

# KWIVER Packet Format (KPF)

## Motivation

The KPF format is used to persist object detections, tracks, events, and associated metadata in KWIVER. KPF uses a subset of YAML as its default text transport, as well as defining a set of semantic tokens useful for object detection and tracking, as well as activity recognition. It is hoped that by defining a core set of semantic concepts and their common representation, KPF will enable interoperability between various applications (e.g. trackers, detectors, GUIs, and evaluation tools.) KPF also provides for application-specific extensions to be persisted and parsed, yet ignored when not applicable.

From a design perspective, KPF tries to **provide**:

- unambiguous representation of objects, tracks, simple and complex events
  - both semantically ("timestamp is 4000? 4000 *what?*") and at the wire-level ("This box is 100, 200, 105, 205-- is that (x1,y1)-(x2,y2) or (x1,y1,w,h)?")
- relational support for linking, grouping, and mapping sets of items
- transport-agnostic representation (should be easily convertible between text, json, xml, protobuf)
- extensibility ("Hey, in Phase 2 the program just defined ten more event types!")
- explicit linkages between documents
- simplicity for common use cases, feasibility for most use cases
- convertibility from existing formats
- compatibility (in text mode) with command-line tools such as awk, grep, and perl

KPF tries to **avoid**:

- MAGIC NUMBERS
- excessive redundancy and verbosity
- obsessive removal of all possible ambiguity

# Key KPF concepts

As stated above, KPF is **not** a single file format or set of formats. There is no single "track file format" or "event file format". Instead, KPF aims to provide a unified representation of the fundamental concepts common to computer vision applications, and a principled way to add new concepts. A KPF parser should be able to process any KPF-compliant input; the decision whether a particular instance meets the requirements of an application is an application-level decision, rather than an output of the parser.

In particular, if an application processes a KPF file containing both tokens it knows about (bounding boxes and frame numbers, for example) and tokens it does not recognize (say, activity instances) it should be able to process the recognized tokens ***without concern that the unknown tokens have altered the interpretation of the known tokens***. In other words, the interpretation of a particular token is independent of the presence or absence of other tokens.

## Packets and Domains

A recurring theme is that many concepts we want to represent (timestamps, object detections, event names) have natural representations which, if naively transcribed, become ambiguous due to different sources / references / coordinate systems. KPF handles this by sharing the representation between ***packets*** and ***domains***.

## Packets

Packets are the semantic schema of a concept. Items such as timestamps, detections, events, bounding boxes are all represented by specific packets. Complex concepts (e.g. an activity instance) may be composed of multiple simpler packets (timestamps, track IDs, etc.)

## Domains

A domain specifies the **context of a packet**. For example, a bounding box might have the domain of "pixel coordinates" or "world coordinates"; a timestamp might have the domain of "frame number" or "usecs since midnight 1 Jan 1970". A location might have the domain of "lat/lon", "utm", or "pixels". These are examples; default domains are provided for the common cases, but clients are free to specify a new custom domain. (The "social" aspects of **defining** a custom domain, ensuring that it does not conflict with somebody else's domain, whether or not it even needs to BE its own domain, are all explicitly outside the scope of KPF. By making the domain explicit, KPF allows for conflict detection, but does not handle conflict resolution.)

Roughly speaking, the packet tells you what something is; the domain grounds it in the units / coordinate frame / event vocabulary / etc.

## Anatomy of a packet

A packet's format is `[packet-tag] [domain] : [space] [payload]`; for example, the packet

`g0: 1080 229 1112 261`

has the packet tag "g", domain "0", and payload "1080 229 1112 261".

***The format of the payload is fixed by the tag. The interpretation of the payload is dictated by the domain.***

An equivalent XML representation might be

```
<geometry domain="0">
  <payload> 1080 229 1112 261 </payload>
</geometry>
```

Note that there is no subdivision of the payload into explicit corner points.

## Packet tags: conceptual extensibility

The packet tag is a string; as we add more concepts, we add more tags.

