USD Concepts

•••

Part 1

Basic Scene Description

A ***prim*** is equivalent to a DAG Node

There are 3 types of prim:

- A ***def*** defines a new prim
- An ***over*** overrides an existing prim
- A ***class*** is an abstract prim that can be inherited from

A namespace is just the full path to a prim (e.g A|B|C)

Layer

A ***layer*** is mostly...just a USD file, and contains a load of prims

They are identified by their file path or asset identifier (e.g in our case a URI e.g ark:/job(gotham)/objecttype(setpiece)/object(chair)/objvar(aeron)/geo(alembic)

Layers can also be anonymous in-memory layers.

A layer can contain not just a tree, but a forest of prims, but will usually specify a default prim for clients to use

Sublayers

A layer can have ***sublayers***. Those layers can have sublayers, and so on. This is called the ***layerStack***.

A sublayer behaves a bit like an include file in C/C++, or a “from module import *” in python - it just adds prims to the current namespace.

Sublayering is the way to do things like per-dept overrides

See glossary

References

Prims can reference:

- Layers by asset identifier
- Namespace locations by Prim Path

Referencing is the way we do aggregation (e.g setpiece-section-set) and build shots e.g chars/props/sets/cameras etc

References encapsulate the scene description behind the reference: the reference brings in a monolithic unit, whose contents can be overridden, but not its structure

Payloads are a special case of referencing that allow deferred loading

References (but not payloads) can be ***list edited***

Variants and VariantSets

A bit like a Switch statement.

E.g “SurfacingVariation” would be a VariantSet with variants:

- Dry
- Wet
- muddy

Can be nested

Any prim can contain N variantSets

Must have a default. Only one can be chosen

Instance

If we mark parts of the hierarchy as “instanceable”, and USD will create a “master” the first time it sees the flag

Any time it sees a reference to the same chunk of hierarchy marked as instanceable, it will instance it

No overrides will be applied to instances

Hierarchy traversal will stop at instances

See instancing

Relationship

...between a Prim and a Property on another Prim

Not a live evaluation mechanism like Maya connections or expressions

USD will ensure all composition arcs are loaded if you follow a relationship target

Stage

A stage is where a root layer and all of it's references, sublayers etc are ***composed*** together and "baked" in memory to give a single "scene description" that we can work with (e.g traverse).

Provides various convenience mechanisms e.g caching, session layer, flattening

A stage has a root layer, an optional session layer

Ordering Rules for Composition

Are quite complex and I don’t understand them fully yet

LIVRPS is the mnemonic for remembering the basic ordering:

- Local (Local Layerstack)
- Inherits
- Variant
- Reference
- Payload
- Specialises

See glossary

Mechanics of Composition

- Composition Arcs (the 6 operators SIVRPS)
- Prim Index (output of Composition)
- Prim Spec/Prim Stack (input of Composition)
- Opinion (each Prim Spec can contain an opinion)
- PCP (the API Layer that handles composition, see docs)

Edit Target

An edit target allows you to specify which layer your changes will affect

Analogy: which Photoshop layer you select

Currently can choose as your Edit Target:

- Any sublayer in a stage's root layerStack
- currently selected variant of any defined top-level variantSet

Defaults to session Layer