Asset Tool

Asset Tool allows to post-process asset libraries exported from SceneComposer for following purposes:

- Shape Asset Library
- Append Languages
- Compare Asset Libraries
- Extract Bitmaps

For the change history related to asset library post-processing refer to:

- Asset Tool Change Log 3.0.0

- Shape Asset Library
  - Asset Shaping Feature Overview
  - Assetlib Shaper Usage
  - Shaper Pitfalls
  - Terminology
- Append Languages
- Compare Asset Libraries
- Extract Bitmaps
- Asset Tool Change Log 3.0.0

Shape Asset Library

Asset Tool Shaper component allows partitioning of asset libraries based on an XML rule set, which can be set individually for each project. Partitioning an asset library provides the following advantages:

Ease of variant handling
Reduction of flash time for partitioned updates
Faster boot up time

Asset Shaping Feature Overview

Partitioning

SceneComposer produces one "big" asset as a result of the asset generation process. In the application deployment process, there are often situations where the big contiguous asset needs to be partitioned in multiple data packages to meet deployment requirements.

Assetlib Shaper works on the "big" asset and it is able to partition it in multiple other assets. The resulting assets have exactly the same format as the "big" asset, meaning that they can be loaded by the existing AssetLoader component within Candera.

To partition an asset, besides Assetlib Shaper, you need: 1. the partitioning configuration: .xml file which tells the tool the rules used in this process; 2. the asset file: .bin file produced by SceneComposer at asset generation.

Partitioning configuration file defines:

- set of partitions;
- content of each partition (e.g., a set of bitmaps, one animation, all the vertex buffers of the initial asset);
- location where the partitions will be produced
- form in which the content will be present in each partition (compressed or not compressed)
- reserved spaces for each partition

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory="C:/temp" block-size="3KB" reserve-space-names="20%" reserve-space-tre
        <partition name="Scenes" master="true" reserve-space-asset="256KB">
        </partition>
        <partition name="BitmapsAndAnimations" reserve-space-lib="10%">
            <compression>
                <lib type="Animation"/>
            </compression>
            <rule type="Animation" item="*">
            <rule type="Bitmap" item="*" reserve-space-lib="20%"></rule>
        </partition>
```

```
    </partitions>
</shaper-configuration>
```

Above you see a partitioning configuration file. Basically, it defines two partitions; one named Scenes and one named BitmapsAndAnimations. The first partition is the master one. Each configuration must have only one master partition. This is the place for all asset elements that are not requested by other partitions.

To partition an asset, you will use this command:

```
  AssetTool -shape -p configurationFilePath assetFilePath
```

Space Reservation

In production mode it is very common to apply software updates, especially for content (e.g. face-lifts). To be able to minimize the flash time and write only minimum data, an asset size reserving mechanism is offered. The reserved space is an empty space that can be filled during future updates.

Features, expressed via rule-based system, for supporting asset size reservation:

- Reserve a buffer for the name table of the asset of a size provided in KB or as a percentage of the total space occupied by the name table.
- Reserve a buffer for the look-up tree for all or for a certain asset lib type for a specific asset of a size provided in KB or as a percentage of the total space occupied by the look-up tree.
- Reserve a buffer for at the end of each asset entry's data block for all or for a certain asset lib type for a specific asset of a size provided in KB or as a percentage of the total space occupied by that asset entry data block.
- Reserve a buffer for at the end of their data block for a certain asset lib type for a specific asset of a size provided in KB or as a percentage of the total space occupied by that asset lib data block.
- Reserve a buffer for at the end of asset's data block for all or for a specific asset of a size provided in KB or as a percentage of the total space occupied by that asset's data block.

The space to be reserved is defined in the partitioning configuration file. Let's consider the next partitioning configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory="C:/temp" block-size="3KB" reserve-space-names="20%" reserve-space-tre
        <partition name="Scenes" master="true">
        </partition>
        <partition name="BitmapsAndAnimations">
            <compression>
```

```
                <lib type="Animation"/>
            </compression>
            <rule type="Animation" item="*">
            <rule type="Bitmap" item="*"></rule>
        </partition>
    </partitions>
</shaper-configuration>
```

This configuration reserves 20% of the name table space, at the end of the name table, and 4Kb of space at the end of all the lookup trees of the asset.

To partition an asset with space reservation, you will use this command:

```
 AssetTool -shape -p configurationFilePath assetFilePath
```

Compression

Assetlib Shaper is able to compress asset data. Data is compressed at asset entry level for all or for a particular subset of the asset libraries.

You can specify enabled or disabled compression in the partitioning configuration file in two ways for flexibility purposes:

- at partition or partition collection level

```
        <compression>
             <lib type="Bitmap"/>
        </compression>
```

- at rule level

```
         <rule type="Bitmap" compress="true"/>
```

You should regard compression specification as a special request. For example; if you don't specify compression at all, no data will be compressed. Another example: when you want all data to be compressed in a partition, except for that of a particular asset entry, you should specify compression for all libs at partition level and use compress attribute with value "false" in the rule that targets that particular element. The next partitioning configuration file models this situation.

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory="C:/temp">
```

```
        <partition name="BitmapsAndAnimations" master="true">
            <compression>
              <lib type="Bitmap"/>
              <lib type="ShaderProgram"/>
              <lib type="VertexBuffer"/>
              <lib type="RawResource"/>
              <lib type="Font"/>
            </compression>
            <rule type="Bitmap" item="TheBitmapX" compress="false">
        </partition>
    </partitions>
</shaper-configuration>
```

Each entry that can be compressed defines in its data section regions, compression being applied to each such region individually. Each region has a so called Reserved field. This contains the value 0, if no compression was applied to the region, or the uncompressed region size, if the region is currently compressed. Please note that entries without data regions cannot be compressed. Only Bitmap, ShaderProgram, VertexBuffer, RawResource and Font entries can be compressed.

To partition an asset with compression, you will use this command:

```
  AssetTool -shape -p configurationFilePath assetFilePath
```

Language Import

Assets can contain internationalized text resources in a dedicated area. The text resources are storead as language asset entries in form of proprietary compiled language packs. Multi-language aware applications can use the asset to retrieve text items from a specific langauge via a Candera API.

You can import into an asset compiled language packs. Please note that a compiled language pack contains the text translations for one language only. You will specify the name of the language to which the compiled file belongs.

You can add any number of language packs to the asset by using the following command:

```
  AssetTool -shape -l assetFilePath [LanguageCode:CompiledLanguagePackSourceFilePath]+
```

Language asset entries are considered common asset entries, so anything that you can do with other asset entries, you can do with language asset entries as well. For instance, you can partition an asset so that one language goes in a partition, while all the others are stored in a different partition.

Asset Profiling

Asset profiling is used for reviewing asset content.

To profile an asset, you need:

- the asset to be reviewed
- the block size (data granularity) for profiling output representation

The output is an image, textual (txt file) or graphical (excel file), showing the components of the provided asset including the reserved space represented with the requested granularity.

As mentioned, there are two types profiling:

- Textual: Text file is created showing the content of each data block identified by its address, one data block per line. This type of presentation eases asset content comparison in the updated/new/removed elements identification process.