## Domains: application extensibility

The domain is an integer specifying the context in which to interpret the packet and effectively acts as a namespace. The general idea is that as more efforts define their own interpretations of tags (for example, different programs with different definitions for "U-turn"), each effort gets its own domain.

For most packet types, a few pre-defined domains will suffice.

Inevitably, the problem of domain allocation and conflict resolution will arise. Similar to the Well Known Ports in `/etc/services`, we propose the following policy:

- domains 0-9 are **reserved** and predefined as necessary.
- Application-specific domains start numbering at 10; the mapping of the domain to a specific application should be provided via the **meta**: tag. The meta tag has no domain; its payload is a string whose format is unspecified. For example:

```
-{ meta: "loc13 coordinate system: see /projects/foo/refcoords.txt"}
```

It may be that a different project was already using loc13 without our knowledge; handling such organizational conflicts is explicitly outside the scope of KPF.

## KPF and YAML

KPF uses [YAML](#) as its default text transport. YAML was chosen over a more lighter-weight ad-hoc text format based on the following considerations:

- YAML parsers are readily available for many languages
- Although YAML is more verbose for very simple schemas, it provides more structure for complex concepts (such as activities)
- Its flexibility reduces risk that future, possibly more complex concepts will require major revisions to the underlying format
- It can still be represented as record-per-line text file, allowing rapid data analysis using standard tools such as awk, grep, and perl.

## KPF Use Case

Here we show an example of how a KPF file might evolve through a pipeline of detection, tracking, and scoring. The files are unrealistically short for brevity, but each line is meant to be complete.

### Step 1: Detection

A detector output might look like:

```
- { meta: "cmdline0: run_detector param1 param2..." }
- { meta: "conf0: yolo person detector" }
- { meta: "id0 domain: yolo person detector" }
- { geom: { id0: 0, ts0: 101, g0: 515 419 525 430, cset0: {Person: 0.8} }}
- { geom: { id0: 1, ts0: 101, g0: 413 303 423 313, cset0: {Person: 0.3} }}
- { geom: { id0: 2, ts0: 102, g0: 517 421 527 432, cset0: {Person: 0.7} }}
- { geom: { id0: 3, ts0: 102, g0: 416 304 421 315, cset0: {Person: 0.2} }}
```

**NEW in KPF V4:** The `cset0: {Person: 0.7}` packet replaces the `conf0: 0.7` packet; this allows detectors to report multiclass likelihoods (e.g. `cset0: {Person: 0.7, Vehicle: 0.2}`).

**NEW in KPF V3:** The `geom: { ... }` wrapper is a **schema tag**; it is meant to convey a hint about how the associated set of packets should be interpreted and to be used in validation. See "Schema types" below.

ad

Here the `id0` and `conf0` domains are specified to be the detections from yolo; the timestamp `ts0` and geometry `g0` domains are predefined to be frame number and pixel coordinates, respectively.

## Step 2: Tracking

A tracker (detection linker) could take the above and generate the following. New packets are highlighted.

```
- { meta: "cmdline0: run_detector param1 param2..." }
- { meta: "conf0: yolo person detector" }
- { meta: "id0 domain: yolo person detector" }
- { meta: "cmdline1: run_linker param1 param2..." }
- { meta: "id1 domain: track linker hash 0x85913" }
- { geom: { id0: 0, ts0: 101, g0: 515 419 525 430, cset0: {Person: 0.8}, id1: 100 } }
- { geom: { id0: 1, ts0: 101, g0: 413 303 423 313, cset0: {Person: 0.3}, id1: 102 } }
- { geom: { id0: 2, ts0: 102, g0: 517 421 527 432, cset0: {Person: 0.7}, id1: 100 } }
- { geom: { id0: 3, ts0: 102, g0: 416 304 421 315, cset0: {Person: 0.2}, id1: 102 } }
```

Here all the tracker has done is defined an additional domain for IDs (`id1`) which it uses to link detections into tracks.

