Extensible Device Metadata

VERSION 1.0

Document version: 10/27/2015

Contents

Contributors

The eXtensible Device Metadata specification was created by the XDM working group, listed below in alphabetical order:

Houman Alagha, Intel Corp.
Ivan Dryanovski, Google Inc.
Jim Foley, Google Inc.
James Fung, Google Inc.
Carlos Hernández, Google Inc.
Ryan Hickman, Google Inc.
Rohit Israni, Intel Corp.
Ravi Teja Kommineni, Google Inc.
Cian Montgomery, Intel Corp.

Overview

The eXtensible Device Metadata (XDM) specification is a standard for storing device-related metadata in common image containers such as JPEG and PNG while maintaining compatibility with existing image viewers. The metadata that can be stored includes depth map, point cloud, device and camera pose, lens perspective model, image reliability data, and vendor-related information about the device and sensors. The data storage format is based on the Adobe XMP standard. XDM is being developed as an open file format at xdm.org.

The XDM specification includes support for multiple cameras, each with its own relative physical orientation. Each camera data structure can optionally contain an image and depth data if the device platform can provide them.

Example: XDM can enhance a standard image (left) with a depth map (right) as metadata.

Backward Compatibility

The eXtensible Device Metadata specification is a significant expansion of the original DepthMap Metadata specification published in 2014. It still supports the original use case of a single-image container with associated depth metadata, but expands that original specification to support more types of metadata and more use cases. The two specifications are not compatible since they use different namespaces.

Applications that supported the DepthMap Metadata spec will require modification to support XDM. The XDM standard handles a number of items differently, including: `Units`, `Confidence`, `Manufacturer`, `Model`, `ImageWidth`, and `ImageHeight`.

XDM supports image containers that include JPEG, PNG, TIFF, and GIF. In this documentation, JPEG is used as the basic model, but the concepts generally apply to other image-file types supported by XMP.

Design

The metadata is serialized and embedded inside the image file as described in the Adobe XMP standard. This section provides an overview of the design. Subsequent sections provide detailed definitions of the data elements and sample data.

Data Structure

The image file (i.e., the JPEG, PNG, TIFF, or GIF file) contains the following items, formatted as RDF/XML. This describes the general structure; specific elements are defined in a separate section, and the sample data provides examples.

- Container image - The image external to the XDM, visible to normal non-XDM apps
- `Device` - The root object of the RDF/XML document as in the Adobe XMP standard
    - `Revision` - Revision number of XDM specification
    - `VendorInfo` - Vendor-related information for the device
    - `DevicePose` - Device pose with respect to the world
    - `Cameras` - RDF sequence of one or more `Camera` entities
        - `Camera` - All the info for a given camera. There must be a camera for any image. The container image is associated with the first camera, which is considered the primary camera for the image.
            - `VendorInfo` - Vendor-related information for the camera
            - `CameraPose` - Camera pose relative to the device
            - `Image` - Image provided by the camera
            - `ImagingModel` - Imaging (lens) model
            - `DepthMap` - Depth-related information including the depth map and noise model
                - `NoiseModel` - Noise properties for the sensor
            - `PointCloud` - Point-cloud data

Versioning

XDM uses a loosely-restricted versioning system that does not change versions for minor enhancements as long as they don't break compatibility. It works like this:

- The `Device` object (the root of the XDM data structure) includes a `Revision` element that specifies the object's overall XDM version number. This is a minimum compatibility level for applications using that `Device` object. For example, any file with a `Revision` of "1.0" should be readable by any application that supports XDM 1.0.
- `Revision` is an informational field that lets applications quickly check compatibility. Applications that create or modify XDM data should take care to place the true value here, and be sure the file does not include any `xmlns` lines that use the namespace of a conflicting XDM version. In the event of a conflict between actual version and the

`Revision` value, applications that consume XDM data should be able to handle the error gracefully.

- Major revisions (e.g., 1.0, 2.0, etc.) indicate approved changes that break compatibility with the old version or introduce major new features that change the scope or direction of the specification.
- Minor revisions (e.g., 1.1, 1.2, etc.) indicate approved changes that expand functionality but do not break compatibility within that major version.

Namespace requirements

Some XDM clients may not support all XDM features, or all versions of a feature. For this and other reasons, apps should be able to quickly get a list of the XDM namespaces (i.e., features and feature versions) used in a file before they begin processing it. Unfortunately, this can be difficult. If the `Device` element is more than 64K (true of most XDM files), the rules of XMP force the `Device` and its children out of the main XMP section and into the extended section. Thus an XDM element and its namespace declaration might appear anywhere in the main or extended XMP. Under these conditions, building a list of all the XDM namespaces used in a file requires checking the entire XDM content, often megabytes in length, causing a performance hit when opening the file.

To avoid this, XDM requires that all namespace declarations appear in the main XMP section or the first 64K of the extended section. This allows clients to quickly create a list of the required namespaces by reading just those two sections (less than 128K), without having to load and parse the entire extended section.

Since the XDM content in a file often includes a depth map and a copy of the container image, XDM is usually larger than 64K and therefore usually found in the extended section. However, the namespace declarations must be located in the main section or first block of the extended section.

Non-standard usage

Be aware that a `Device` object may contain other fields or objects that are not defined in the spec for the specified version of XDM. For instance, these may be objects specific to a particular vendor, device, or use-case, or experimental objects that have not been formally added to the XDM spec. As long as they do not break compatibility or cause failure in applications that are otherwise compatible with the XDM version, these non-standard files can use the standard value for that XDM version in the `Revision` field.

Requirements

The `Device` element is always required, since it is the root of the object. Other elements may be optional, depending on the usage requirements. Several sample use cases are provided here to help clarify this.

Use case: Depth Photography

For an application to support depth-related use cases, at a minimum it requires:

- The container image (that is, the image external to the XDM data, which is visible to normal non-XDM apps). In depth photography, think of this as the "presentation" or "display" copy of the image.
- An XMP `Device` structure containing a `Camera` (Camera 0).
- The `Image` for Camera 0. Since the first camera is the one associated with the container image, this is an exact copy of the original container image. This is not required if the camera 0 image and container image are identical. As discussed below, this is useful if an application later modifies the container image in a way that changes its relation to the depth data. Where the container image is thought of as a "display copy", the `Image` is the "ground-truth" image.
- The `DepthMap`, which is also stored in Camera 0.

Image and depth map correlation

The camera image and depth map must be rectified to the same pose and cropped to the common overlapping field of view (same aspect ratio). It is not necessary for the image and depth map to have the same resolution.

If a use-case requires that you save all of the captured image and depth data, then in addition to the structure described above, store the uncropped, full-sized camera image in an additional `Camera` element, and the uncropped, full-sized `DepthMap` in a third `Camera` with accurate `CameraPose` (position and orientation data). The resulting file will be large, but preserves all options for later use by consumer apps.

Gaps in depth maps

A depth map for the first camera must have no holes, since it may be used for image processing by applications. For any region where depth cannot be calculated, the creator of the depth map must provide a best-guess value.

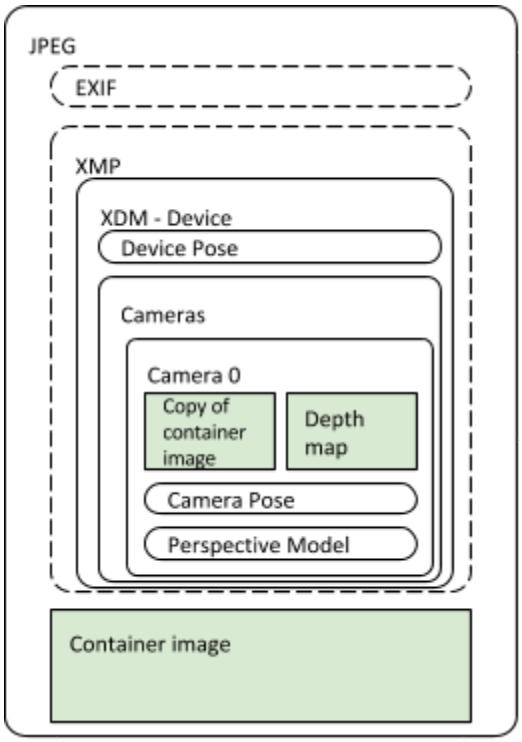
Compatibility with non-XDM applications

As suggested above, in depth photography the Camera 0 element contains a ground-truth copy of the container image as a fallback in case a non-XDM-aware app edits the container image (or

"display" image) in a way that breaks the depth feature. In support of this, if an XDM-aware application edits a container image for which there is no corresponding `Image` in Camera 0, it is suggested to create one before writing the revised container image to the file.