```
AssetTool -shape -st blockSizeInBytes assetFilePath destinationFilePath
```

- Graphical: Excel file is created showing with colors, data entities, multiple data blocks per row, highlighting the free spaces (where new data can be added in case of asset update). Each data entity and free space is shown with size in bytes and, if applicable, number of entries (for name table and lib headers).

```
AssetTool -shape -sx blockSizeInBytes assetFilePath destinationFilePath
```

Address	Info	Bytes	Entries
0x00000	Asset Header	528	
0x00210	Name Table	888	44
0x00588	Free Space	30720	
0x07D88	Other Header Data	180	
0x07E3C	Animation Lib Header and Lookup Tree	12	
0x07E48	Free Space	10240	
0x0A648	CompositeAnimation Lib Header and Lookup Tree	12	
0x0A654	Free Space	10240	
0x0CE54	AnimationGroup Lib Header and Lookup Tree	12	
0x0CE60	Free Space	10240	
0x0F660	Bitmap Lib Header and Lookup Tree	12	
0x0F66C	Free Space	10240	
0x11E6C	Display Lib Header and Lookup Tree	12	

Asset Partition Updating

Software updates are very frequent in production, so that it is critical to reduce the time required to deploy these updates. Flashing might take a large amount of time if for a software update the

entire CGI Studio asset or asset partitions are considered. In order to reduce the flashing time and also the amount of data to be written on software update, Assetlib Shaper provides the asset partition updating functionality.

Updating process requires:

- current (original) asset partitions location;
- modified asset;
- partitioning configuration file used in obtaining the current partitions.

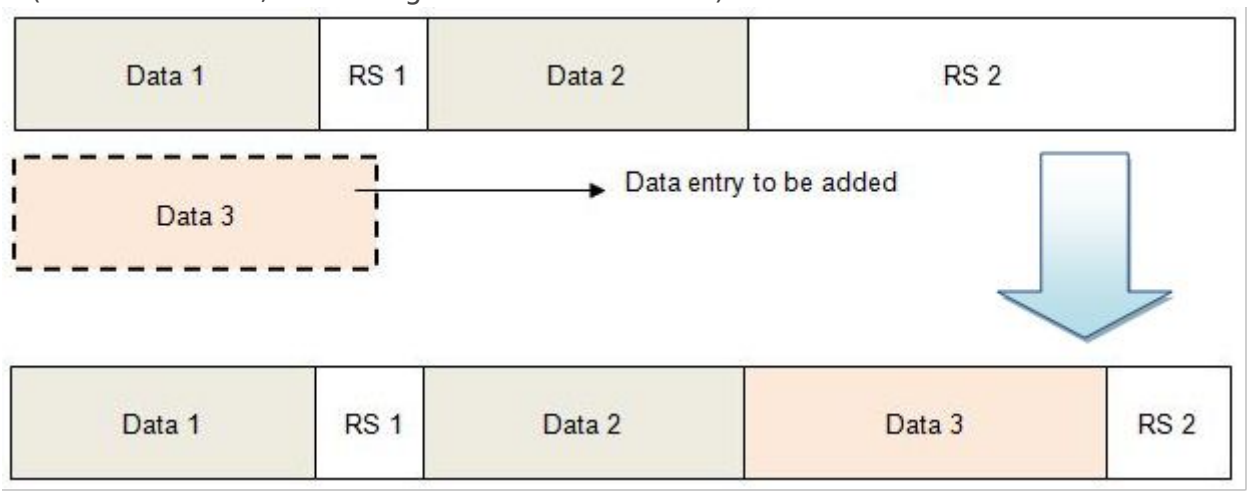
The following command is used for asset partitions update:

```
AssetTool -shape -u currentAssetPartitionsDirPath newAssetFilePath configurationFilePath
```

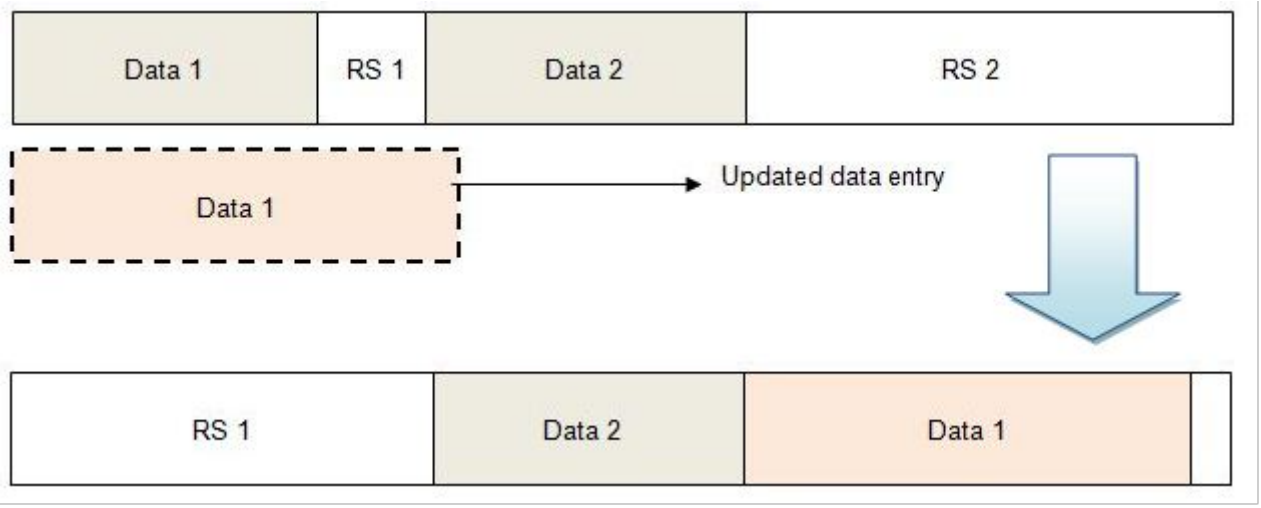
It is important to note that for the updating process to work, it is mandatory for each partition of the current asset to have at its location its associated .xml file, with the same name as the partition file, the asset report file. This .xml file was produced during the partitioning of the current asset or during a previous asset update operation.

The updating process is described in more details. There are three basic operations during an asset update:

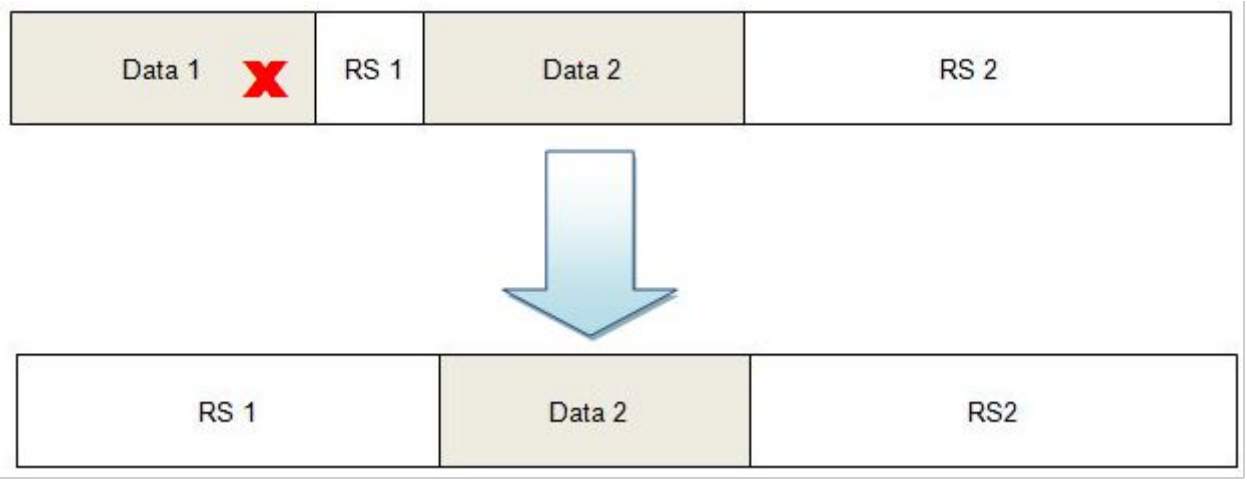
- **Adding an Item**: If a new data entry is to be added, this new entry will be located into the reserved space (RS) that fits it best or at the end of the asset if no reserved space fits it (in this last case, increasing the size of the asset).



- **Updating an Item**: If an existing data entry is to be updated, the updated data entry tries to fit the old data entry space including the adjacent reserved space (if there is one). If it doesn't fit this space, the updated data entry is entirely moved to the reserved space that fits it best or to the end of the asset if no reserved space fits it (in this last case, increasing the size of the asset).



- **Deleting an Item**: When removing an existing data entry, the space occupied by it is left blank, becoming a reserved space that can be used during the next asset updating operations.



The updating process creates several directories at the location of the modified asset involved, so that full permissions are required for that location (read, write, execute). The output directory that contains the updated asset partitions is named "Result - Updated Partitions" and it is located in the parent directory of the modified asset involved in the process. The other created directories are automatically removed after obtaining the results. Please note that the target directory provided in the partitioning configuration has no effect in the updating process, the output directory being always created at the location of the modified asset involved in the process.

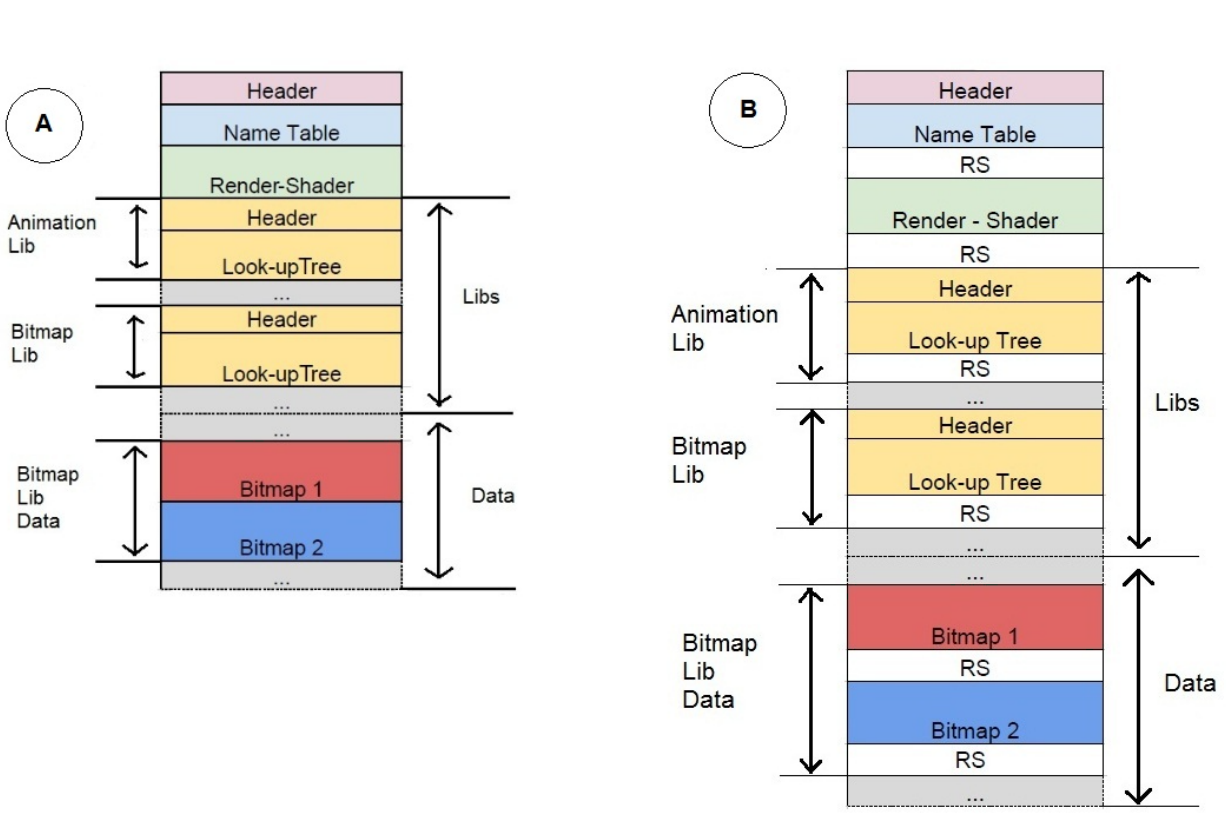
The incremental asset updating is possible. For this, you will use the output of an update operation as input for a new asset update.

Asset Update How To

If an evolution of the original asset content is expected, in order for the asset update process to work, there are some early decisions that you might want to consider:

- Make sure that at the partitioning of the original asset the asset report file exists for the original asset (the asset report file of the original asset was generated by SceneComposer).
- Use a partitioning configuration that reserves space to: name table (mandatory), render-shader zone (recommended), asset library list of items (also knows as look-up trees) where new elements are expected (mandatory); you can optionally reserve space to the end of asset library entries, library data and asset. An example of such a configuration is presented below, also with the structure of the original asset (A) and that of the partitioned asset (B), where RS stands for reserved space:

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory="C:/temp" reserve-space-names="2KB" reserve-space-render-shader="40%"
        <partition name="All" master="true">
             <rule type="Animation"></rule>
             <rule type="Bitmap" reserve-space-entry="30%"></rule>
        </partition>
    </partitions>
</shaper-configuration>
```



Using the above partitioning configuration implies that the name table of the asset is not expected to extend with more than 2KB, the render-shader zone will not grow in size with more than 40% and the look-up trees (or list of items) of each library are not expected to extend with more than 1KB.