## Step 3: Scoring

An evaluation run could take the output from the tracker and produce the following. Again new packets are highlighted.

```
- { meta: "cmdline0: run_detector param1 param2..." }
- { meta: "conf0: yolo person detector" }
- { meta: "id0 domain: yolo person detector" }
- { meta: "cmdline1: run_linker param1 param2..." }
- { meta: "id1 domain: track linker hash 0x85913" }
- { meta: "cmdline2: score_tracks param1 param2..." }
- { meta: "overall track pd/fa count: 0.5 / 1" }
- { meta: "eval0 domain against id0" }
- { meta: "eval1 domain against id1" }
- { meta: "id2 domain false negatives from official_ground_truth.kpf" }
- { geom: { id0: 0, ts0: 101, g0: 515 419 525 430, cset0: {Person: 0.8}, id1: 100, eval0: tp, eval1: tp } }
- { geom: { id0: 1, ts0: 101, g0: 413 303 423 313, cset0: {Person: 0.3}, id1: 102, eval0: fa, eval1: fa } }
- { geom: { id0: 2, ts0: 102, g0: 517 421 527 432, cset0: {Person: 0.7}, id1: 100, eval0: fa, eval1: tp } }
- { geom: { id0: 3, ts0: 102, g0: 416 304 421 315, cset0: {Person: 0.2}, id1: 102, eval0: fa, eval1: tp } }
- { geom: { id2: 0, ts0: 101, g0: 600 550 605 610, eval0: fn, eval1: fn } }
- { geom: { id2: 1, ts0: 102, g0: 603 553 608 615, eval0: fn, eval1: fn } }
```

Here, the scoring code has done several things:

- It has added a summary of its scoring results to the preamble via the **meta** packets.
- It has added two sets of eval packets to each detection; **eval0** is the detection-level result against **id0**, **eval1** is the track-level result against **id1**.
- It has added a wholly new set of boxes in a new domain (**id2**); these are the false negatives (undetected boxes) from the ground-truth file named in the **meta** packet.
  - These new tracks have IDs which collide with those from domain 0, but they are still separated since they come from a different domain.

This KPF file could be used for visualizing results; one could easily imagine a pull-down menu allowing selection of individual ID domains populated with the text from the corresponding **meta** packet.

## Packet types

This is a proposed list of packet types, drawn mostly from the `data_terms` in `track_oracle` plus the `attributes` file.

Generally, when a number is undefined, 'x' is used.

Strings should be quoted when they contain spaces and use `\` as an escape character.

packet	YAML format	definition	pre-defined domains / notes
<b>id</b>	<b>idN:</b> <i>int</i>	object identifier	none; may start at 0 but should specify source via a meta packet.
<b>ts</b>	<b>tsN:</b> <i>double</i>	timestamp	0: frame number 1: seconds since beginning of video 2: usecs since unix Epoch (1 Jan 1970 UTC)
<b>tsr</b>	<b>tsrN:</b> [ <i>double double</i> ]	timestamp range	(same as ts) (maybe use 'x x' to mean "all the time"?)
<b>loc</b>	<b>locN:</b> <i>x y z</i>	location	0: pixel coordinates (z is undefined) 1: lon / lat / altitude-in-meters 2: UTM (e.g. "17N 630084 4833438")  Locations in world coordinates (e.g. via homographies) should use a domain > 9 and specify the homography file used via a <b>meta</b> packet.
<b>g</b>	<b>gN:</b> <i>x1 y1 x2 y2</i>	bounding box	0: pixel coordinates. (x1,y1) is upper-left; (x2,y2) is lower-right; image origin (0,0) is upper-left corner.
<b>poly</b>	<b>polyN:</b> [ [x1,y1] [x2,y2] ... ]	polygon	0: pixel coordinates

