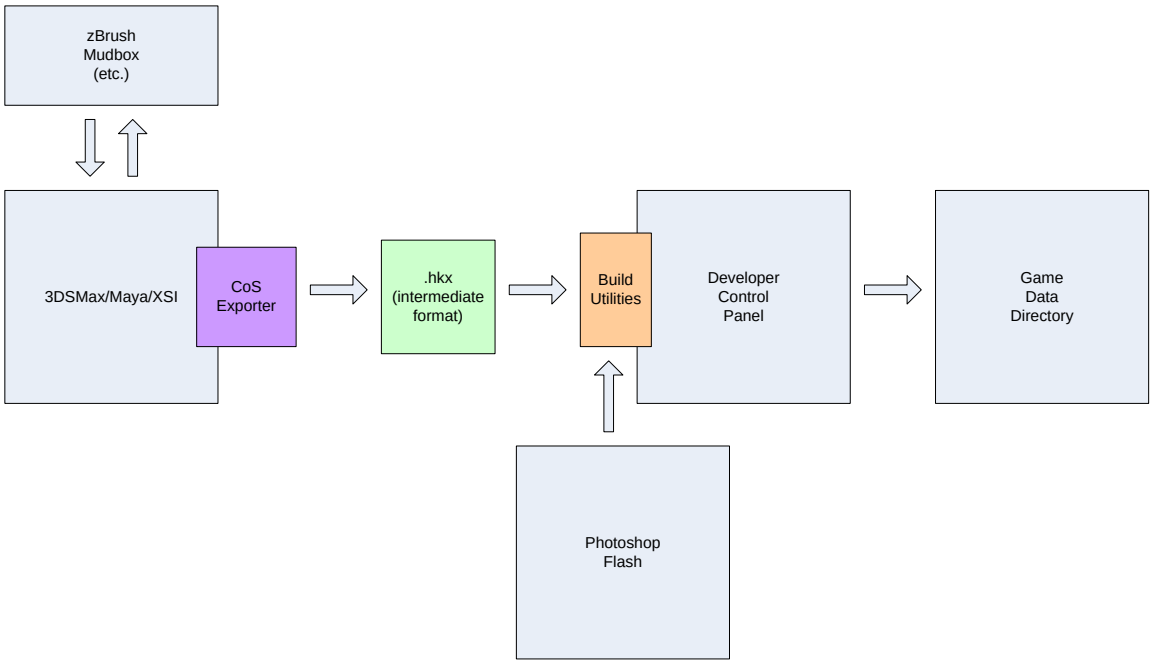
The CoS Development Environment

The CoS development environment is designed, from the artists' point of view, for minimum distraction and maximum productivity. Only the pieces of the pipeline that are needed for creating art assets are present in the artists' setup. This consists primarily of the game executable(s) and the tools required to create game-ready data.

Tools

The primary tools used by an artist are not provided by the CoS team – these usually include 3D content creation tools such as Autodesk 3DSMax/Maya/XSI, zBrush, Mudbox, and so on, as well as 2D content creation tools such as Adobe Photoshop and Flash.

Content flows through the CoS pipeline as shown (at a high level) in the following diagram:



The most important thing to notice about the above diagram is that all game-ready data is produced through the *Developer Control Panel*.

Developer Control Panel

The Developer Control Panel, from the perspective of the artist, is a game data build automation tool. It exists in the CoS directory structure and runs as a

system-tray-resident application. It is a plugin-based tool, and from the artist's perspective, the most important plugin is the Build Manager. This tool is responsible for monitoring changes to the export data directory and "compiling" data files into game-ready format as the export data changes. This means that when you export, for example, a mesh model from Maya, into the export directory, then so long as the Developer Control Panel application is running, that exported data will be conditioned into the proper runtime format and placed into the proper runtime data directory.

Directory Structure

From the artists' perspective, there are only three directories of interest (and assuming you have used the relevant convenience checkout script when you first set up your workspace, these are all you will see):

"bin" Directory

This is where you will find the latest game executable(s) as well as all of the content creation and conditioning tools (including the Developer Control Panel). There is a "data" subdirectory under "bin" for game-ready data, but this is **not** where you put exported data.

"data" Directory

This is where you will put data exported or saved from your content creation tools (Max, Maya, Photoshop, Flash, etc). The Developer Control Panel's Build Manager plugin will see to compiling the data into game-ready form and placing it into the proper location under bin/data.

"artsource" Directory

This is where you put your original source art files – Maya (.ma or .mb), Max (.max), Photoshop (.psd), Flash (.fla), and so on. Do **not** "version" your files with different names for different revisions of a given asset – that is what source control is for. If you are iterating on an asset and would like to keep several copies at hand locally, that is fine, but "artsource" should only ever contain a single (source-control-versioned) file for a given asset.