For XDM-enabled applications such as a depth-based bokeh filter, it is expected that the app will start with the unedited `Image`, apply the effect, and write the result to the container image so that all applications can view the applied effect – while leaving the original `Image` copy unchanged.

The obvious drawback of this redundancy is that it increases file sizes by 50% or more. If file size is a concern, the original creator of the file may opt to leave out the `Image`, at the risk of a non-XDM app modifying the container image and possibly invalidating the depth information.



XDM file for depth photography

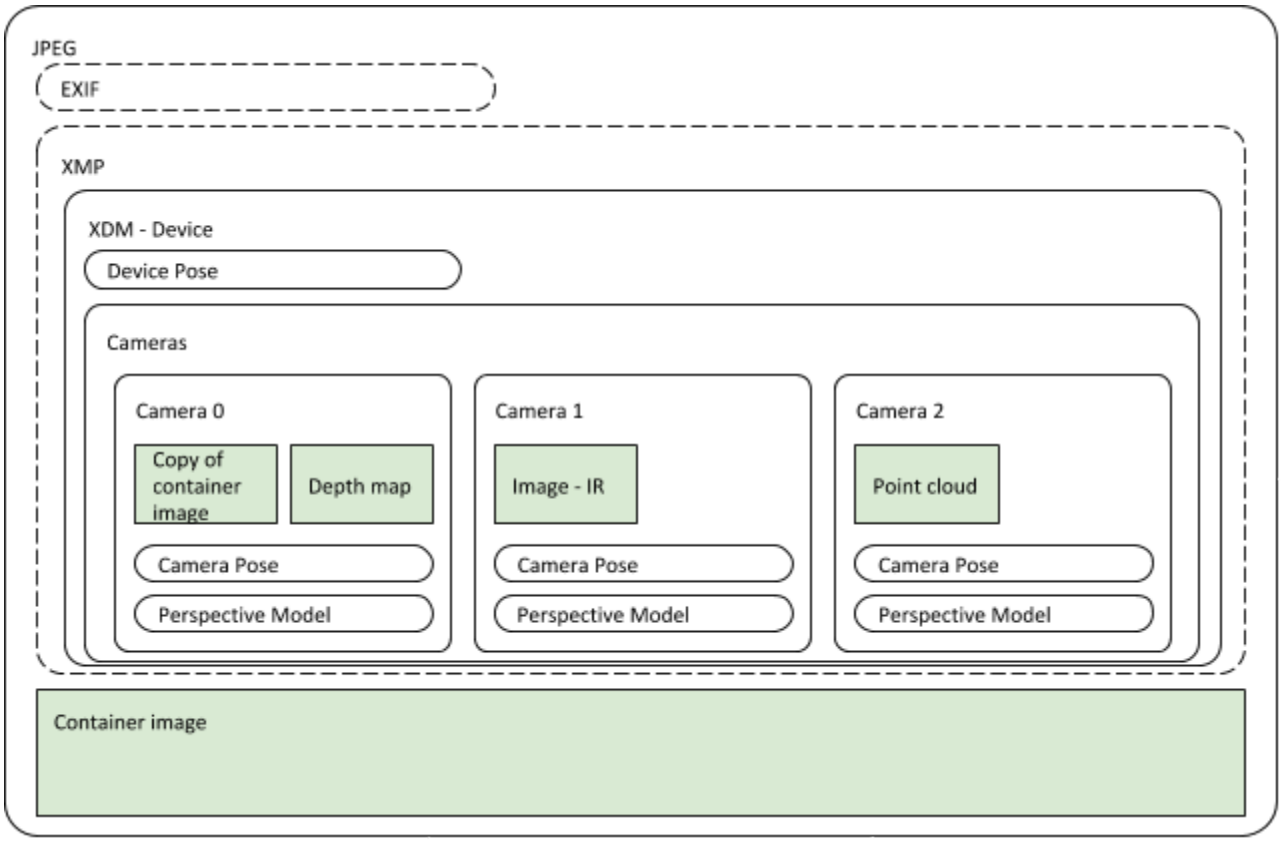
Use case: Multiple cameras

For an application to support multiple cameras on a device, at a minimum it requires:

- The container image, external to the XDM
- The `Device` element and `Cameras` element
- Within the `Cameras` structure, one or more `Camera` elements which each contain at least one of these: camera image, depth image, point cloud, etc. If it contains more than one of those elements, they must be rectified and cropped so that they match without further manipulation. Camera 0 is associated with the container image, so if the first `Camera`

contains a camera image it should be a ground-truth version of the container image as discussed under the Depth Photography use case.

- Any `Camera` element other than Camera 0 must have a pose relative to the first camera. If not, it is assumed that any images in that `Camera` are rectified to Camera 0.



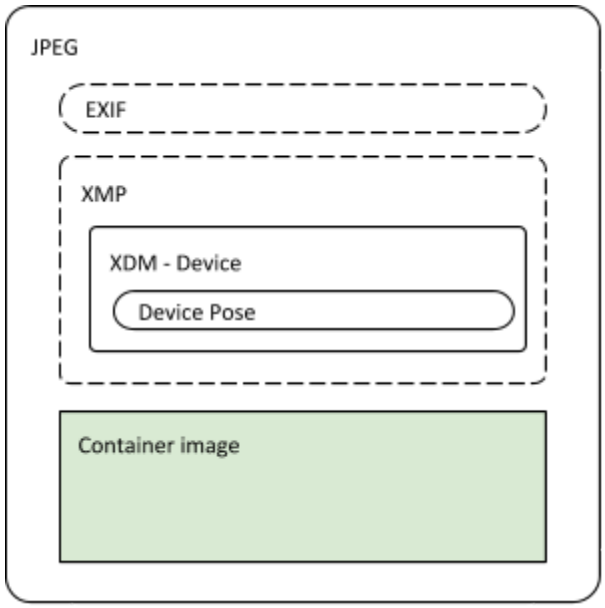
One possible XDM file for a multi-camera use case

Use case: Pose only

For an application to store full pose data with an image, at a minimum it requires:

- The container image, external to the XDM
- The `Device` structure that contains a `DevicePose` structure

The pose data would contain the latitude, longitude, altitude, and spatial orientation for the container photo.



Very simple XDM for storing location and orientation with a photo

Element Definitions

This long section provides details on each defined XDM element.

Device

The `Device` element describes device-related information and a sequence of cameras that are related to the JPEG image that contains this device.

- The namespace URI is `http://ns.xdm.org/photos/1.0/device/`.
- The default namespace prefix is `Device`.

The first camera in the sequence is the one that created the container JPEG image.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Revision	string	Yes	N/A	The revision of the XDM spec, e.g. “1.0”. Changes to a minor version number ("1.1", "1.2", etc.) do not break compatibility within that major version. See the section on Versioning, under Schema, for more on this. Note that this field is informational; actual compatibility depends on namespace versions.	No change
ContainerSignature	string	Yes		Adler-32 checksum of the encoded image in the container image. XDM apps should update this value if they change the container image. If the stored checksum does not match the calculated checksum, the container image has been altered by a non-XDM application.	Update to match new container signature
Cameras	Sequence of Camera (rdf:Seq)	Depends on use case	N/A	Each `Camera` in the `Cameras` sequence contains the properties of a camera on the device associated with this JPEG. The first `Camera` in the sequence is for the camera that captured the container image.	Needs update
Pose	DevicePose	No		The pose of the device, i.e., location on the earth and orientation relative to gravity and magnetic north.	No change
VendorInfo	VendorInfo	No		Vendor information for the device	No change

Vendor Information

The `VendorInfo` element describes vendor information for a camera or a device.