The amount of space to be reserved depends on the asset content evolution expectations. Please note that if the name table, render-shader zone or asset library look-up trees extend beyond the reserved space during the update process, the process fails, so that it is better to reserve more than too little space.

Starting the update process, you'll need:

- Set of partitions to be updated with asset reports for each partition (generated during partitioning or during a previous update process).
- Original partitioning configuration file.
- Modified asset with its asset report (generated by SceneComposer).

Supported Library Types

Supported Asset Library Types

List of supported asset library types:

- Animation
- AnimationGroup
- Bitmap
- CameraGroup
- ClearMode
- Display
- Font
- RawResource
- ReferencedTemplate
- RenderTarget
- Scene2D
- Scene
- ShaderProgram
- Composite2D
- Composite
- TextStyle
- Theme
- Transition
- VertexBuffer
- AppearanceCollection
- Appearance
- Material
- RenderMode
- UniformSetter
- Texture

- Language

Assetlib Shaper Usage

Usage Considerations

Assetlib Shaper uses as main input an asset produced by SceneComposer.

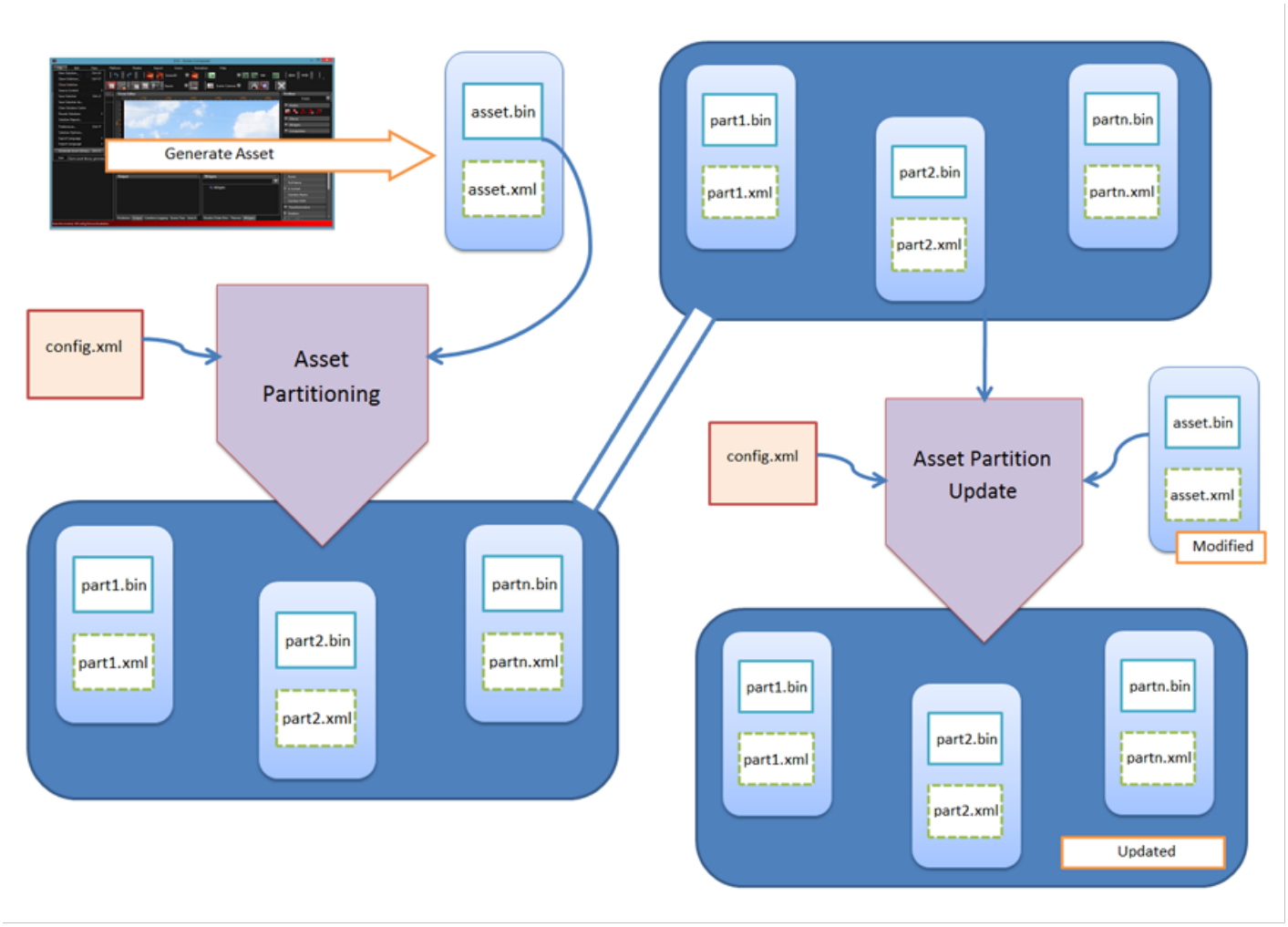
Besides the asset file, during the asset generation, SceneComposer also creates the asset report file at the same location and with the same name as the asset file (only their extensions differ). It is important to keep the report file of the asset in the same location as the asset (for example, asset partitions update works only when all the asset report files are at the same location as their associated asset files).

During the partitioning process Assetlib Shaper is looking for the asset report file of the asset provided as input. If this file is not found in the asset location, the partitioning process will output only the partitions of the given asset, no asset report being produced for these partitions. In this case, Assetlib Shaper shows a warning because it won't be possible to further update these partitions (with the content of an updated version of the original asset).

Example: you have an asset and its asset report (produced by SceneComposer during asset generation) and the partitioning configuration file. The general workflow of Assetlib Shaper:

- Add language packs to the asset if multi-language support is required. Please note that if translatable texts are used in the SceneComposer solution, a default language pack is automatically added to the asset during asset generation.
- Run the partitioning process, obtaining the asset partitions, each partition with its own asset report. If further updates are expected, the partitioning configuration should reserve space in the asset.
- Store the produced asset partitions and their asset reports to a defined location, A.
- Further, if the SceneComposer solution is updated (bitmaps changed, elements added etc.), generate the new asset and its report and store them to the desired location, B.
- Run asset partition updating process giving as input the location A (where the currently used asset partitions are stored) the configuration stored also at location A and the new asset stored at location B. Please note that by using this updating process the changes to the original asset partitions will be on a minimum and the size of the partitions will increase only if the existing reserved space is not enough for storing new or updated objects. Assetlib Shaper allows for iterative updates.

