Hero Lab® Online Export Format

Version 1.1 – November 23, 2020

Overview

Hero Lab Online (HLO) makes character creation and management a breeze. The Hero Lab Export Format enables users to extract those characters in a format that can then be readily incorporated into other digital tools, ranging from custom character sheet output to virtual tabletops.

Character export data can be obtained from HLO via multiple methods. Users are able to manually export their character at any time from within the product, either as raw data or a file. This can be accomplished via the Tools menu in the upper right of the product (the gear icon). Alternately, a public API is available that offers a way for external tools to directly retrieve character export data from HLO. Full details of this API are documented separately.

This particular document focuses on the structure of the export format and how it can be consumed by other tools.

NOTE! A revision history of behavioral changes will be found at the end of this document.

Export Basics

The export format uses the JSON standard (www.json.org) for encapsulating the details of a given character. The overall format is consistent across game systems. However, the specifics for each game system are distinct, just as the game systems themselves are unique.

The content of each export represents a single character created within HLO. If a tool wants to work with multiple characters, each will be exported separately and consumed independently.

Each character is comprised of one or more *actors*, where each actor can operate individually within the game. Many characters will only have a single actor, but some will have a second, and it is possible to have many. An example of a character with a second actor is a Druid with an animal companion or a Wizard with a familiar. A Mechanic might control multiple drones, in which case all of those drones would be separate actors accompanying the main character.

General Structure

The entire export is a single JSON object. Within this object, there are three main sections that describe the export: portfolio, metadata, and actors. Each of these sections has its own top-level property within the export object, correspondingly named. The actors are an array, while the portfolio and metadata are objects.

```
{
  “portfolio”: {},
  “metadata”: {},
  “actors” : []
}
```

Portfolio Section

The portfolio section provides basic details about the character and the export data returned. The “portfolio” object is always included and consists of the following properties:

charId	Unique identifier for the character within HLO
version	Current version of the character within HLO. Whenever a character is modified in some way, its version is increased, and this value reflects the version at the time it was exported.

baseline	Baseline version of the character against which a differential export was generated. Character data can become large and only small portions may change. The export mechanism can optionally report only the differences between an older version and the current version. The baseline indicates the older version that the export contains the differences from. A baseline of zero indicates the full character is reported.

```
“portfolio”: {
  “charId”: “p8NtGLfq”,
  “version”: 10,
  “baseline” 0
}
```

Metadata Section

The metadata section outlines an assortment of important internal details about HLO that can impact the export contents. The “metadata” object is only included when a full export occurs (i.e. baseline equals zero) and is comprised of the following properties:

gameCode	Unique identifier for the game system within HLO
gameName	Friendly name of the game system to which the character applies
gameMajor	Major version number of the game system data files. Each game system has distinct data files with its own version. When the major version changes, the contents of an export are likely to change in some manner, so differential exports across major version changes should not be used.
gameMinor	Minor version number of the game system data files. When the minor version changes, differential exports should be safe, as this typically indicates smaller changes and bug fixes.
hloVersion	Version of the HLO engine. Changes to the engine should not typically impact the export data, but this value could be helpful in isolating bugs.
exportVersion	Version of the export format itself. When this changes, there could be implications across the export format.
legal	Legal text governing the exported data

```
“metadata”: {
  “gameCode”: “Starfinder”,
  “gameName”: “Starfinder Roleplaying Game”,
  “gameMajor”: 3,
  “gameMinor”: 8,
  “hloVersion”: 989,
  “exportVersion”: 1
}
```

Actors Section

The actors section is the meat of the export format, as it contains all of the actual character information. The “actors” object is always present and holds one or more properties that identify the various individual actors that comprise the overall character. If a character is a simple Fighter, then there will be a single actor, but more complex characters may have minions that are also included as separate actors (e.g. animal companions, drones, etc.).

Each actor is identified by a unique value that never changes. The primary character is always assigned a value of one (and sometimes referred to as the *lead actor*). The list of properties within the actors section is simply a list of the actors by their id, with the lead actor always appearing first. Within each of these properties, the full details of that actor will be found as an object, making each a self-contained unit.

```
“actors”: {
  “actor.1”: {…},
  “actor.7”: {…},
..”actor.12”: {…}
}
```

Minions may be added to and removed from a character over its lifetime. Each minion is assigned a new unique value that is never re-used. So a Mechanic with a bunch of drones that are added and deleted may end up with gaps in the numbering, as shown in the example above.