- The namespace URI is `http://ns.xdm.org/photos/1.0/vendorinfo/`.
- The default namespace prefix is `VendorInfo`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Model	string	No		The model of the element that created the content	scale/crop: No change
Manufacturer	string	Yes	N/A	The manufacturer of the element that created the content	scale/crop: No change
Notes	string	No		General comments	scale/crop: No change

Device Pose

The `DevicePose` element describes the 3D pose of the device – specifically, the pose of the camera that captured the image.

- The namespace URI is `http://ns.xdm.org/photos/1.0/devicepose/`.
- The default namespace prefix is `DevicePose`.

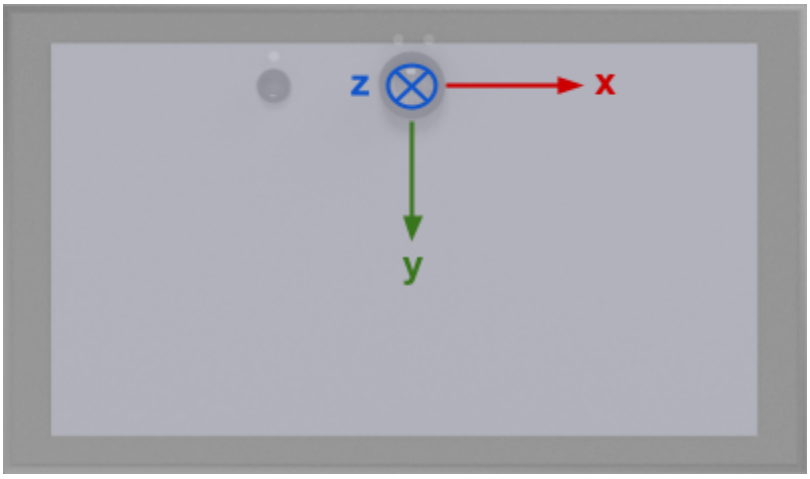
A *pose* describes where something is located (*position*) and in what direction it is facing (*orientation*).

In the XDM metadata for a photograph, it's important to understand that the "device pose" is the pose of the camera that captured the image. The use of the term "device" in this section should be understood in that context.

Overview

To describe the position is pretty simple; with the earth as a reference frame, we use the device's latitude, longitude, and altitude to describe its position. XDM follows WGS84 conventions; altitude is height in meters above the standard spheroidal reference surface, and latitude is geodetic latitude. This is consistent with the GPS data provided by most mobile devices.

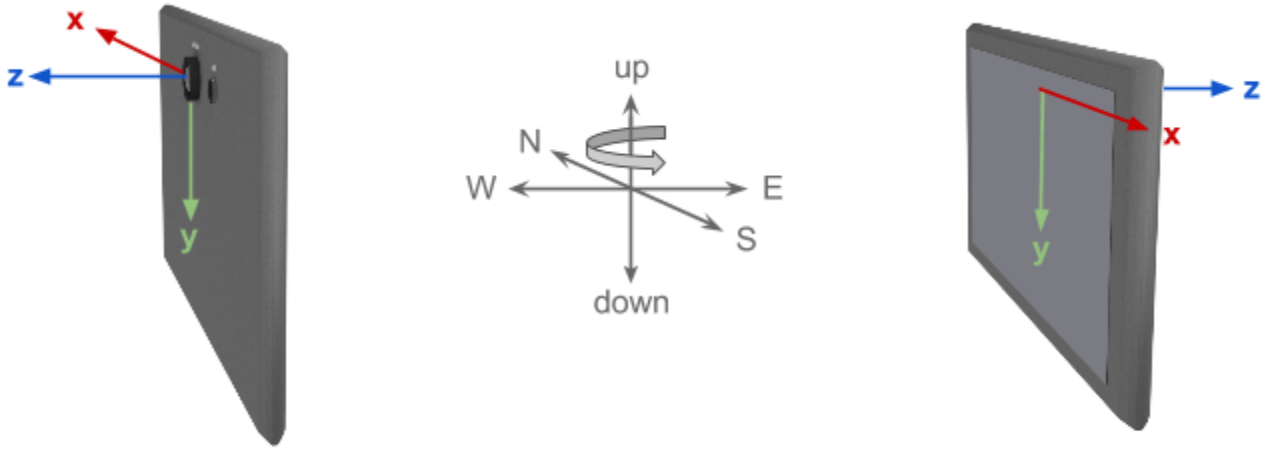
To describe orientation, XDM assigns x, y, and z axes to the camera as shown in the image below, so that when looking into the device's screen (i.e., out through the camera), x increases horizontally to the right, y increases vertically downward, and z increases going forward into the camera's field of view. Thus x and y are coordinates on the image plane, and z is depth (distance from the camera).



View from the screen side of the device, with depth sensor and camera showing through from back.

Orientation is described relative to a *reference orientation*, which is the device in a vertical position and the camera facing east, as if taking a photo of the sunrise. That is, the x axis points south, the

y axis points down, and the z axis points east. In the `DevicePose`, the device orientation is always described relative to this vertical, east-facing "sunrise" reference orientation.



Rotated 180° around y from reference frame

Reference frame "facing sunrise"

As an example, for a photo of a Pacific sunset taken from a beach in Santa Cruz, California, at a time of year when the sun sets directly to the west, the `DevicePose` metadata should specify:

- Position: latitude 36.97° N, longitude 121.95° W, altitude 1.7 meters above sea level.
- Orientation: rotated 180° around the device's vertical axis from the reference orientation, as shown below.

In practice, since the data comes directly from the sensors, it will have a higher accuracy than the numbers in this example.

Screen rotation

Apps that read XDM files are not interested in the orientation of the device, as such, but in the orientation of the camera and the image. Thus XDM does not have a field for "landscape", "portrait", or other information about the rotation of the screen or camera with respect to the device.

Orientation data format

Mathematically, the task of describing orientation or rotation can be difficult and counterintuitive. Each of the popular formalisms for this – rotation matrix, Euler angles, axis-angle representation, and quaternions – has advantages and disadvantages. XDM uses the axis-angle representation, which is intuitive, compact, and well suited for applications that use static image files.

Typically, applications that consume XDM will read and store orientation data in axis-angle format, and use library functions to convert it to matrices for calculations.

About axis-angle representation

According to Euler's rotation theorem, any difference in orientation can be expressed as a single rotation around some axis. Axis-angle representation describes the direction of that axis by giving the x, y, and z coordinate values of a 3D unit vector (a vector whose length, the square root of the quantity $x^2 + y^2 + z^2$, is 1). It also specifies the angle or amount of rotation (θ) around that axis.

The four values x, y, z, and θ are all reals. The direction of rotation and the relative directions of x, y, and z follow the standard right-hand rule.

The device pose element

The `DevicePose` element tells where the device was located and where it was pointing when the photo was taken. (More exactly, it tells the pose of Camera 0, as described below.) This information typically comes from GPS, accelerometer, and magnetometer data at the time the photo is taken.

The one-or-more `CameraPose` elements give the spatial relationships of the cameras and sensors so that depth images and other data can be rectified to the photo. This must come from stored calibration or configuration data.

The device pose and camera poses

It's important to understand that the first camera, "Camera 0", has special status in two ways:

1. **The device pose and the Camera 0 pose are effectively one and the same; they are the pose of the captured image.** To avoid storing redundant data, the Camera 0 `CameraPose` contains all zeroes for pose. (It is still useful for capturing the time of the image capture.) If the IMU and the first camera are well aligned and reasonably close together, then the creating app can write the pose of the IMU (relative to the reference pose) to the `DevicePose`. If not, the app must transform the data to more closely match the true pose of the first camera before writing it to the `DevicePose`. Consuming apps should consider the `DevicePose` to be the pose of Camera 0 and the photograph.

2. **The `CameraPose` elements of additional cameras show those cameras' poses with respect to the first camera.**

Alternative cameras and rotations

This documentation generally assumes that Camera 0 is the device's main or default camera, and that the camera is in its default orientation mode (landscape or portrait). In the images above, for example, the z axis projects from the rear-facing camera on a tablet device in its default landscape orientation, and the device pose is based on those two facts.