The next image shows the partitioning and partition updating workflows:



Command Line Arguments

The Assetlib Shaper, as part of the Asset Tool, is a command line tool providing the following options:

```
AssetTool -shape -h
```

Shows shaper help

```
AssetTool -shape -c
```

Shows shaper configuration file help

```
AssetTool -shape -k assetFilePath
```

Checks the format of the asset found at 'assetFilePath'

```
AssetTool -shape -it destinationfilePath assetFilePath
```

Generates asset info found at 'assetFilePath' to the text file at 'destinationFilePath'

```
AssetTool -shape -ix destinationfilePath assetFilePath
```

Generates asset info found at 'assetFilePath' to the Microsoft Excel file at 'destinationFilePath'

```
AssetTool -shape -p configurationFilePath assetFilePath
```

Partitions the asset found at 'assetFilePath' using the partitioning rules provided by the configuration file at 'configurationFilePath'

```
AssetTool -shape -l assetFilePath [LanguageCode:CompiledLanguagePackSourceFilePath]+
```

Adds or updates compiled language pack sources as language entries with LanguageCode name in the asset

```
AssetTool -shape -r assetFilePath TypeOfObjectToRemove:NameOfObjectToRemove
```

Removes the object with type 'TypeOfObjectToRemove' (or all if *) and name 'NameOfObjectToRemove' (* are accepted) from the asset

```
AssetTool -shape -st blockSize assetFilePath destinationFilePath
```

Generates the data overview at destinationFilePath (text file) of the asset found at assetFilePath using the provided blockSize

```
AssetTool -shape -sx blockSize assetFilePath destinationFilePath
```

Generates the data overview at destinationFilePath (Excel file) of the asset found at assetFilePath using the provided blockSize

```
AssetTool -shape -u currentAssetPartitionsDirPath newAssetFilePath configurationFilePath
```

Updates the partitions of the current asset with the new/updated/removed elements of the new asset with as little changes as possible.

Configuration File

The partitioning is done based on a provided configuration xml file. This file defines the partitions and the rules to be used in the partitioning process.

Configuration file defines:

- general properties of the partitions that will be created (location, block size, space reservation, compression)
- particular properties of each specific partition (name, space reservation, compression)

- assignment rules (the target partition for asset entries)

Configuration File Format

The format of the configuration file is the one shown next:

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory="Destination-Directory-Path" [block-size="Number-KB"]
                              [reserve-space-render-shader="Number-Units"] [reserve-space-names
                              [reserve-space-tree="Number-Units"] [reserve-space-entry="Number-
                                                    [reserve-space-lib="Number-Units"] [reser
        [<compression>
            [<lib type="Asset-Lib-Type"/>]+
        </compression>]
        [<partition name="Partition-Name" [reserve-space-render-shader="Number-Units"]
                      [reserve-space-names="Number-Units"] [reserve-space-tree="Number-Units"]
                                          [master="true or false"] [reserve-space-entry="Number-
                                          [reserve-space-lib="Number-Units"] [reserve-space-asse
            [<compression>
                [<lib type="Asset-Lib-Type"/>]+
            </compression>]
            [<rule type="Asset-Entry-Type" [alias="Rule-Alias"] [item="Asset-Entry-Name"] [exclu
           [reserve-space-tree="Number-Units"] [reserve-space-entry="Number-Units"] [reserve-spa
           [compress="true or false"] [case-sensitive="true or false"] [exclude-from-partition="
        [&lt;include-rules&gt;
            [<include-rule alias="Other-Rule-Alias"/>]*
            </include-rules>]
            [&lt;exclude-rules&gt;
                [<exclude-rule alias="Other-Rule-Alias"/>]*
            </exclude-rules>]
            [<include-items>
                [<include-item item="Asset-Entry-Name"/>]*
            </include-items>]
            [<exclude-items>
                [<exclude-item item="Asset-Entry-Name"/>]*
            </exclude-items>]
        </rule>]*
        </partition>]*
    </partitions>
</shaper-configuration>
```

Some of the attributes indicated during their presentation accept wildcards. Working with wildcards involves the usage of the * character as a substitute for any number of characters.

Each tag and attribute is presented next:

`<shaper-configuration>`	This is root tag of the xml configuration file. Attributes: • **xmlns** (mandatory): its only value is "http://tempuri.org/ConfigSchema.xsd", having the purpose of using an XSD schema for validating the configuration file.
`<partitions>`	This tag appears only once and, through its attributes, it defines the general properties of the outputted partitions. Attributes: • **directory** (mandatory): path of the destination directory of the produced partitions; this path can be relative (e.g., temp/partitions) or absolute (e.g., C:/temp); the location of the asset file is considered the base directory for a relative destination directory path (e.g., for destination directory temp/partitions and asset file location C:/work, the partitions are created at C:/work/temp/partitions). • **block-size** (optional): size of an individual storage block; it's value represents the block size in KB (number followed by the "KB" suffix). • **reserve-space-render-shader** (optional): amount of memory (in KB or %) reserved at the end of the render-shader region for all partitions. • **reserve-space-names** (optional): amount of memory (in KB or %) reserved at the end of the name table for all partitions. • **reserve-space-tree** (optional): amount of memory (in KB or %) at the end of the look-up tree for each lib and for all partitions. • **reserve-space-entry** (optional): amount of memory (in KB or %) reserved at the end of the data block of each asset entry for all partitions. • **reserve-space-lib** (optional): amount of memory (in KB or %) reserved at the end of the data block of each asset lib for all partitions. • **reserve-space-asset** (optional): amount of memory (in KB or %) reserved at the end of the data block of each partition.
`<compression>`	This tag refers to the compression of partitions' data. If this tag has **<partitions>** as parent tag, the compression rules defined under it apply to all partitions outputted by the partitioning process. Otherwise, if this tag has **<partition>** as parent tag, the compression rules defined under it apply to the particular partition addressed by those rules. This tag is optional and has no attributes but **<lib>** children tags (at least one if **<compression>** tag is present). Only Bitmap, ShaderProgram, VertexBuffer, RawResource and Font entries can be compressed.

`<lib>`	This tag is expected under the **<compression>** tag and defines a compression rule: an asset lib type to which compression is applied. Attributes: • **type** (mandatory): the asset lib type (e.g., Bitmap) to which compression is applied; besides the lib type names (available in Annex 1), the user can provide * as value for this attribute, representing "all lib types".
`<partition>`	This tag refers to a specific partition and defines the particular properties of that partition through its attributes. Attributes: • **name** (mandatory): name of the partition (e.g., Bitmaps), forms the file path to which the partition will be written along with the directory attribute of the <partitions> tag (e.g. C:/temp/Bitmaps.bin). • **master** (optional, but required to be true for one partition): specifies if the partition is a master partition or not; please note that all the asset entries that are not targeted by any rule will be included in the master partition. • **reserve-space-render-shader** (optional): amount of memory (in KB or %) reserved at the end of the render-shader region of the partition. • **reserve-space-names** (optional): amount of memory (in KB or %) reserved at the end of the name table of the partition. • **reserve-space-tree** (optional): amount of memory (in KB or %) reserved at the end of the look-up tree for all the libraries of the partition. • **reserve-space-entry** (optional): amount of memory (in KB or %) reserved at the end of the data block of each asset entry of the partition. • **reserve-space-lib** (optional): amount of memory (in KB or %) reserved at the end of the data block of each asset lib of the partition. • **reserve-space-asset** (optional): amount of memory (in KB or %) reserved at the end of the data block of the partition.

`<rule>`	This tag regards a partitioning rule and is in a many-to-one relation with the partition tags. It has several attributes. Attributes: • **type** (mandatory): type of the asset library to which the rule applies. It can be * (representing all types) or a specific asset entry type (e.g., VertexBuffer). For a complete list of supported types, please consult Annex 1. • **alias** (optional): name associated to a rule that can be used for referencing the rule (see include-rule or the include-rules tag). The alias is not required to be unique per rule. • **item** (optional): name pattern of the target asset entries, this attribute accepting wildcards. If this attribute is missing and no <include-items> and <include-rules> tags are defined, all the asset entries of the rule type (see the type attribute of the <rule> tag) are included by the rule. If this attribute is missing and at least one of the <include-items> and <include-rules> tags are present, this attribute has no implicit meaning (it is ignored, being undefined). If this attribute is present and the <include-items> tag is also present, then the value of the item attribute is considered as part of the defined elements of the <include-items> tag (actually, it is implicitly appended to the included items list). In this context, please note that defining a <include-items> tag containing one <include-item> is equivalent to providing the attribute item to the rule. • **exclude** (optional): name pattern of the asset entries that are excluded by the rule, this attribute also accepting wildcards. If this attribute is missing, this attribute has no implicit meaning (it is ignored, being undefined - nothing is excluded). If this attribute is present and the <exclude-items> tag is also present, then the value of the exclude attribute is considered as part of the defined elements of the <exclude-items> tag (actually, it is implicitly appended to the excluded items list). In this context, please note that defining a <exclude-items> tag containing one <exclude-item> is equivalent to providing the attribute exclude to the rule. • **case-sensitive** (optional): the name patterns in this rule are case-sensitive (true or false; false if missing). • **exclude-from-partition** (optional): the asset entries matched by this rule are excluded from this rule's partition (true or false; false if missing). • **compress** (optional): true if the rule compresses the target asset entries or false otherwise. • **reserve-space-tree** (optional): amount of memory (in KB or %) reserved at the end of the look-up tree for the asset libraries targeted by the current rule. • **reserve-space-entry** (optional): amount of memory (in KB or %) reserved at the end of the

`<include-items>`	This tag appears no more than once under a <rule> tag and regards the definition of the asset entry name patterns of the entries that are target of the rule. This tag is optional and has no attributes, only children <include-item> tags.
`<include-item>`	This tag appears at least once under an <include-items> tag of a rule and regards the definition of one asset entry name pattern that refer to entries that are target of the rule. This tag has one mandatory attribute. Attribute: • **item** (mandatory): the name pattern of the target asset entries, this attribute accepting wildcards.
`<exclude-items>`	This tag appears no more than once under a rule tag and regards the definition of the asset entry name patterns of the entries that are excluded by the rule. This tag is optional and has no attributes, only children <exclude-item> tags.
`<exclude-item>`	This tag appears at least once under an <exclude-items> tag of a rule and regards the definition of one asset entry name pattern that refer to entries that are excluded by the rule. This tag has one mandatory attribute. Attribute: • **item** (mandatory): the name pattern of the asset entries that are excluded, this attribute accepting wildcards.

`<include-rules>`	Besides defining its own included items, a rule can include the items filtered (included minus excluded) directly by other rules, directly meaning using <include-items> and <exclude-items> tags. These rules are referenced by their alias (an alias should be assigned to rules as needed). Please note that every rule has its defined type (see type attribute of the <rule> tag). Including a rule defined for a different type by using the <include-rules> tag takes into consideration the asset entry name patterns defined through <include-items> and <exclude-items> tags and applies them to the items of the container rule type, the type of the included rule being ignored for flexibility. For example, let's consider 3 rules: alias1, alias2 and alias3. Let's say the alias1 rule includes through <include-items> tag all the asset entries having their name starting with "A" and excludes through <exclude-items> tag all the asset entries having their names ending in "A". The alias2 rule includes through <include-items> tag all the asset entries having their name starting with "B" and excludes through <exclude-items> tag all the asset entries having their names ending in "B". Besides these, the alias2 rule includes the alias1 rule through <include-rules> tag. The alias 3 rule includes through <include-items> tag all the asset entries having their name starting with "C" and excludes through <exclude-items> tag all the asset entries having their names ending in "C". Besides these, the alias3 rule includes the alias2 rule through <include-rules> tag. In this scenario, the asset2 rule includes all the asset entries having their name starting with "A" or "B" and excludes all the entries with name ending in "A" or "B". On the other hand, the asset3 rule includes all the asset entries having their name starting with "B" or "C" and excludes all the entries with name ending in "B" or "C". Please note that the <include-rule> of the included rule is not considered in the container rule. Only the direct definition of items through <include-items> and <exclude-items> are taken into account. The <include-rules> tag appears no more than once under a rule tag and regards the definition of the rule alias patterns of the rules to be include in our rule. This tag is optional and has no attributes, only children <include-rule> tags.
`<include-rule>`	This tag appears at least once under an <include-rules> tag of a rule and regards the definition of one rule alias pattern that refer to rules that are included in our rule. This tag has one mandatory attribute. Attribute: • **alias** (mandatory): the rule alias pattern of the rules to be included in our rule, this attribute accepting wildcards.

`<exclude-rules>`	Besides defining its own excluded items, a rule can exclude the items filtered (included minus excluded) directly by other rules, directly meaning using <include-items> and <exclude-items> tags. These rules are referenced by their alias (an alias should be assigned to rules as needed). Excluding a rule defined for a different type by using the <exclude-rules> tag takes into consideration the asset entry name patterns defined through <include-items> and <exclude-items> tags and applies them to the items of the container rule type, the type of the included rule being ignored for flexibility. For example, let's consider 3 rules: alias1, alias2 and alias3. Let's say the alias1 rule includes through <include-items> tag all the asset entries having their name starting with "A" and excludes through <exclude-items> tag all the asset entries having their names ending in "A". The alias2 rule includes through <include-items> tag all the asset entries having their name starting with "A" or "B" and excludes through <exclude-items> tag all the asset entries having their names ending in "A" or "B". Besides these, the alias2 rule excludes the alias1 rule through <exclude-rules> tag. The alias 3 rule includes through <include-items> tag all the asset entries having their name starting with "A", "B" or "C" and excludes through <exclude-items> tag all the asset entries having their names ending in "A", "B" or "C". Besides these, the alias3 rule excludes the alias2 rule through <exclude-rules> tag. In this scenario, the asset2 rule includes all the asset entries having their name starting with "A" or "B" and those ending in "A" (from the excluded rule) and excludes all the entries with name ending in "A" or "B" and starting with A (from the excluded rule). Simplifying, the alias2 rule includes the entries with name starting with "B" and excludes those with name ending in "B". On the other hand, the asset3 rule includes all the asset entries having their name starting with "A" or "C" and excludes all the entries with name ending in "A" or "C". Please note that the <exclude-rule> of the excluded rule is not considered in the container rule. Only the direct definition of excludes through <exclude-items> and <include-items> are taken into account. The <exclude-rules> tag appears no more than once under a rule tag and regards the definition of the rule alias patterns of the rules that are excluded in our rule. This tag is optional and has no attributes, only children <exclude-rule> tags.
`<exclude-rule>`	This tag appears at least once under an <exclude-rules> tag of a rule and regards the definition of one rule alias pattern that refer to rules that are excluded by our rule. This tag has one mandatory attribute. Attribute: • **alias** (mandatory): the rule alias pattern of the rules that are excluded by our rule, this attribute accepting wildcards.

Simple Configuration File Example

Consider the next partitioning configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory=".">
        <partition name="Master" master="true">
            <compression>
                <lib type="Bitmap"/>
            </compression>
        </partition>
    </partitions>
</shaper-configuration>
```