<b>conf</b>	<b>confN:</b> <i>double</i>	confidence or likelihood	same protocol as id. (Ground-truth should additionally be represented via a 'src: truth' kv packet, rather relying exclusively on a confidence of 1.0.)
<b>cset</b>	<b>csetN:</b> { <i>label1: likelihood1, [label2: likelihood2, ...]</i> }	set of label / likelihoods	3: DIVA objects
<b>act</b>	<b>actN:</b> { <i>activity_label_1: likelihood1, [activity_label_2, likelihood2, ...]</i> } <i>id packet</i> , <b>timespan:</b> [ { <i>tsr-packets</i> } ], <i>[opt: kv-packet...]</i> <b>actors:</b> [ { <i>id-packet</i> , <b>timespan:</b> [ { <i>tsr packets</i> } ] }, (more actors) } ]	activity	0: VIRAT 1: vidtk 2: DIVA activities (Phase 1) 3: DIVA activities (Phase 2)  Activity names are spelled out. Participating objects should specify timestamp ranges in the same domains as the event itself.  Timespans are represented as arrays of tsr packets to allow for future inclusion of synchronized world clocks.
<b>eval</b>	<b>evalN:</b> <i>result-string</i>	evaluation result	same protocol as id. The result-string is e.g. 'tp' for true positives, 'fa' for false alarms, etc.
<b>a</b>	<b>aN:</b> <i>attribute_string</i>	attribute	same protocol as id. Specifies that the named attribute applies in the current scope.
<b>tag</b>	<b>tag:</b> <i>packet string</i>	tag a packet / domain pair	Used to link multiple files, i.e. 'tag: id0 collect5' in file A and 'tag: id3 collect5' in file B essentially means file A's id0 domain is the same as file B's id3 domain.
<b>key</b>	<b>key:</b> <i>value</i>	key / value	Keys are distinguished from KPF packets as they have no domain integer before the colon, i.e. 'src0:' might be a KPF packet but 'src:' is not.

## Schema types

The schema type tag associates the set of packets with a text string intended to convey how to interpret the set of packets and to allow for validation. If no schema tag is supplied, the record is parsed with an implicit 'unspecified' schema (i.e. no validation.)

Defined schema types are:

tag	interpretation
geom	Geometry

<b>act</b>	Activity
<b>types</b>	Object types
<b>regions</b>	Polygonal scene regions
<b>unspecified</b>	"other"; no specified intent

## DIVA-specific schemas

### Annotation delivery:

1. file for geometry + frame-level attributes + track IDs + polygons. Size:  $O(n\text{Detections} \times n\text{Frames})$
2. file for object labels: size  $O(n\text{Objects})$
3. file for events: size  $O(n\text{Events})$
4. file for region polygons: size  $O(n\text{StaticObjects} \times \text{lenStaticObjectTracks})$

### Geometry schema: (line breaks for clarity)

```
- { geom: { id0: detection-id, id1: track-id, ts0: frame-id, g0: geom-str, src: source
  [occlusion: (medium | heavy)]
  [cset3: {object: likelihood, ... } ]
  [evalN: eval-tag...] }}
```

### Example (simple) (line breaks for clarity):

```
- { geom: { id1: 0, id0: 37, ts0: 37, g0: 432 387 515 444 ,
src: truth, occlusion: heavy }}
```

...Detection 37 is associated with track 0 on frame 37, and is a box from image coordinates (432, 387) to (515,444). This detection is ground truth and an annotator has marked the object's occlusion level as "heavy".

### Example (slightly more complicated) (line breaks for clarity)

```
- { geom: { id1: 0, id0: 37, ts0: 37, ts1: 18.5, g0: 432 387 515 444 ,
  src: XYZ_v2, occlusion: heavy,
  cset3: {Person: 0.6, Vehicle: 0.2},
```



```
eval0: fn, eval1: tp }}
```

Similar to previous example, but now with another timestamp (18.5 seconds since the beginning of the video) and results from an evaluation run: **eval0** marked it as a miss (false negative) in the detection domain, while **eval1** found that it was a hit (true positive) in the track domain. The src tag indicates this is not a ground-truth packet but instead a classification from the XYZ\_v2 system.

**NEW in KPF V4:** The **cset3** packet contains the XYZ\_v2 system's classification results. (If classification results are at a track level rather than the detection level, they may be provided in a types file for brevity.)

