



Extensible Device Metadata

Version 1.01 - DRAFT

Contents

[Overview](#)

[What's new in version 1.01](#)

[Data Structure](#)

[Required Elements](#)

[Versioning](#)

[Namespace Requirements](#)

[Non-standard Usage](#)

[Compatibility](#)

[Profiles](#)

[Profile: Depth Photo](#)

[XDM Elements](#)

[Required](#)

[Profile](#)

[Cameras](#)

[Container Image](#)

[Image and depth map correlation](#)

[Gaps in depth maps](#)

[Profile: Virtual Reality \(VR\) Photo](#)

[XDM Elements](#)

[Required](#)

[Profile](#)

[Cameras](#)

[Poses and Coordinate Systems](#)

[World coordinate system](#)

[Device coordinate system](#)

[Camera coordinate system](#)

[Orientation data format](#)

[About axis-angle representation](#)

[Examples](#)

[Element Definitions](#)

[Device](#)

[Profile](#)

[Vendor Information](#)

[Device Pose](#)

[Camera](#)

[Camera Pose](#)

[Image](#)

[Audio](#)

[Equirect Model](#)

[Fisheye Model](#)

[Perspective Model](#)

[Brown's Distortion](#)

[Mesh Distortion](#)
[Depth Map](#)
 [Depth Data](#)
 [RangeLinear](#)
 [RangeInverse](#)
 [Depth map definition](#)
[Noise Model](#)
 [Noise model examples](#)
 [Noise model definition](#)
[Point Cloud](#)
[Appendix: Noise Model Examples](#)

Except as otherwise noted, the content of this document is licensed under the [Creative Commons Attribution 4.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).

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.

Mira Leung, 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](#), camera [pose](#), lens image 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.


As of version 1.01, XDM supports a variety of use cases including Depth, VR, and 360 photography.



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

What's new in version 1.01

Significant changes in version 1.01 include:

- Products that support the XDM spec are encouraged to display this new icon, available in several formats on the xdm.org website: 
- Added a new **Profile** element for describing the intended use case of the container image with the XDM metadata. For example, the XDM metadata for a depth photo is structurally different from that for a virtual reality photo. This element allows an image reader to quickly identify the media type without having to parse all the metadata, and identify the intended usage of each camera listed in the **CameraIndices** list.
- Added an **EquirectModel** to support rendering for equirectangular panoramas.
- Added a **FisheyeModel** to support rendering for equirectangular panoramas.
- Added **Audio** fields (data and media type) to **Camera** that represent an audio track. An example use case is recording audio with an image, such as with virtual reality photography.
- Deprecated the **ContainerSignature** field in **Device**, originally intended as a required field for detecting whether or not a container image was edited by a non-XDM-aware app. Computation of the signature is non-trivial, and therefore unlikely to be universally implemented.
- Added a **LensDistortionType** field to **PerspectiveModel**, to support other distortion models such as that for a fisheye lens.
- Added a **Timestamp** field to **DevicePose**. This simplifies the access to global capture time information without having to search for a particular **Camera**.

Data Structure

The metadata is serialized and embedded inside the image file as described in the [Adobe XMP](#) standard.

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
 - [Profiles](#) - RDF sequence of one or more [Profile](#) entities
 - Profile - Defines the intended usage(s) of the XDM metadata with the container image.
 - [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
 - [Audio](#) - Audio provided by the camera
 - [Image](#) - Image provided by the camera
 - [ImagingModel](#) - Imaging (lens) model. The currently allowed types are: [EquirectModel](#), [PerspectiveModel](#), and [FisheyeModel](#).
 - [DistortionModel](#) - Lens distortion model. The currently supported types are [BrownsDistortion](#) and [MeshDistortion](#).
 - [DepthMap](#) - Depth-related information including the depth map and noise model
 - [NoiseModel](#) - Noise properties for the sensor
 - [PointCloud](#) - Point-cloud data

Required Elements

The [Device](#) element is always required, since it is the root of the object. The inclusion of other elements depend on the usage requirements. Each profile or use case will have its own set of required data. Profiles are used to help applications identify what use cases a given XDM file can support. Multiple profiles can be supported by a single file. Applications are not expected to support all profiles.

Versioning

XDM uses a loosely-restricted versioning system that does not change major versions for minor enhancements that 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., 1.01, 1.02, 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 when using a JPEG container. 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.

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 backwards compatible since they use different XML namespaces, but they can co-exist.

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.

Profiles

A profile describes the intended use case of an XDM image. This allows image parsers to quickly identify that usage without going through the entire metadata structure and clears up ambiguity with respect to the purpose of each Camera. The `Profile` element contains a profile's name and the indices of cameras used. Currently-supported use cases are depth photos and VR photos.

The expected XDM elements for each profile are outlined below.