Individual Actors

The structure of an actor object breaks down the actor into a small number of standard aspects and the game-specific details, which are represented by the following properties:

name	Name given to the actor by the user
player	Name of the player controlling the character (as entered by the user)
gameValues	Object containing all of the game-specific properties for the actor. These are all simple properties, so please refer to the specific game system section for details.
items	Object containing the collection of game-specific elements that comprise the character, encompassing all ability scores, skills, equipment, spells, etc.

```
“actor.1” : {
  “name”: “Hero”,
  “player”: “Blake”,
  “gameValues”: {…},
  “items”: {…}
}
```

Items

Within the “items” object, the structure is similar to the “actors” object. Each item is a uniquely named property within the object. It is possible that two items on a character represent the same general entity (e.g. two separate daggers), so each incorporates a unique value that guarantees distinction.

```
“items”: {
  “asStr.12”: {…},
  “skPerception.35”: {…},
..”wpDagger.47”: {…},
  …
}
```

Each individual item is an object with a collection of properties. A few of these properties are standard across all game systems, with the rest being game-specific.

name	Name of the item for display
description	Full description of the item for presentation to the user
summary	Short summary of the item for display to the user
compset	Identifies the grouping of related items to which this one belongs. For example, all ability scores will have the same compset, all skills will have the same compset, etc.

```
"asStr.12": {
  "name": "Strength",
  "description": "[description text]",
  "summary": "[summary text]",
  "compset": "AbilScore",
  …
}
```

Nesting of Items

There are frequently situations where items will be nested in the export. Typical examples include complex items built from multiple component items, items slotted into another (e.g. batteries and ammo for weapons), and gear stored within other gear. When any of these situations arise, any items held within another will appear within an "items" object on the containing item, and each such item will have an additional property that indicates the nature of the containment.

Containment	Means by which the item is contained – one of: Installed – Item is mounted within its container for use (e.g. ammo or a battery) Stored – Item is placed in its container for storage (e.g. food in a backpack)

```
"grBackpack.31": {
  "name": "Backpack",
  …
  "items": {
    "grBedroll.38": {
      "name": "Bedroll",
      "Containment": "Stored",
      …
    },
    "grSack.42": {
      "name": "Sack",
      "Containment": "Stored",
      …
      "items": {
        "wpDagger.54": {
        "name": "Dagger",
        "Containment": "Stored",
        …
        }
      }
    },
  .."grRope.67" : {
      "name": "50' Rope",
      "Containment": "Stored",
      …
    }
  }
}
```

Special Notes About Export

When properties have their default value, they are omitted from the export in an effort to keep the size from ballooning unnecessarily. There are numerous properties that are only used in certain situations, and they will have their default value unless actually used. Omitting them represents a huge savings in the overall export size and the volume of data to digest for a tool. The default value for numeric values is zero and the default for strings is the empty string.

There are a few instances where the default value for a property is meaningful. However, that property will still be omitted from the export. These specific exceptions will be documented when they are introduced in the details for each game system.

Differential Export

Complex characters can get quite large. That's fine for tools that take a snapshot of a character and never look back. However, some tools will want to monitor characters over time and track the changes applied to them by users, such as a well-integrated VTT. It's horribly inefficient for such a tool to repeatedly retrieve the entire character, so the HLO export makes it possible to report only the changes that have actually been applied to a character. This mechanism is referred to as ***differential export***.

A differential export can be explicitly requested by a tool. Each character has a version number that increments whenever a change is applied, and the current value is always reported in the "portfolio" section. If a tool already has version 42 of a character, then all it needs is the changes that have accrued since that version. So the tool can request differential export relative to a *baseline* version of 42. If three changes have been applied since then, the new version will be 45, and the differential export will return only the net effects of those three changes.

There are a variety of behavioral changes that will be seen in the differential export, and these are outlined in the following subsections.