To partition an asset, asset.bin, using the partitioning configuration, config.xml, run this command:

```
AssetTool -shape -p config.xml asset.bin
```

Using the configuration above, the partitioning tool will create one partition (asset) named Master in the current directory.This partition will contain all elements present in the original asset, the Bitmap elements being compressed.

Complex Configuration File Example

Consider the next complex partitioning configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory="C:/temp" block-size="3KB" reserve-space-names="20%" reserve-space-tre
        <compression>
            <lib type="Bitmap"/>
            <lib type="VertexBuffer"/>
        </compression>
        <partition name="Scenes" master="true" reserve-space-asset="256KB">
        </partition>
        <partition name="BitmapsAndFonts" reserve-space-lib="10%">
            <compression>
                <lib type="Font"/>
            </compression>
            <rule alias="RuleOne" type="Font">
                <include-items>
                    <include-item item="*one*"/>
                <include-item item="Anim*"/>
                </include-items>
                <exclude-items>
```

```
                    <exclude-item item="*two*"/>
                </exclude-items>
            </rule>
            <rule alias="RuleN" type="Font">
                <include-items>
                        <include-item item="*end"/>
                </include-items>
                <exclude-items>
                    <exclude-item item="Start*"/>
                </exclude-items>
            </rule>
            <rule type="Bitmap" item="*" reserve-space-lib="20%"/>
            <rule type="Bitmap" item="*BMP*" case-sensitive="true" exclude-from-partition="true"
        </partition>
        <partition name="VertexBuffers" reserve-space-tree="10%">
            <rule type="VertexBuffer" item="Vert*">
        <include-items>
                <include-item item="FirstVert*"/>
        </include-items>
        <include-rules>
            <include-rule alias="RuleOne"/>
            </include-rules>
            <exclude-rules>
                <exclude-rule alias="RuleN"/>
            </exclude-rules>
        </rule>
        </partition>
    </partitions>
</shaper-configuration>
```

Using the above configuration, the partitioning tool will create three partitions (assets) with the names: Scenes, BitmapsAndFonts and VertexBuffers, that will be found at C:/temp. The block size is 3KB, meaning that a data item will start only at the beginning of a new 3KB storage block.

The partition named Scenes is the master partition, containing any element of the original asset that is not contained in the other two partitions. In this partition, 20% of the name table space will be reserved at the end of the name table, 4KB will be reserved at the end of each asset lib look-up tree, and 256KB will be reserved at the end of the asset's data block. If there are VertexBuffer entries in this first partition, they'll be compressed (if there were Bitmap entries, they would be compressed two, but there is none since all go into the BitmapsAndFonts partition).

The partition named BitmapsAndFonts will contain all fonts of which names start with "Anim" or "Start" or contain "one" or end with "end" (wildcards is used) excluding all the entries with names that contain "two". Also, this partition includes all the bitmaps of the original asset, except for those that contain "BMP" in their name, case-sensitive, which are excluded from this partition and, without other rules to target them, are included in the master partition. In this partition, 20% of the name table space will be reserved at the end of the name table, 4KB will be reserved at the end of each asset lib look-up tree, and 10% of the total lib data block will be reserved at the end of each lib's data block, except for the Bitmaps lib for which 20% will be reserved. Compression is applied to all Bitmap elements (because of the global compression rule involving this type of elements) and

to all Font elements (because of the local partition compression rule involving those elements).

Finally the last partition, VertexBuffers, will contain all the VertexBuffer entries of the original asset that have names that start with "Vert" (direct item definition - equivalent to defining an <include-item>) or "FirstVert" or contain "one" or start with "Anim" except all the entries with names that contain "two" (see <include-rule> definition). Moreover, this partition will contain all the VertexBuffer entries with names that start with "Start", being excluded from the partition all those with names that end in "end" (see <exclude-rule> definition). In this partition, 20% of the name table space will be reserved at the end of the name table and 10% of the look-up tree space will be reserved at the end of each asset lib look-up tree. Compression is applied to all VertexBuffer elements (see local compression rule for this third partition).

In conclusion, in what concerns the partitioning rules, the more specific definitions override the more general ones. Regarding the compression rules, if there are compression rules defined globally (under partitions tag), they are inherited to all the assets outputted by the partitioning process.

Shaper Pitfalls

This section refers to situations where the partitioning behaviour isn't obvious. The aim of this section is to clarify the algorithm used by Assetlib Shaper in choosing the partitioning rule to apply to a particular asset object.

Let's take for example the following partitioning configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<shaper-configuration xmlns="http://tempuri.org/ConfigSchema.xsd">
    <partitions directory="C:/temp">
        <compression>
             <lib type="Font"/>
        </compression>
        <partition name="Master" master="true">
        </partition>
        <partition name="Fonts">
             <rule type="Font" item="*"></rule>
             <rule type="VertexBuffer" item="abc*"/>
             <rule type="Font" item="*" compress="false"/>
        </partition>
    </partitions>
</shaper-configuration>
```

By using this configuration two partitions are produced; Master and Fonts. The question is how Font entries are kept in the Fonts partition, compressed or uncompressed. There are two rules in the configuration for the Fonts partition referring to all Font entries (type="Font" and item="*"): 1. Doesn't overwrite the compression rule defined globally for the Font entries (it will compress all the Font entries) 2. Has the compress flag set to false, overwriting the globally defined compression rule (it won't compress the Font entries)

Note: from all the rules for a partition in a configuration file that regards a partition there's only one selected to be used for a particular entry. If there are more rules targeting the same entry (in our example, all the Font entries), the first criteria used in selecting the rule to apply is specificity. If one rule has type="Font" and item="*" and some other rule type="Font" and item="a*", the second rule is more specific than the first one, applying to a subset of all the entries (the entries that have a name that start with "a"). So the more specific rule wins.