Checking Out, Updating, and Checking In

Initial Checkout and Workspace Setup

In order to ensure the smoothest experience while working on CoS content, it is important to set up your directory structure properly at the outset (meaning, when you first set up your workspace). The simplest way to do this is to run the `Artist.bat` batch file, which will set up your source controlled workspace with the directories mentioned above. Simply create (on your local computer) the root directory where you would like your workspace to live, copy the

`Artist.bat` file into that directory, and double-click it to run the initial checkout. This will create 3 directories in your workspace root: `bin`, `data` and `artsource`, and populate them from source control, with the latest available version of all of the files in those directories (and their subdirectories). You only need to run Artist.bat one time for a given workspace – from that point, you will be doing source control "updates" (also known as "syncs") and "checkins" (also known as "commits" or "submits").

Updating To The Latest Revision

From time to time you will need to get in sync with what others have checked in. This could be everything in your workspace; it could simply be a new game build checked in by an engineer; it could simply be new tools. Usually, however, unless you have a reason not to do so, you will typically want to do a full update.

When you did your initial checkout, you will have received in your top-level workspace directory, another batch file called `UpdateWorkspace.bat`. Due to the way that the source control directories are set up, if you want to update your workspace, simply double-click this file to update to the latest version from source control.

Note that since someone else may have changed and checked in a different version of a file that you also have changed, you may encounter what is known as a "conflict". Source-control conflicts need to be resolved as early as possible. Resolving them is another matter, and is handled on a case-by-case basis. For example, if one artist changed a Flash .fla file and checked it in, and you are working on the same Flash scene, you can either accept "their" changes (which will delete any changes you have made to the Flash file, unless you made a copy of it under another name), or you can accept "your" changes, which will ignore their changes. Which one you choose depends greatly on communication – ideally, you and the other artist will have been on contact and you will know whose changes are the "right" ones to accept. **When in doubt, do not either (a) "stomp on" unknown changes, nor (b) ignore the conflict**. Doing either will invariably cause problems farther down the line, when they are more difficult to figure out.

Checking In Your Changed Files

When you are iterating on content creation, typically you will follow a standard process similar to the following:

1. Make a change or set of changes to a file or files
2. Export or save the changed intermediate data to the "export" data directory (the "data" directory in your workspace root)
3. Run the game to see if the changes had the desired effect
4. Repeat 1-3 as necessary
5. When satisfied with your changes, check them into source control

This last step is often where confusion occurs. Generally, when making changes to an art asset, you are performing this work in a tool such as Max or Maya, which produces its own "source art" files (.ma/.mb for Maya, .max for 3DSMax, etc.). While iterating on changes, you will also have produced exported data (using the CoS exporter) into your workspace "data" directory. For reasons of source-control coherence, it is important that

a) You check in all files together, that "go together". This means that if you were iterating on several animations for a given character, check in all of the .hkx animation files in the same source-control operation. If you were also making changes, say, to skin bindings, you would want to check in the changed .hkx mesh files as well (and likewise for the rig, if that changed, and so on). The point is to be able to track related changes together, so that later on, it is much less difficult to identify which art assets changed together if a bad export or data build occurs.
b) Likewise, it is important for the project to be able to trace exported data to its source form, which means that you also need to check in the source art (into "artsource") that goes with the exported data. This means that for the animations mentioned in (a), you would also want to check in the Maya/Max/XSI file that produced those animation exports **in the same source control operation as the exported data files**. It is the part in bold that sometimes causes confusion – often, an artist will check in their exports, and then in a later source control operation, their source art, and the connection between the source art and the exports is lost (which can cause difficulty when trying to track down potential art issues).

Understand that there is no "cost" to checking in incremental versions of a file. In other words, if you are working on a character, you do not need to wait until everything is "perfect" to check in the source art. In fact, when checking in exported data, you **must** check in the source art as well (if the source art changed – there are cases where data is re-exported but the source art file did not change in any way; in this case, most source-control systems will not let you check in the file because it is identical to the current latest version).

Creating 3D Art

The first step is to know what you want to make. Obvious, but needs to be said. The following applies to *all* 3D models, regardless whether they are animated or not (indeed, at some future point, we may want to employ physical destruction of otherwise static models, for example).

High-Poly (HP)

Assuming that you've got that part down, you'll want to make a high-poly version of your model. Shoot the moon -- poly count doesn't matter here because you are going to use this version for generating textures such as normal maps, ambient occlusion maps, or maybe you'll do a UV Unwrap on it to generate the diffuse texture -- regardless what you intend to use it for, this is where your high-res detail will come from. Feel free to use any tool you like (ZBrush, Mudbox, and so on) without regard for whether there is a Havok exporter for it, because it doesn't matter -- you are not going to be exporting it into the game art pipeline.