Profile: Depth Photo

XDM Elements

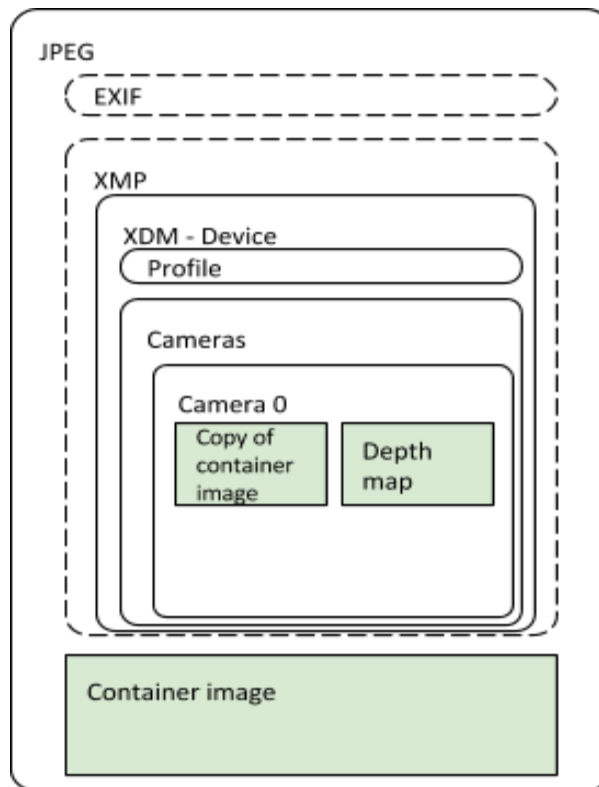
Required

Profile

- `Profile:Type` must be set to `DepthPhoto`.
- `Profile:CameraIndices`. This list must have only one integer i , which represents the i th camera in the `Device:Cameras` list.

Cameras

- Camera i
 - `DepthMap`
 - `Image` (optional if $i \neq 0$)



XDM file for depth photography

Container Image

In depth photography, the container image is the "presentation" or "display" copy of the image. Note that the image is not required if the camera index is 0 and the image and container image are identical to avoid redundancy.

Image and depth map correlation

The camera image and depth map in a given **Camera** 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. Since they may not have the same number of pixels, the aspect ratio may be slightly off, and apps should be able to handle that.

If a use-case requires that you save all of the captured image and depth data, then in addition to the easy-to-consume 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.

Profile: Virtual Reality (VR) Photo

A virtual reality (VR) photo is a stereoscopic 360° panorama and may contain an audio track and geolocation information. An example of VR photos are those captured with Cardboard Camera.

XDM Elements

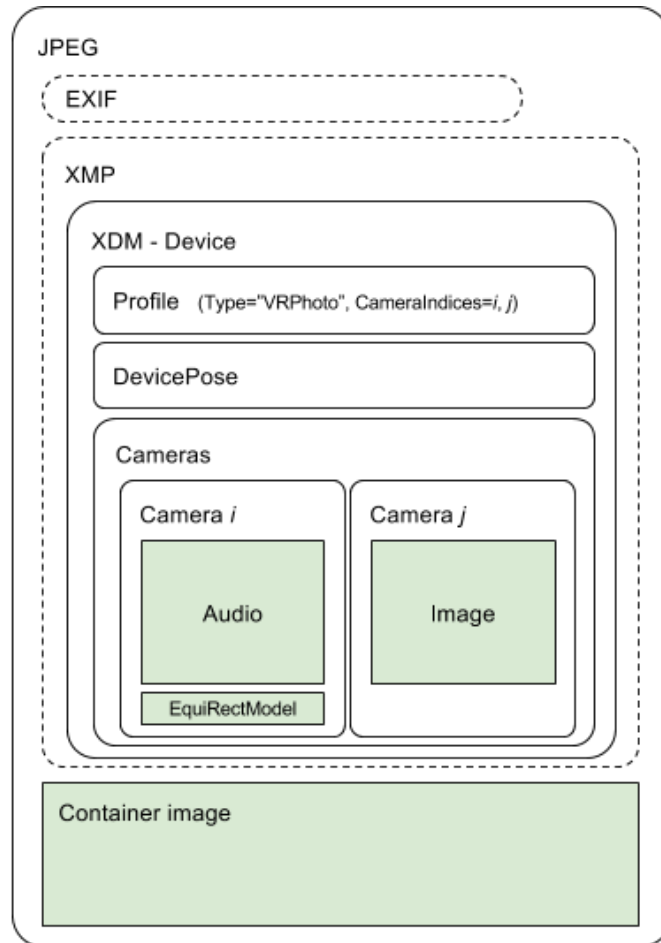
Required

Profile

- **Profile:Type** must be set to **VRPhoto**.
- **Profile:CameraIndices**. This list must have exactly two integers (i, j), which represent the i th and j th cameras in the **Device:Cameras** list.