Both rules in the provided sample configuration file, referring to Font entries, have the same specificity. Both apply to all Font entries (item="*"). In such a situation, the AssetLib Shaper always selects the first rule.

Summarizing criteria:

- more specific rule wins;
- equal specificity; the first rule defined in the configuration file wins.

Using the given rule selection criteria and considering the sample configuration file, the Font entries are compressed in the Fonts partition.

Terminology

Terms used by the Assetlib Shaper documentation:

- **Asset**: file with .bin extension produced by SceneComposer.
- **Asset Report**: file with .xml extension attached to an asset, having the same name as the asset file; also produced by SceneComposer.
- **Partitioning (or Shaping)**: process in which an asset is broken down into several assets, each containing a part of the original asset, based on a set of rules.
- **Asset Partition (or Partition)**: asset resulting from a partitioning process, containing a part of the original asset used in the process.
- **Asset Library**: specific part of asset content; library of objects of the same type or purpose.
- **Asset Library Entry (or Asset Entry)**: specific object in the asset content.
- **Partitioning Configuration (or Configuration)**: .xml file containing the set of rules to be used in the partitioning process.
- **Master Partition**: partition that implicitly includes all the Asset Entries that are not referred by rules; each Partitioning Configuration defines exactly one Master Partition.
- **Asset Partition Updating**: process of updating the partitions of an original asset to the content of a new asset, which is an evolution of the original asset.

Append Languages

Description

Language Appender, as part of the Asset Tool, offers functionality for working with language packs in the context of assets generated with SceneComposer. To use this tool, run AssetTool -append.

For more information about this optional feature refer to the Candera Application Development Tutorial Globalization Support.

Usage

For appending language packs to an asset library file, use the following workflow:

- Generate asset including translatable texts from SceneComposer
- Run `AssetTool -append -a` to append a language pack .lps file to the asset file.

> Globalization license is required for generating assets including translatable texts and using the Globalization feature in Candera applications.

Command line options:

Append the asset at 'inputAssetFilePath' with the compiled language packs from 'languagePackSourcePath' into a new asset at 'outputAssetFilePath':

```
AssetTool -append -a inputAssetFilePath outputAssetFilePath languagePackSourcePath
```

Remove the language pack with code 'languageCode' from the asset at 'inputAssetFilePath' and created a new asset at 'outputAssetFilePath':

```
AssetTool -append -r inputAssetFilePath outputAssetFilePath languageCode
```

Show help:

```
AssetTool -append -h
```

Example

Following is an example how to append language packs to an asset:

1. Input asset: MasterLanguagePack_Asset_Host.bin
2. Output asset: MultiLanguagePack_Asset_Host.bin
3. Language Pack source: MultiLanguagePack_Source.lps

```
AssetTool -append -a MasterLanguagePack_Asset_Host.bin MultiLanguagePack_Asset_Host.bin MultiLar
```

Compare Asset Libraries

Description

The CGI Studio **Asset Library Verification** license supports the user in managing asset files in larger environments or projects with multiple variants. It provides means for verification of matching asset library / widget / application versions and allows to assign assets libraries to their originating solution.

For more information about this optional feature refer to the Candera Application Development Tutorial Asset Library Verification (Optional) .

Usage

For comparing asset library files, use the following workflow:

- Obtain the two assets for comparison (e.g., generate assets from SceneComposer);
- Run `AssetTool -diff` on these two assets, generating an Excel file with diff info; this information contains the versions of the two assets, their name table entries and all the asset lib entries grouped on libraries.

> Asset Library Verification license is required for comparing the complete set of asset version information.

Command line options:

Create the diff between left and right asset files and write it as Excel file on outputFilePath:

```
AssetTool -diff -d leftAssetFilePath rightAssetFilePath outputFilePath
```

Show help:

```
AssetTool -diff -h
```

Extract Bitmaps

Description

Use the Bitmap Extractor component of Asset Tool for extracting bitmaps from a given asset generated with SceneComposer.

Usage

For extracting bitmaps from an asset library file, use the following workflow:

- Obtain the asset for extracting the bitmaps (e.g., generate asset from SceneComposer);
- Run `AssetTool -extract` on this asset;
- at the path given in the command, the structure of folders containing the bitmaps found in the asset is created.

Command line options:

Extract all bitmaps from the given asset to the output path:

```
AssetTool -extract -e assetFilePath outputPath
```

Show help:

```
AssetTool -extract -h
```

Asset Tool Change Log 3.0.0

Combined Asset Tool

In CGI-Studio V3.0.0, the following previous tools related to asset library post-processing have been combined into a single application called `AssetTool.exe`:

- CgiStudioAssetDiff
- CgiStudioAssetLibShaper
- CgiStudioLanguageAppender
- CgiStudioAssetExtractor

Asset Tool provides the following commands:

`-h`	Shows help.
`-v`	Shows version.
`-shape [Parameters]`	Runs AssetLibShaper with the given parameters. See Shape Asset Library.
`-append [Parameters]`	Runs LanguageAppender with the given parameters. See Append Languages.
`-diff [Parameters]`	Runs AssetDiff with the given parameters. See Compare Asset Libraries.
`-extract [Parameters]`	Runs AssetExtractor with the given parameters. See Extract Bitmaps.

Asset Format Changes

In CGI-Studio V3.0.0, the format of the asset changed:

- The header of each library contains a new list structure which replaces the old lookup tree.
- The data part of the library elements contain new structures. Among these are the data regions.

These changes slightly affect the partitioning process, for which the Assetlib Shaper component tool is responsible. Regarding this process, no matter the asset format changes, the partitioning configuration offers the same options with two remarks:

Space Reservation Rules

The space reservation rules used for reserving space at lookup tree end, "reserve-space-tree", are now used, with no change, for reserving space at the end of the list that replaces the lookup tree.

Compression

Compression can now be applied only to library elements that contain data regions. There is only a subset of all the asset libraries that contain elements with data regions. The library types that allow compression are:

- Bitmap
- ShaderProgram
- VertexBuffer
- RawResource
- Font.

If a partitioning configuration specifies other library element types for compression, then a warning is shown in the console and no compression is applied to these elements.

Supported Asset Library Types

Also relating to the shaping feature, the asset library types are updated.

The library types that are currently available in assets are: `Animation, AnimationGroup, Bitmap, CameraGroup, ClearMode, Display, Font, RawResource, ReferencedTemplate, RenderTarget, Scene2D, Scene, ShaderProgram, Composite2D, Composite, TextStyle, Theme, Transition, VertexBuffer, AppearanceCollection, Appearance, Material, RenderMode, UniformSetter, Texture, and Language.`