Low-Poly (LP)

This is the model that will be used in-game. However you arrive at it (decimating the HP, starting from scratch -- not recommended, btw, especially if you intend to use the HP and LP together to extract normal maps), you'll need to keep some things in mind, that will make life easier later in the process.

Batch Count vs. Poly Count

Understand that batch count (the number of "draw calls" it takes to render your character) is more important than raw poly count; 10,000 polygons in one batch will render faster than 10 batches of 1,000 polys each, even though the total poly count is the same. Understand as well that this does not mean you get to use 250k polys for your "low poly" model; use some common sense -- use as many polygons as you need to get the detail/fidelity/resolution you want (with all of the different maps applied), and no more. The game tools will help you understand the rendering performance you can expect from a given model, early enough in the process that you can tweak the level of detail in the base LP mesh.

Level of Detail (LoD) Meshes

TBD -- LoD system is not yet implemented or designed.

Submesh Naming Convention

TBD – naming convention for the meshes themselves probably not important

Interchangeable Weapons

(tbd – probably use the standard method of animating/skinning a weapon to a skeleton, but then the question is about same meshes on different skeletons – can we just tag a bone and skin to that in the runtime, or do we need to have different copies of the same weapon with different skin bindings? Also does this apply to infantry or will we have just the one – or two if we have male/female – skeletons?)

Guidelines

Keep the following in mind when modeling:

- Model in the "bind pose" -- this means the "rest" position for your model; for an infantry model, this is the classic "T" pose, for a MACRO, it is the basic "idle" posture (not "shut down" or any other pose), and so on. This makes rigging and animating the model MUCH easier later in the process.
- Model it in "parts" -- every type of animated model will need to have various defined parts on it for weapon customization/swapping, hitbox creation, and so on. The asset conditioning pipeline will take care of reducing the batch count on your model as much as possible, and if you model the parts such that they correspond to their various hitboxes, then generating the hitbox collision bodies will be much simplified later. For rigid-skinned models (MACRO, MICRO and other mechanical models), it also makes skinning the mesh (binding the various parts to their corresponding bones in the skeleton) greatly simplified as well, and your rigger (even if it is you) will thank you for it. How you define the parts, of course, will also depend on what parts/hitboxes are supported by the game runtime; there is no point (other than perhaps for ease of rigging) in having the arm broken into four parts if there is only going to be, say, two hitboxes on the arm.

Animation

Creating A Skeleton

- Minimal number of bones needed to get the job done

Mount Points

- Tag points for weapons mounting

Rigging/Skinning

- Max 4 bones per vertex (won't be a problem for any mechanical model, since they are all rigid skinned)
- Max 72 bones per submesh (will be hard to tell in art tools without a CoS plugin to help figure out how meshes will be reduced to minimal batches – or we can leave that in the artist's hands, probably best)

Animating

- Creating animations on a single timeline
- Export each animation individually
- Make frame 1 be the bind pose (for sanity)
- Animation length not an issue; bear in mind how it will play in game
- Define base set of required animations
- Artists can also create "flavor" animations, and use those in animation setup/blend tool, perhaps as part of transitions between base animations

Physics

Hitboxes

- Use Havok ragdolls for hitboxes
- Will need to describe the steps needed to
    - Create the shapes
    - Map the ragdoll and animation skeletons
    - Verify that the ragdoll is aligned and driven properly by the animations

Prop Collision Shapes

- Default to MOPP/trimesh for concave meshes (meshes you can go into/under, for example a hangar), convex hull for anything else (trees/signs, for example)

Exporting

Max Specifics

Maya Specifics

General Exporting

- Use the .hko config files, and start with (copy and customize) an existing one – these are set up by engineers to work properly with the needs of the runtime
- Export animations separately, will need to create a new config for each one
- Export mesh, skeleton (rig), ragdoll (or prop rigid body), all separately
- Run the CoS exporter and not the Havok one, due to the need to capture CoS material information
- Export to data directory, not to bin/data – highly recommended (required?) that the Developer Control Panel be running to automate builds
    - Also have this running when doing SVN updates –soon no built data will exist in the bin directory

Creating 2D Art

User Interface

- Flash
- How to author ActionScript to work with the runtime

Visual Effects (VFX, aka Particles)

- Hard to talk about this yet since none of it exists…

Audio

Sound Effects (SFX)

- Will need to design the art interface to audio, but it's largely going to be based on XAudio2, so therefore the "export" format is going to be 4-bit ADPCM .wav files.
- Explain the difference between streaming and samples

2D (Ambient) Audio

- Used primarily for music or environmental ambient noise/sound – streaming
- Depending on audio system load, can also be used for in-cockpit noises that are otherwise omnidirectional

3D (Positional) Audio

- Used for samples (almost never music or streaming audio) that need to have positional cues in the 3D world
-