#_DID) (Optional)
  + Elements:
    - Frame (Optional): A frame defines a single frame of animation. Only attributes are used in a frame. Every attribute is optional but if nothing is assigned then
      * Continue: Boolean value if the previous frame should be continued in this frame. The difference between this and FrameCount is that this can override the effects of the previous frame. Values from the previous frame are used in this frame and can be compounded, so if a Color is used on the previous frame but not on this frame then the previous frame’s Color is used. If the same Color does not exist on the previous Frame but exist on this frame then this frame is the only one with that Color. (Default: false)
      * FrameCount: Positive integer defining the number of frames this Frame represents. (Default: 1)
      * Effect: A [Microsoft.Xna.Framework.Graphics.SpriteEffects](http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.spriteeffects.aspx) to apply to the animation. (Default: None)
      * Color: The tint that the animation will have when it is displayed. A hexadecimal color value in the format of AARRGGBB. (Default: the tint of the sprite)
      * Frame: A rectangle that defines the area of the sprite sheet to display. The first value is the X position of the upper-left hand corner of the rectangle. The second is the Y position of the upper-left hand corner of the rectangle. The third value is the width of the rectangle. The fourth value is the height of the rectangle. (Default: the entire sprite sheet)
      * Origin: A two-dimensional floating point value defining the origin of the animation (this will also change the origin of the drawing in the inverse direction of the origin). The first value is the x-axis, the second is the y-axis. (Default: “0,0”)
      * Scale: A two-dimensional floating point value defining scale. The first value is the x-axis, the second is the y-axis. (Default: “1,1”)
      * Rotation: A floating point value defining rotation. (Default: 0)
      * RotationFormat: What format the Rotation is stored. This is ignored if Rotation is not used. Possible values are “Radian” or “Degree” which are self-explanatory. (Default: Degree)

## Other

### Assembly Qualified Name

An “*Assembly Qualified Name*” is a universal ID which is used to identify a type in .Net. The basic format is “<full namespace>.<class name>, <assembly>”. If you don’t know, an assembly is the DLL/EXE that you run or reference from your program.

### General container format

The first line is a standard XML declaration:

<?xml version="1.0" encoding="utf-8"?>

The root element is “SGDE” with the attribute “Version” which specifies the version, which is 1.0 (as of SGDE 1.0):

<SGDE Version="1.0">

The following element is one of the specified element types listed [above](#_Types).

### Not supported and why

* Animation, this one takes some explaining. Animation is supported but global animation, changing animation, and animation IDs are not. This is due to a lack of foresight into creating the animation ID system. The format, support, and everything mentioned is actually implemented and supported but it works only if used in specific cases. Also the “default animation” is not supported right now and everything in code is zero based right now. Because of that and flaw in the code everything that has no animation gets assigned to animation zero (the default animation) but the default animation could be one that is developer assigned so everything gets affected.
* Partial constructors: When making an entity the whole constrictor must be used, can’t have a “params”, “ref”, or “out” argument, and arguments can’t be skipped if in the current version of the code.
* RotationFormat is not supported right now (well it is but it could cause an exception if the value is not in floating point format).