There will be cases in which an image is captured from an alternative camera, such as a phone's front-facing camera, and there will be cases in which a camera is not in its default landscape or portrait mode. In these cases, the image-capture application must treat the active camera, in its current mode, as Camera 0. That is, it must transform the device pose appropriately (typically just

an update to the orientation data) and adjust the poses of any additional cameras so that they are relative to the current Camera 0 in its current orientation. If the application stores vendor information with the file, it should use the correct vendor information for the active camera.

Image consumers are not interested in whether the camera that took a photo is a particular device's default camera, or what the camera's default orientation is. The burden is on the image-capture application to be sure that the poses are correct for the currently-active cameras and sensors and the image is written right-side up.

Element definition

The table below shows the components of the `DevicePose` element.

If providing the device's position, all three of the position fields are required. If providing orientation, all four orientation fields are required.

Name	Type	Required	Property Content	If Container Image Modified
Latitude	real	Yes, if providing position.*	WGS84 latitude in degrees	No change
Longitude	real	Yes, if providing position.	WGS84 longitude in degrees	No change
Altitude	real	Yes, if providing position.	WGS84 altitude in meters	No change
RotationAxisX	real	Yes, if providing orientation.	The x component of the rotation vector in the axis-angle representation. Writers of this format should make an effort to normalize [x,y,z] (into a unit vector), but readers should not expect the rotation axis to be normalized.	No change
RotationAxisY	real	Yes, if providing orientation.	The y component of the rotation vector in the axis-angle representation. Writers of this format should make an effort to normalize [x,y,z] (into a unit vector), but readers should not expect the rotation axis to be normalized.	No change
RotationAxisZ	real	Yes, if providing orientation.	The z component of the rotation vector in the axis-angle representation. Writers of this format should make an effort to normalize [x,y,z] (into a unit vector), but readers should not expect the rotation axis to be normalized.	No change
RotationAngle	real	Yes, if providing orientation.	The θ rotation angle in radians in the axis-angle representation.	No change

Camera

The `Camera` element describes a camera or image sensor. (This element is a child of the `Cameras` element, as described below.) A `Camera` includes at least one depth map, image, and/or point cloud. Optional additional child elements are one camera pose, one imaging model, and one vendor information element.

All `Camera` elements are members of an RDF sequence (`rdf:Seq`) called `Cameras`. Camera 0 (the first camera) is associated with the container image. As described under Device Pose, Camera 0 always has the same position and orientation as the device, so for Camera 0 the `CameraPose` element contains zeroes for pose data (but should still be used for the timestamp).

- The namespace URI is `http://ns.xdm.org/photos/1.0/camera/`.
- The default namespace prefix is `Camera`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
DepthMap	DepthMap	Depends on use case		The `DepthMap` property of this `Camera`	If image scaled or cropped, update accordingly
Image	Image	Depends on use case		The `Image` property of this `Camera`	If image scaled or cropped, update accordingly
PointCloud	PointCloud	Depends on use case		The `PointCloud` property of this `Camera`	If image scaled or cropped, update accordingly
ImagingModel	PerspectiveModel	No		The imaging model of this `Camera`, e.g., a `PerspectiveModel`. Currently `PerspectiveModel` is the only type of model supported.	If image scaled or cropped, update accordingly
Pose	CameraPose	Depends on use case		The pose of this `Camera`. If it's the first camera, the pose is relative to the `Device` that owns this `Camera`, and should contain zeros for Pose, plus a timestamp. The poses of additional `Camera` elements are all relative to the first `Camera`, i.e., to the main photograph.	No Change
VendorInfo	VendorInfo	No		Vendor info for this camera	No change

Camera Pose

The `CameraPose` element describes the 3D pose (position and orientation) of a camera with respect to Camera 0, the camera that captured the container image. This allows additional images or data to be rectified to the container image.

- The namespace URI is `http://ns.xdm.org/photos/1.0/camerapose/`.
- The default namespace prefix is `CameraPose`.

Important basic information about poses in XDM is provided under Device Pose.

About camera poses

As described under Device Pose, the camera coordinate system is as follows: The x axis increases horizontally to the right, the y axis increases vertically going down, and the z axis increases along the camera's optical axis, pointing directly away from the camera into the center of its field of view.

Camera 0

The pose of the first camera and the device pose are one and the same. If the device IMU is distant from or not well aligned with Camera 0, then image-capture apps must transform the IMU data to be correct for the first camera. This pose data is captured from the device's sensors at the time of image capture and stored in the `DevicePose` element. The `CameraPose` element for Camera 0 is populated with zeroes and a timestamp.

Additional cameras

The poses of additional cameras specify their position and orientation with respect to the first camera – effectively, their spatial relationship to the container photo. This information is typically available to the image-capture app in a calibration or configuration file on the device. For instance, the device shown here features a depth sensor a few centimeters away from the color camera. Image-capture apps should be able to find the relative coordinates listed in the device config.

A small physical offset between sensors like this can make a significant difference when fusing photo and depth images, especially if the photo is taken at a close distance (say, a few yards or less).

If separate `Camera` elements are present in a file and valid `CameraPose` data is provided, then XDM consumer apps should assume that the additional images are not rectified to the container image, and be prepared to use the pose data and work with the images on that basis. However, some image-capture apps do depth entirely with Camera 0, or may do the extra work of rectifying the additional images before storing them (i.e., adjusting the depth data as if it had been captured at the pose of Camera 0, and cropping both images down to just the overlapping area). In those cases, the app should either

store the `DepthMap` under Camera 0 along with the primary image, or store it under an additional `Camera` with a zero-filled `CameraPose`.

Pose data

All camera positions and orientations are relative to the first camera, which captures the container image. The first camera has zeroes in all fields except for the timestamp.

Position data is stored in meters, and show the distance between the center of the sensor lens and the center of the Camera 0 lens, which should be either found in a device configuration file or manually added to the configuration of the image-capture app. If providing any position data, provide all three position fields and all four rotation fields.

Rotation (orientation) data is stored in axis-angle format, as described under Device Pose. If providing any orientation data, provide all four orientation fields and all three position fields.

The timestamp shows the time of image capture in Epoch time (milliseconds).

Name	Type	Required	Default Value	Property Content	If Container Image Modified
PositionX	real	Depends on use case	0	The x position in meters. For the first camera, this is 0. For additional cameras, this is relative to the first camera.	No change
PositionY	real	Depends on use case	0	The y position in meters. For the first camera, this is 0. For additional cameras, this is relative to the first camera.	No change
PositionZ	real	Depends on use case	0	The z position in meters. For the first camera, this is 0. For additional cameras, this is relative to the first camera.	No change
RotationAxis X	real	Depends on use case	0	The x component of the rotation vector in the axis-angle representation. Writers of this format should make an effort to normalize [x,y,z] (to a unit vector), but readers should not expect the rotation axis to be normalized.	No change
RotationAxis Y	real	Depends on use case	0	The y component of the rotation vector in the axis-angle representation. Writers of this format should make an effort to normalize [x,y,z] (to a unit vector), but readers should not expect the rotation axis to be normalized.	No change
RotationAxisZ	real	Depends on use case	0	The z component of the rotation vector in the axis-angle representation. Writers of this format should make an effort to normalize [x,y,z] (to a unit vector), but readers should	No change

				not expect the rotation axis to be normalized.	
RotationAngle	real	Depends on use case	0	The θ rotation angle in radians in the axis-angle representation.	No change
Timestamp	long	Depends on use case		Time of capture, in milliseconds since the Epoch (January 1 1970, 00:00:00.000 UTC).	No change

Image

The `Image` element describes a base64-encoded image.

- The namespace URI is `http://ns.xdm.org/photos/1.0/image/`.
- The default namespace prefix is `Image`.

One use of the Image element is described under Use case: Depth Photography, in which a backup of the original image is stored along with the matching depth data in the Camera 0 Image, and the container image is treated as a modifiable "presentation" or "display" copy.