General Behaviors

When differential export is employed by a tool, HLO assumes that the tool is solely interested in the mechanical data. Consequently, aspects like the description or summary of items are deemed to be of no interest to the tool and simply omitted, as they are typically lengthy and unnecessarily bloat the amount of data that needs to be transmitted and digested by the tool. The operational assumption is that the tool already has its own concept of the item appearing (e.g. weapon, ability, etc.), and it therefore already has its own description and summary. This keeps everything lean and efficient.

Properties

There are a diverse array of properties across all the various items on a character. The values of these properties will change as the character is modified by the user. To keep the differential export compact, only the properties whose values have actually changed will be included. Any property that has not changed will be omitted from the export and should be assumed to retain its previous value.

There are two important caveats to the above behavior:

- Properties that transition from the default value to a new value will suddenly appear within an item.
- Properties that transition from a non-default value to the default value will always appear in the differential export to clearly indicate the new value that has been assigned.

Newly Added Items

Whenever a new item is added to the character, it will simply appear within the differential export. If the new item is added within an existing item, it will be added beneath it appropriately. If a single change causes a chain reaction that adds multiple new items to the character, all of those new items will appear in the proper locations. This can even result in new items appearing as children of other new items.

Any new item will include all non-default fields, exactly the same way a full (non-differential) export behaves. The notable exception to this is that the summary and description properties will always be omitted from differential output.

Deleted Items

Items will occasionally be deleted from a character. When this occurs, the differential export will include a new property named "deletedItems". This property will appear at the topmost level and is an object that lists every item deleted from any actors across the entire portfolio. Each deleted item is an object whose name is the deleted item and whose lone property identifies the containing item from which it was deleted.

fromItem	Item that previously contained the deleted item (null when deleted directly from the actor and not a containing item)
fromActor	Actor that previously possessed the deleted item

```
“deletedItems”: {
  “grBackpack.31”: {
    “fromItem”: null,
    “fromActor: “actor.2”
  },
  “grSack.42”: {
    “fromItem”: “grBackpack.31”,
    “fromActor”: “actor.2”
  },
  “wpDagger.54”: {
    “fromItem”: “grSack.42”,
    “fromActor”: “actor.2”
  }
}
```

As shown in the example above, the “fromitem” value will be null if the item was deleted from the actor and not another containing item.

Moved Items

Items can also be moved from one location within the character to another. When this occurs, the moved item will disappear from beneath its old container and appear beneath its new container. In addition, the differential export will include a new object named “movedItems” at the topmost level. This object contains entries for each item that has been moved anywhere within the portfolio, where each identifies both the containing items from which it was moved and the new item that it now appears beneath.

fromItem	Item that previously contained the moved item (null when moved directly from the actor and not a containing item)
fromActor	Actor that previously possessed the moved item
toItem	Item beneath which the moved item now resides (null when moved directly to an actor and not into a containing item)
toActor	Actor that now possesses the moved item

```
“movedItems”: {
  “grSack.42”: {
    “fromItem”: “grBackpack.31”
    “fromActor”: “actor.2”,
    “toItem”: null,
    “toActor”: “actor.1”
  },
  “wpDagger.54”: {
    “fromItem”: null,
    “fromActor”: “actor.2”,
    “toitem”: “grBackpack.31”,
    “toActor”: “actor.1”
  }
}
```

As shown in the example above, the “fromItem” value will be null if the item was moved from the actor (not another containing item), and the “toItem” value will similarly be null if the item was moved into the actor (not another containing item).

Newly Added Actors

When a new minion is added to the character, a corresponding new actor will appear within the export, and all of the items for that actor will appear beneath it, with each following the same rules for added items. In addition, all of the non-default properties will be included for the new actor, along with the “gameValues”. Just like with newly added items, the appearance of the new actor is all the tool should need to recognize its existence and incorporate it into the tool’s data model.

Deleted Actors

Deleting an actor from a character causes two different behaviors. First, a new “deletedActors” property appears at the topmost level that contains an array of the actors that have been deleted. Secondly, all of the items belonging to the actor that were deleted are included within the “deletedItems” object.

```
“deletedActors”: [
  “actor.2”,
  “actor.7”
]
```

The primary character can never be deleted, so the actor with id 1 will always be present. All other actors can come and go, but “actor.1” will always remain.