Object label schema:

```
- { types: {id1: track-id, cset3: { object_type: likelihood, ... } } }
```

**NEW in KPF V4:** types are communicated via a **cset3** packet, rather than an **obj\_type**: key-value packet.

Examples:

```
- { types: { id1: 35 , cset3: {Vehicle: 1.0} } }
- { types: { id1: 36 , cset3: {Vehicle: 1.0} } }
- { types: {id1: 5000 , cset3: {Parking_Meter: 1.0} } }
- { types: {id1: 5001 , cset3: {Dumpster: 1.0} } }
```

Activity schema:

```
- { act { actN: {activity_name: likelihood, ...} , id_packet, timespan:
[{{tsr_packet} (... tsr_packet)}], src: source, actors: [ {id_packet, timespan:
[{{tsr_packet} (... tsr_packet)}]} (, next actor identification... ) ]}}
```

**NEW in KPF V4:** the activity type is a name/likelihood map, much like a **cset** packet.

Example (line breaks for clarity)

```
- { act: {act2: {Talking: 1.0}, id2: 3, timespan: [{tsr0: [3293, 3314]}]},
  src: truth,
  actors: [{id1: 9, timespan: [{tsr0: [3293, 3314]}]} ,
    {id1: 12, timespan: [{tsr0: [3293, 3314]}]} ,  ]}}
```

This line of YAML can be rendered as a python object via  
<http://yaml-online-parser.appspot.com/> as

```
[{'act': {'act2': {'Talking': 1.0},
```

```

'actors': [{ 'id1': 9, 'timespan': [{ 'tsr0': [3293, 3314] } ] },
            { 'id1': 12, 'timespan': [{ 'tsr0': [3293, 3314] } ] } ],
'id2': 3,
'src': 'truth',
'timespan': [{ 'tsr0': [3293, 3314] } ] } ]

```

Here the event is parsed as follows:

- **act2**: this is an event in domain 2 (notionally DIVA)
- **{Talking: 1.0}** event is 'walking' with a likelihood of 1.0
- **id2: 3** event ID is 3 (explicitly also in domain 2, DIVA)
- **timespan: [{tsr0: [3293, 3314]}]** The activity as a whole starts at frame 3293 and ends at 3314. Frame numbers are domain 0 for timestamp range.
- **actors:...** This is an array of the actors participating in the event:
  - **id1: 9** First actor is track ID 9
  - **timespan: [{tsr0: [3293, 3314]}]** The first actor is participating in the event from frames 3293 to 3314
  - **id1: 12** Second actor is track ID 12
  - **timespan: [{tsr0: [3293, 3314]}]** The second actor is also participating in the event from frames 3293 to 3314
- **tsr0: 19 402** event takes place from frames 19 to 402 (in timestamp domain 0)
- **src truth** the event is a ground-truth event (i.e. sourced from "truth"; other detectors would substitute their own sources)

Timestamp ranges are stored as arrays in anticipation that multiple-camera events will be accessible from multiple time reference points (e.g. frames-since-video-start as well as world-clock-time.)

## Regions schema:

Some objects (static objects, scene segmentation) have been annotated as polygons; for example, to delineate "do-not-score" areas. The schema is

```

- { regions: { idN: id, tsN: timestamp, [ keyframe: (0|1) ], polyN: [[ x0, y0 ], [ x1, y1 ] ... ] } }

```

Example:

```

- { regions: { id1: 1, ts0: 9063, keyframe: 1, poly0: [[ 1435.88, 1 ], [ 1435.88, 68.76 ], [ 1456.88, 68.76 ], [ 1456.88, 1 ], ] } }

```

This is parsed as:

- **id1: 1** This polygon pertains to track ID 1
- **ts0: 9063** This polygon is on frame 0
- **keyframe: 1** This particular polygon is marked as keyframe (for interpolation)
- **poly0: [[ 1435.88, 1 ] ...]** An array of (x,y) points in image coordinates (domain 0).

##

## Questions and comments

Please send any questions and/or comments to [roddy.collins@kitware.com](mailto:roddy.collins@kitware.com) and [keith.fieldhouse@kitware.com](mailto:keith.fieldhouse@kitware.com).