For other uses, such as storing an infrared photograph to accompany a normal color photograph, it's better to put the `Image` in a separate `Camera`. It's likely that this approach will correspond to the actual equipment – for instance, an infrared image is taken by a separate IR camera on the same device. The additional `Camera` element should either include accurate pose data for that camera relative to Camera 0, or have no pose data, indicating that the image has already been rectified to the Camera 0 image.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Mime	string	Yes	"image/jpeg"	The mime type for the base64 string containing the image content, e.g., "image/jpeg" or "image/png"	scale/crop: No change
Data	string	Yes	N/A	The base64-encoded image	scale/crop: The data needs to be decoded into an image, then scaled/cropped, then re-encoded again

Perspective Model

The `PerspectiveModel` element describes the imaging model of a perspective image.

- The namespace URI is `http://ns.xdm.org/photos/1.0/perspectivemodel/`.
- The default namespace prefix is `PerspectiveModel`.

The perspective model assumes a standard pinhole camera with 5-DoF radial distortion model.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
FocalLengthX	real	Yes	N/A	The focal length of the lens along the X axis, normalized by the maximum dimension of the sensor. I.e., given the focal length in pixels f_x and the size of the sensor in pixels (*width*, *height*), then: $FocalLengthX = f_x / max(width, height)$	If image cropped, update accordingly
FocalLengthY	real	Yes	N/A	The focal length of the lens along the Y axis, normalized by the maximum dimension of the sensor. I.e., given the focal length in pixels f_y and the size of the sensor in pixels (*width*, *height*), then: $FocalLengthY = f_y / max(width, height)$	If image resized or cropped, update accordingly
PrincipalPointX	real	No	0.5	The x position indicating where the camera optical axis crosses the image plane center of the camera along the X axis, normalized by the sensor width.	If image resized or cropped, update accordingly
PrincipalPointY	real	No	0.5	The y position indicating where the camera optical axis crosses the image plane center of the camera along the Y axis, normalized by the sensor height.	If image resized or cropped, update accordingly
Skew	real	No	0	The skew of the image camera in degrees	
LensDistortion	Sequence of reals (rdf:Seq)	No		$[k_1, k_2, p_1, p_2, k_3]$ where: k_1, k_2, k_3 are radial distortion coefficients, and p_1, p_2 are tangential distortion coefficients, using Brown's distortion model.	

Depth Map

The `DepthMap` element contains a depth map image and information about its creation and format.

- The namespace URI is `http://ns.xdm.org/photos/1.0/depthmap/`.
- The default namespace prefix is `DepthMap`.

A depth map is defined as an image of integer or real values that represent distance from the view point. The exact definition of depth can vary depending on the depth sensor. As an example, two common definitions are depth along the optical axis (typically the Z axis), and depth along the optic ray passing through each pixel. (That is, the distance of an object from the plane perpendicular to the Z axis, versus the distance from the object directly to the camera lens.) The `MeasureType` element specifies which definition is used.

The primary `DepthMap` data associated with the container image must not have any holes.

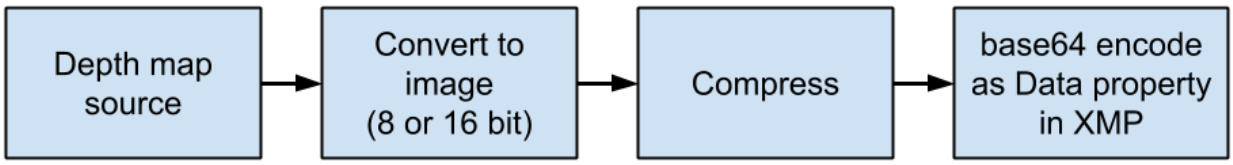
Name	Type	Required	Default Value	Property Content	If Container Image Modified
Format	string	Yes	“RangeInverse”	The format used to encode depth: RangeInverse or RangeLinear	scale/crop: No change
Near	real	Yes	N/A	The near value of the depth map. If metric is set to true, the units are meters. Otherwise the units are undefined.	scale/crop: No change
Far	real	Yes	N/A	The far value of the depth map. If metric is set to true, the units are meters. Otherwise the units are undefined.	scale/crop: No change
Data	string	Yes	N/A	The base64-encoded depth image; see the Appendix on Depth Data for more details. The depth map is stretched to fit the corresponding color image as needed. The reference depth map must not have any holes, as it may be used for image processing by applications.	scale/crop: The data needs to be decoded into an image, scaled/cropped, then re-encoded again
Mime	string	Yes	"image/jpeg"	The mime type for the base64 string describing the depth image content, e.g. “image/jpeg” or “image/png”.	scale/crop: No change

Metric	boolean	No	"false"	Whether the near and far values are expressed in meters. If not set or set to false, the units of depth are unknown (i.e., depth is defined up to a scale). If this value is not set, then some cases such as measurement will not be possible. Valid values are "true" (or 1) and "false" (or 0).	scale/crop: No change
MeasureType	string	No	"OpticalAxis"	The type of depth measurement. Current valid values are "OpticalAxis" and "OpticRay". "OpticalAxis" measures depth along the optical axis of the camera, i.e., the Z axis. "OpticRay" measures depth along the optic ray of a given pixel.	No change
NoiseModel	NoiseModel	No	N/A	The description of the noise model	Update accordingly
Software	string	No	N/A	The software that created this depth map	No change

Depth Data

A depth map is serialized as a set of XMP properties. As part of the serialization process, the depth map is first converted to a traditional image format. The encoding pipeline contains three steps:

1. Convert from the input format (e.g., float or int32 values) to an integer grayscale image format, e.g., bytes (8 bits) or words (16-bit).
2. Compress using a standard image codec, e.g., JPEG or PNG.
3. Serialize as a base64 string XMP property.



The pipeline can be lossless or lossy, depending on the number of bits of the original depth map and the number of bits used to store it, e.g., 8 bits for a JPEG codec and 8 or 16 bits for a PNG codec.

Two different formats are currently supported: **RangeLinear** and **RangeInverse**. **RangeInverse** is the recommended format if the depth map will lose precision when encoded, such as when

converting from float to 8-bit. It allocates more bits to the near depth values and fewer bits to the far values, in a similar way to how the z-buffer works in GPU cards.

If the depth map has an attached noise model, the reliability of the noise model can also be converted to a traditional image format using a pipeline similar to the one used for depth. The noise map is always encoded using the **RangeLinear** format, with the confidence range assumed to be [0, 1].

RangeLinear

Let d be the depth of a pixel, and near and far the minimum and maximum depth values considered. The depth value is first normalized to the [0, 1] range as:

$$d_n = \frac{d - near}{far - near}$$

then quantize to 8 or 16 bits as:

$$\begin{aligned} d_{8bit} &= \lfloor d_n \cdot 255 \rfloor \\ d_{16bit} &= \lfloor d_n \cdot 65535 \rfloor \end{aligned}$$

Conversely, given the quantized depth d_{8bit}, one can recover depth d as:

$$\begin{aligned} d_n &= d_{8bit}/255 \\ d &= d_n \cdot (far - near) + near \end{aligned}$$

RangeInverse

Let d be the depth of a pixel, and near and far the minimum and maximum depth values considered. The depth value is first normalized to the [0, 1] range as:

$$d_n = \frac{far \cdot (d - near)}{d \cdot (far - near)}$$

then quantize to 8 or 16 bits as:

$$\begin{aligned} d_{8bit} &= \lfloor d_n \cdot 255 \rfloor \\ d_{16bit} &= \lfloor d_n \cdot 65535 \rfloor \end{aligned}$$

Conversely, given the normalized depth d_n, one can recover depth d as:

$$d = \frac{far \cdot near}{far - d_n \cdot (far - near)}$$

Noise Model

The `NoiseModel` element contains properties that provide information about the noise properties of a depth sensor.

- The namespace URI is `http://ns.xdm.org/photos/1.0/noisemodel/`.
- The default namespace prefix is `NoiseModel`.

A depth sensor is modeled as a 2-dimensional grid of independent continuous random variables $D(x)$, where $D(x)$ represents the measured depth at pixel x. Let $D^*(x)$ denote the true depth of pixel x. The measurement $D(x)$ is assumed to be contaminated by random noise that is modeled as a mixture of an inlier and outlier process:

$$p(D(x) \mid D^*(x)) = N(D^*(x), \sigma(D^*(x))) * p_{inlier}(x) + U(D_{min}, D_{max}) * (1\text{-}p_{inlier}(x))$$

where $N(D^*(x), \sigma(D^*(x)))$ is a normal distribution centered around $D^*(x)$ with standard deviation $(D^*(x))$, $p_{inlier}(x)$ is the probability of the measurement $D(x)$ being an inlier, and $U(D_{min}, D_{max})$ is a uniform distribution in the range $[D_{min}, D_{max}]$.