Special Notes About Differential Export

It is entirely possible that a differential export will report zero net changes. Consider the case of a character that takes damage a few times and heals it all back. Or a player that makes changes to aspects of the character that simply aren’t included in the export format. In each of these cases, the export produced will indicate the version number has changed, but there will be no actual changes included in the “actors” section.

In the interest of efficiency and resource utilization on the server, HLO only tracks a limited history of changes made to each character. If a tool requests the differential changes relative to a baseline version that is older than that history, the export will simply include the entire character anew. When this occurs, the “baseline” property reported in the export will always be zero to indicate this result.

All Game Systems

Each game system has a variety of properties that appear on every actor, although each game system has its own unique set that reflects its distinct nature. In addition to these core properties that appear on every actor, there are thousands of different items that can be added, corresponding to the skills, abilities, gear, and other details that exist within the game.

Each of these items derives from a particular “compset”, or grouping of similar items, and each compset has a collection of properties that will typically appear within the export format. Many of these items and properties will be self-evident from their name, so an exhaustive list seems unnecessary. Instead, the various properties that are non-obvious are detailed in the sections beneath each game system.

Weapons

Weapons are emitted in a special manner to convey all the pertinent details, and the same basic structure is used across all game systems, so it’s outlined here. Each weapon possesses a “wpMelAttacks” and/or “wpRngAttacks” child object, depending on the nature of the weapon (melee, ranged, or both). Within this object, one or more attack methods are described, and they are named “atk1”, “atk2”, etc. Each attack method describes a style of attack with the weapon that confers different rolls to the character (e.g. standard attack, full attack, soldier’s onslaught, etc.). Depending on the game system and weapon, there may be one attack style or many listed.

For each attack style, there are six properties detailed:

Id	Identifier used for the attack style
name	Display name of the attack style
attack	Basic attack roll adjustment (e.g. "+3")
attackExpr	Dice roll expression for rolling the attack (see below)
damage	Familiar notation for the damage dealt by the attack (see below)
damageExpr	Dice roll expression for rolling the damage (see below)

NOTE! The damage information is typically only included for the first attack style ("atk1"). Unless the weapon is special and has different damage behaviors for each style, the damage properties are omitted for all subsequent attack styles, since it's the same for all of them.

NOTE! The dice roll expressions utilize the syntax adopted internally within HLO. That syntax has not been formally published at this time, although we expect to do so. Please be advised that the syntax may change prior to being publicly defined.

Starfinder

The sections below detail the non-obvious properties and behaviors for the Starfinder game system.

Actor Properties

The following perhaps non-obvious properties appear on an actor within the "gameValues" object:

actCR	Challenge rating for the character. Values from 1 upwards indicate that number as the CR. Other values and their meanings are: 0 = ½, -1 = ⅓, -2 = ¼
actXPAward	Amount of XP awarded when the actor is vanquished by the PCs
actSize	Size rating of the character as one of these values: 0 = Medium, -1 = Small, -2 = Tiny, -3 = Diminutive, -4 = Fine, 1 = Large, 2 = Huge, 3 = Gargantuan, 4 = Colossal
actSizeWeapon	Size rating of the weapons that can be comfortably wielded by the character as one of the same values as actSize above
actCarryingLevel	Net encumbrance rating of the character as one of these values: 0 = Unencumbered, 1 = Encumbered, 2 = Overburdened
actEncumbered	Bulk threshold at which the actor becomes Encumbered
actOverburdened	Bulk threshold at which the actor becomes Overburdened
actLevel	Current level of the character without any adjustments applied
actLevelNet	Current level of the character with adjustments applied, such as negative level effects
actSocietyId	Starfinder Society ID for the player, as issued by the SFS
actSocietyChar	Starfinder Society ID for the character, as assigned by the player and starting at 701. When combined with the actSocietyId, a unique id is generated for the character
actFameNet	Net Fame possessed by the character for use in Starfinder Society play

The following actor properties have important meaning when at their default value of zero, which is omitted from the export:

actCR	0 = CR ½
actSize	0 = Medium

actSizeWeapon	0 = Medium
actCarryingLevel	0 = Unencumbered

Item Properties

The following properties will appear on different items and may not be obvious:

AbilType	Type of ability as one of these values: SpellLike = spell-like ability, Super = supernatural ability, or Extra = extraordinary ability
Period	Interval at which an ability or other resource can be used (e.g. Day, Hour). When combined with the trkMaximum property, it identifies the actual frequency, such as 3/Day or 1/Hour.
UsageSpec	Frequency at which an ability or resource can be used as one of these values: AtWill, Constant, OneDFourRd (usable every 1d4 rounds), or OneDSixRd (usable every 1d6 rounds)
sitEffect	Stipulates specific situations under which adjustments are conferred, typically for an ability. For example, Perception might have a “+2 vs Sight” value, which means a +2 bonus to Perception is conferred on site-based checks.
AugmentCat	Indicates the Augment category that applies as one of these values: Biotech, Cyber, Magitech, Necrograft, Personal, or Symbiend
spLevelBase	Base level of a spell, which can then be adjusted for use
spLevelNet	Net adjusted level of a memorized spell
bnIsSlotted	Indicates whether an SFS boon has been slotted for use, with a value of 1 indicating slotted
Maneuver	Specifies the fly speed maneuverability as one of these values: Perfect, Good, Average, Poor, or Clumsy. Only ever assigned to the mvFly item.

The following item properties have important meaning when at their default value of zero, which is omitted from the export:

stNet	Final adjusted value is zero for ability scores, initiative, armor class, etc.
arKAC	Armor class value of +0
arEAC	Armor class value of +0
arArmorCheckPen	Armor check penalty of 0 (i.e. none)
arUpgrade	Armor has no upgrade slots
arSpeedPen	Armor has no movement speed penalty
stackQty	Indicates the stacking quantity is actually one (not zero)
grBulk	Gear has negligible bulk
spLevelBase	Spell is a cantrip
spLevelNet	Spell is a cantrip
skClassSkillBon	Skill is not a class skill

Pathfinder 2nd Edition

The sections below detail the non-obvious properties and behaviors for the Pathfinder 2nd Edition game system.

Actor Properties

The following perhaps non-obvious properties appear on an actor within the “gameValues” object:

actSize	Size rating of the character as one of these values: 0 = Medium, -1 = Small, -2 = Tiny, -3 = Diminutive, -4 = Fine, 1 = Large, 2 = Huge, 3 = Gargantuan, 4 = Colossal
actCarryingLevel	Net encumbrance rating of the character as one of these values: 0 = Unencumbered, 1 = Encumbered, 2 = Overburdened

The following actor properties have important meaning when at their default value of zero, which is omitted from the export:

actSize	0 = Medium
actCarryingLevel	0 = Unencumbered

Item Properties

The following properties will appear on different items and may not be obvious:

stNet	When shown on a spell, this is the spell attack roll
reqLevelNet	Identifies the level of a feat and the required level of the character to take the feat
Action	Identifies the type of action the an ability requires for use as one of these values: Action1, Action2, Action3, Reaction, or Free
Trait	Identifies the various traits associated with the item (see below)

The following item properties have important meaning when at their default value of zero, which is omitted from the export:

<none>	

NOTE! Traits are a core game concept, and the list of traits increases with each new book published. Trait ids will possess an obvious value that closely parallels the name in the book. In most cases, the trait will have a prefix of 2 or 3 letters. For example, classes will have the “cl” prefix (e.g. “clRogue”), spell schools will have the “ss” prefix, and monster/npc traits will have the “trt” prefix (e.g. “trtDragon” or “trtFire”).

Appendix 1: Revision History

This appendix identifies the various behavioral changes that have been introduced into the Export Format and when those revisions occurred. Only behavioral changes are listed here. Documentation changes that don’t involve the functionality of the Export Format (e.g. typos, formatting, clarifications) are not included here.

V1.1 2020/11/23	• Introduced support for Pathfinder 2nd Edition • For Starfinder, weapons are now exported differently. Previously, three properties represented a single attack/damage, and those have been eliminated. In their place, a “wpMelAttacks” and/or “wpRngAttacks” object is now emitted, and this object contains a separate child object for each attack style (e.g. standard attack, full attack, soldier’s onslaught). Each of these attack styles has a collection of properties that describe its details.