For example, let's say a pixel has minDepth 0.1 meters, maxDepth 10 meters, measured depth 2 meters, accuracy 0.3 meters, and reliability 0.2. Then we know there is a 20% chance (.02 reliability) that its measurement is correct (an inlier), with its true depth being a Gaussian distribution N(2 meters, 0.3 meters), i.e., centered around 2 meters with a standard deviation of 0.3 meters. And there is an 80% chance its measurement is wrong (an outlier), in which case its true depth is a uniform distribution in the range of 0.1 meters to 10 meters.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Accuracy	Seq of pairs of reals: (depth, std deviation)	Yes		Inlier model ($D^*(x)$) discretized as a table of pairs , where D is in depth units, e.g., D=1 m, σ(D)=0.1 m D=3 m, σ(D)=0.3 m ...	No change
Reliability	Image	Yes		The probability of a given pixel x being an inlier $p_{inlier}(x)$	The data needs to be decoded into an image, scaled/cropped/rotated, then re-encoded again. This is a non-trivial operation and may significantly alter its accuracy.
MinDepth	real	Yes		Outlier model D_{min} , in depth units	No change

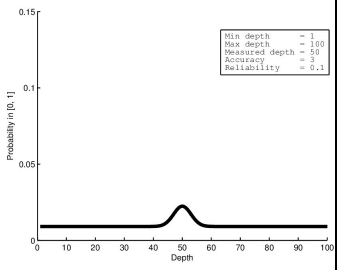
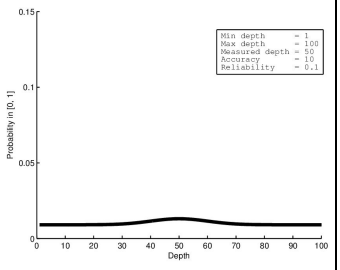
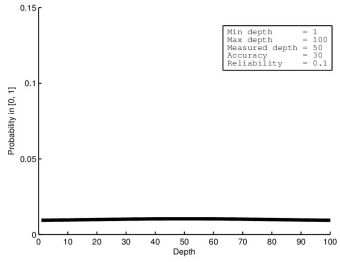
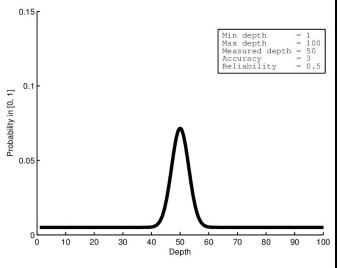
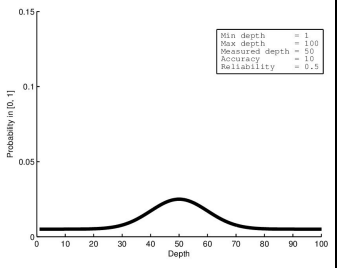
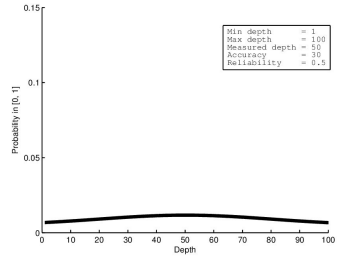
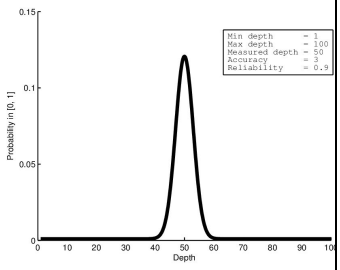
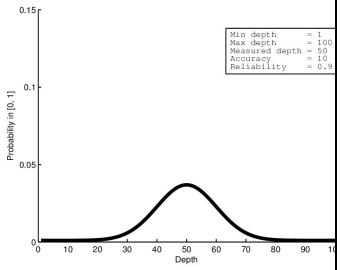
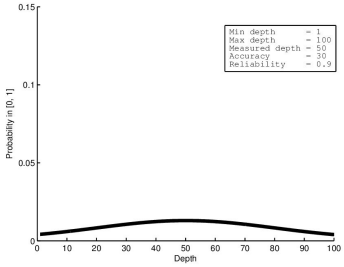
MaxDepth	real	Yes		Outlier model D_{max} , in depth units	No change

Noise model examples

The examples below show the significance of the `Reliability` and `Accuracy` values. In each case, `MinDepth` is 1, `MaxDepth` is 100, and the measured depth is 50 meters. Probability (that the measurement is an inlier) scales up along the Y axis from 0.0 to 0.15, and depth scales up along the X axis from 0 to 100 meters.

As `Reliability` increases with each row going down (from 0.1 to 0.9), we see higher probability at the measured depth. As `Accuracy` increases with each column going across (from 3.0 to 30.0), we see probability distributed across a wider range of depths, but lower peak probability.

The full-size charts are provided in Appendix 2.

Reliability / Accuracy	3.0	10.0	30.0
0.1			
0.5			
0.9			

Point Cloud

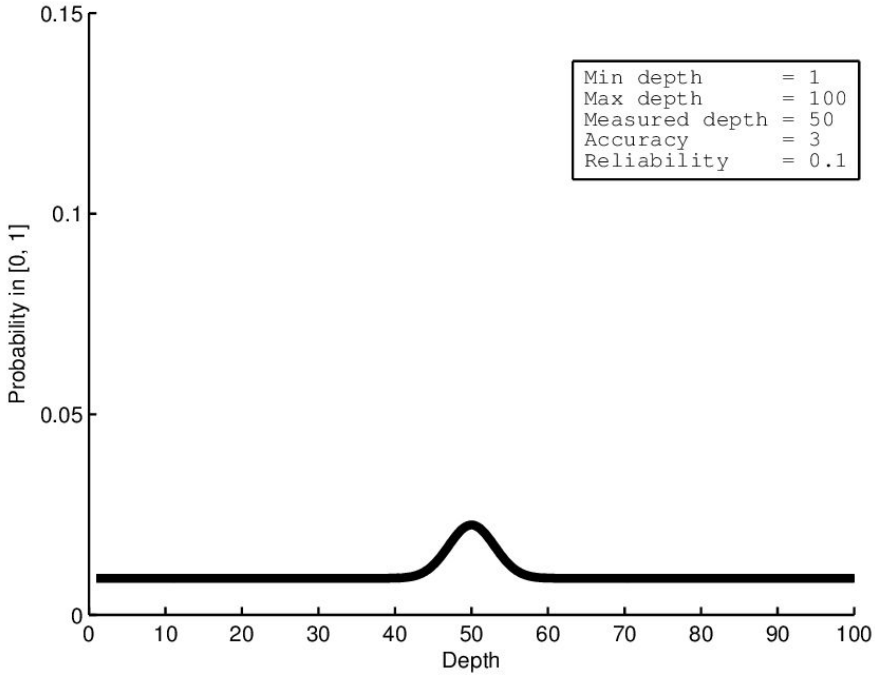
The `PointCloud` element contains properties that provide information regarding the creation and storage of a point cloud.

- The namespace URI is `http://ns.xdm.org/photos/1.0/pointcloud/`.
- The default namespace prefix is `PointCloud`.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Count	integer	Yes	N/A	Number of points (specified by x, y, z triplets) in the data	scale/crop: No change
Position	string	Yes	N/A	Little-endian base64 serialization of a list of x, y, z floating-point triples, using the current camera's coordinate system: [X1, Y1, Z1, X2, Y2, Z2,...]	scale/crop: No change
Color	string	No		Little-endian base64 serialization of a list of R, G, B byte triples: [R1, G1, B1, R2, G2, B2,...]	scale/crop: No change
Metric	boolean	No		Whether the Position values are expressed in meters. If set to false or not set, the units are unknown (i.e., the point cloud is defined up to a scale). If this value is not set, then some cases (such as measurement) will not be possible. Valid values are true (or 1) and false (or 0).	scale/crop: No change
Software	string	No		The software that created this point cloud	No change

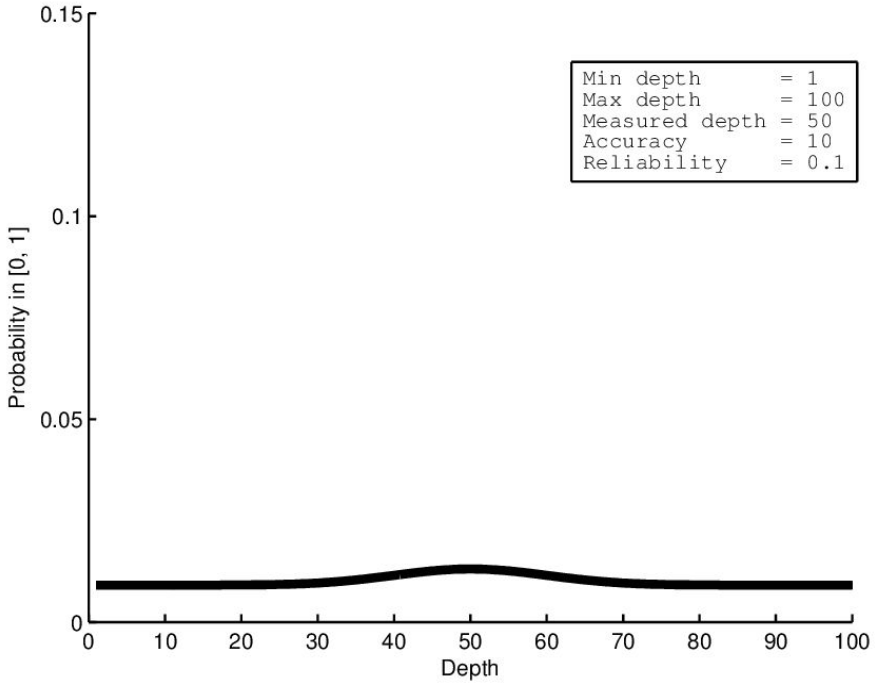
Appendix: Noise Model Examples

Here are larger views of the sample-data chart thumbnails in the Noise Model section.



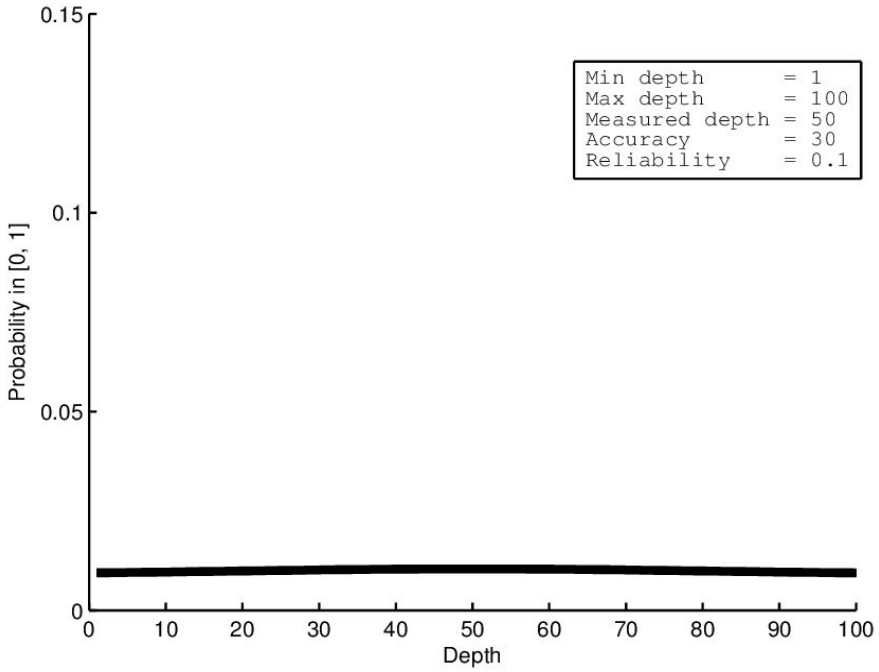
Accuracy: 3

Reliability: 0.1



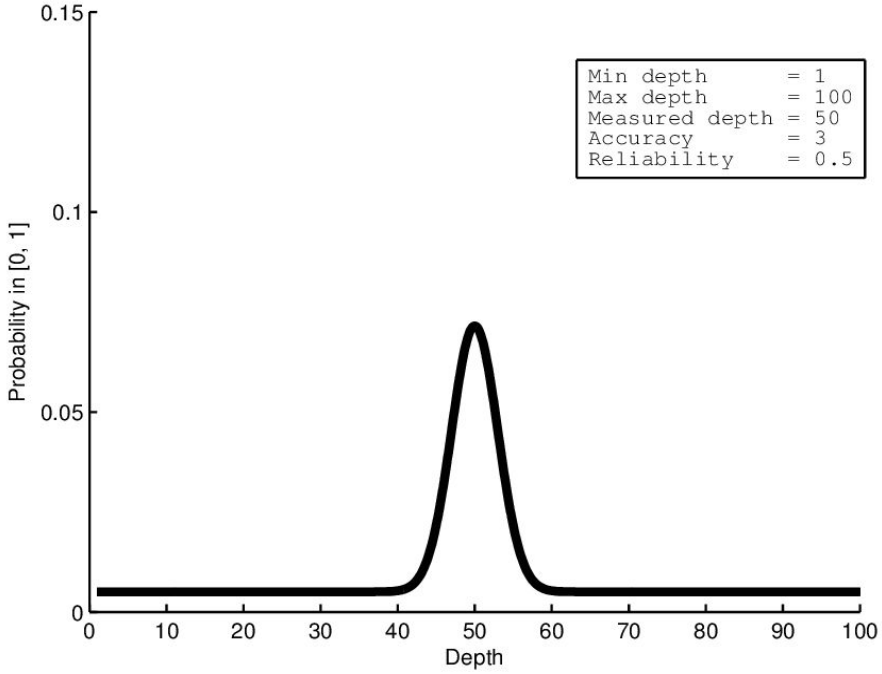
Accuracy: 10

Reliability: 0.1



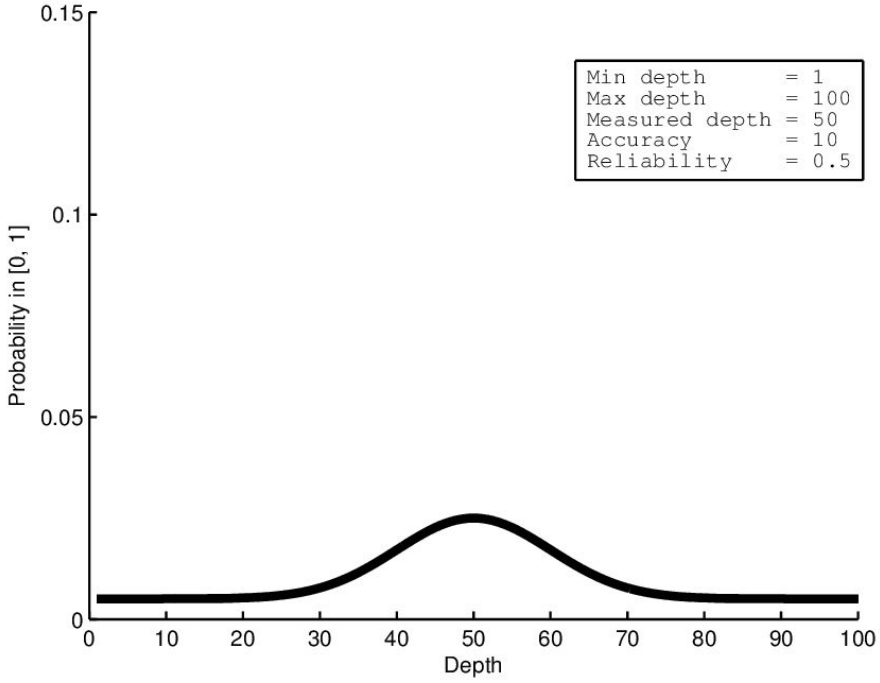
Accuracy: 30

Reliability: 0.5



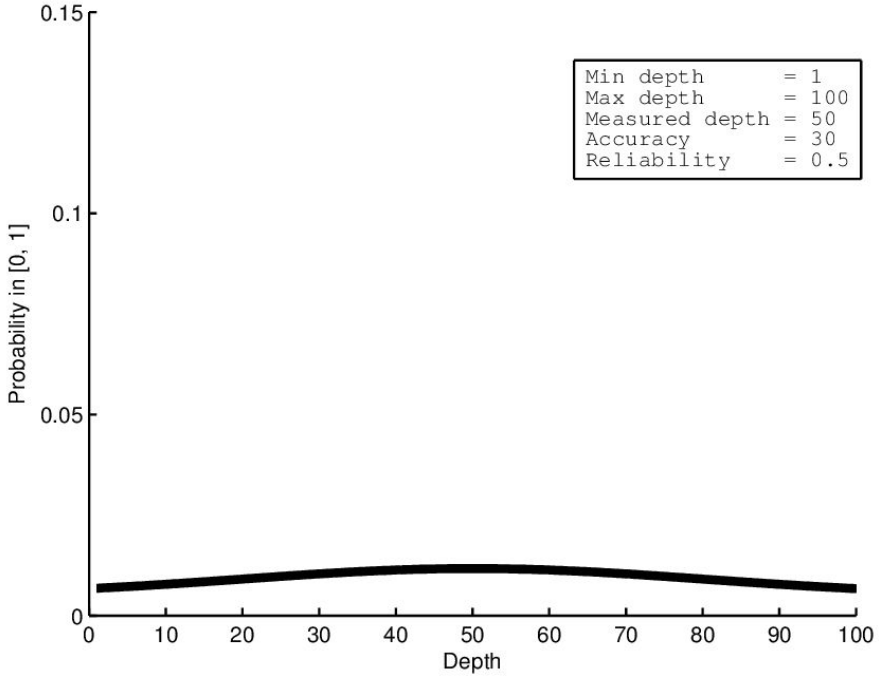
Accuracy: 3

Reliability: 0.1



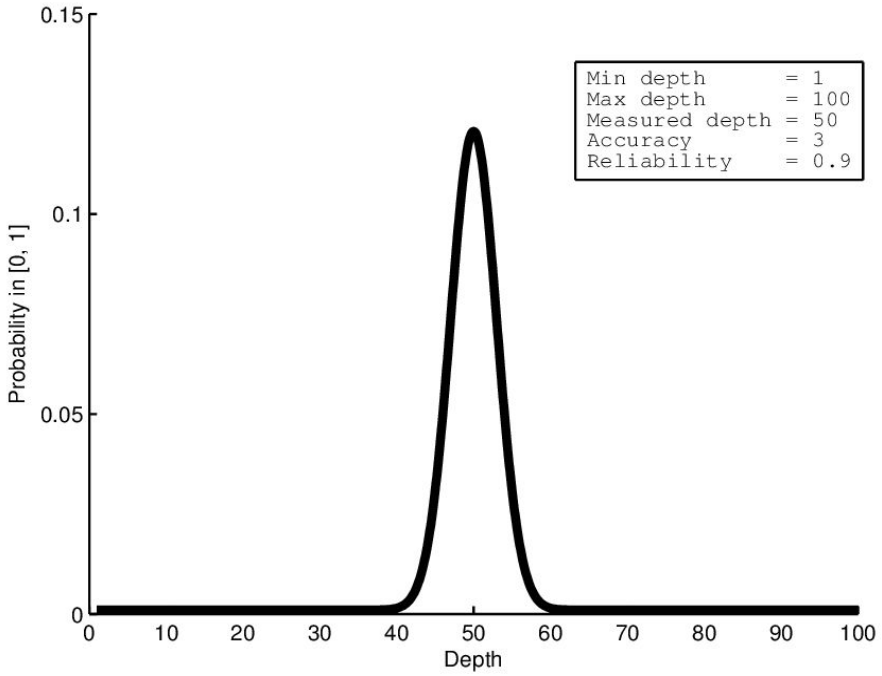
Accuracy: 10

Reliability: 0.5



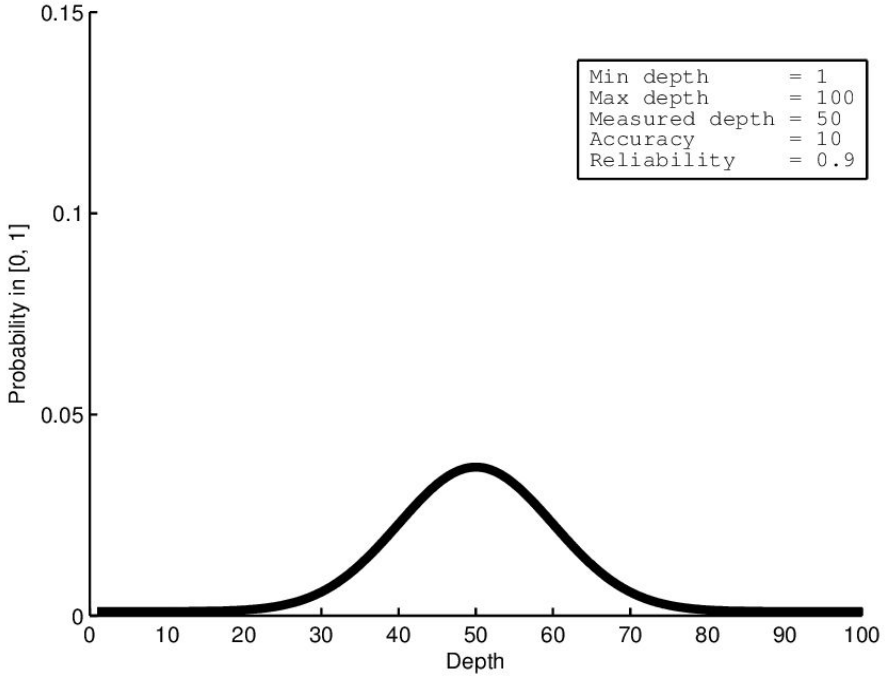
Accuracy: 30

Reliability: 0.5



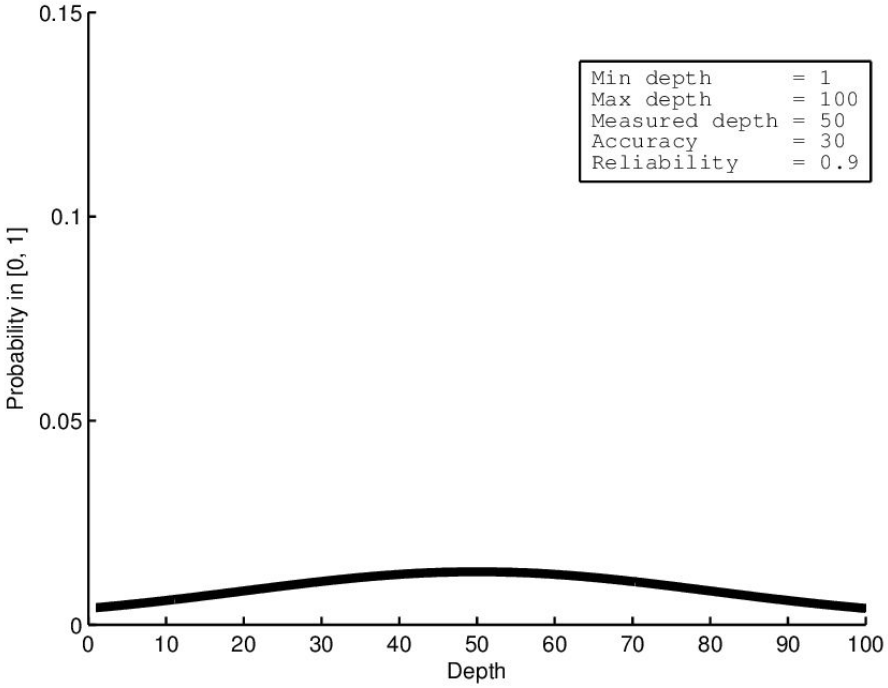
Accuracy: 3

Reliability: 0.9



Accuracy: 10

Reliability: 0.9



Accuracy: 30

Reliability: 0.9