Cameras

- **Camera i** (left eye)
 - **EquirectModel**
 - **Image** (optional if $i \neq 0$)
 - **Audio** (optional)
- **Camera j** (right eye)
 - **Image**



A typical XDM file for a VR photo use case

Poses and Coordinate Systems

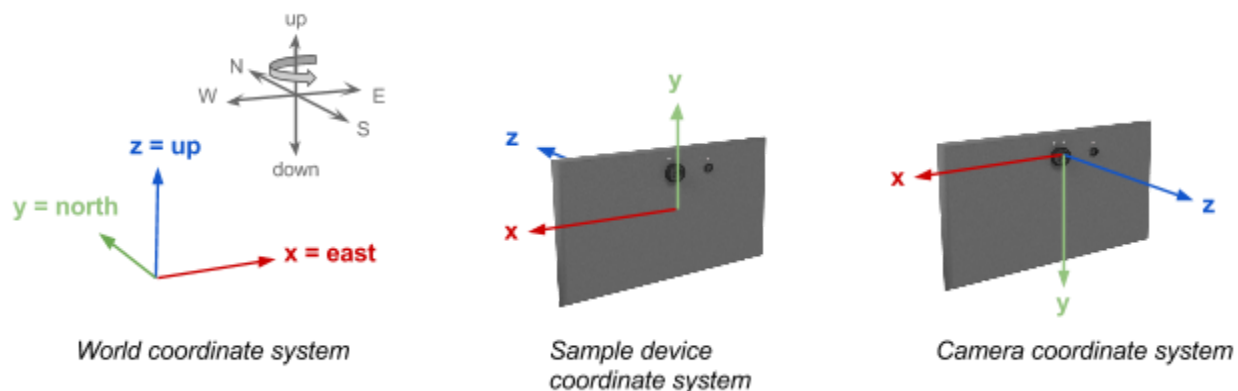
XDM stores the [pose](#) (that is, the *position* and *orientation*) of the device relative to the world, and of the camera(s) relative to the device. This enables applications to use multiple images together, as when mapping depth data onto a photograph, and provides information about the image capture, such as the position and orientation of the photographer.

For simple devices and applications, the device pose is arguably not needed; the creator app could simply store the pose of Camera 0 relative to the world, and the poses of other cameras relative to that. But as more complex devices and applications emerge (such as camera arrays and 360-degree photography), the device pose provides a robust and intuitive frame of reference.

Position is expressed in three dimensions. For the device pose, these are latitude, longitude, and altitude. For the camera pose, they are the distance in meters from the device origin point along the camera's x, y, and z axes.

Orientation is also expressed in three dimensions, as a rotation around x, y, and z axes relative to a frame of reference. For the device, the frame of reference is a standard "ENU" world coordinate system, described below. For a camera, the frame of reference is the device coordinate system.

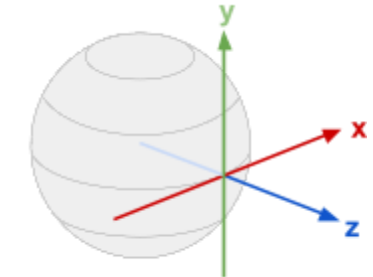
Each of these 3D [coordinate systems](#) has a defined origin from which x, y, and z axes emerge in defined directions. Different devices may have different systems. For instance, on a smartphone the origin of the coordinate system is typically at the center of the screen, but not all devices will have screens. On a 360-degree camera, the device origin is likely to be the center of the array. On a planar camera array, the device origin might be the center of the [0,0] camera. The image-creation app knows the origins and axes for the device that it uses to capture the image, and uses that information to generate the pose values. The consumer app doesn't need to know the device coordinate system; it just uses the pose values to transform the data as needed.



World coordinate system

XDM uses a right-handed, east-north-up ([ENU](#)) world coordinate system. This is the same world coordinate system used in the [Android](#) and [iOS](#) operating systems, and in [Tango](#) visual-inertial devices. This system is defined as follows.

- The origin is the location specified by latitude, longitude, and altitude.
- X is tangential to the ground at that location and points roughly East. (It is the vector cross product $\mathbf{y} \times \mathbf{z}$.)
- Y is tangential to the ground at that location and points towards the North Pole.
- Z is perpendicular to the ground at that location and points towards the sky.



World coordinate system

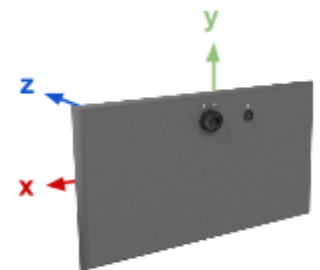
The 3D position is represented in [WGS84](#) coordinates as longitude, latitude, and altitude. In keeping with the WGS84 documentation, altitude is height in meters above the standard spheroidal reference surface, and latitude is [geodetic latitude](#). This is consistent with the GPS data provided by most mobile devices.

Device coordinate system

The device's coordinate system may depend on the device. It must have an origin (0,0,0) and x, y, and z axes. The consumer app doesn't need to know the device coordinate system, as long as the camera pose data correctly transforms data from the device coordinate system to the camera's.

For an example, consider the [Android sensor coordinate system](#) and [Tango's device coordinate system](#). When the Android device is held in its default orientation:

- The origin (0,0,0) is the center of the screen.
- X is horizontal and points to the right.
- Y is vertical and points up.
- Z points outward from the screen face. The back of the device faces the negative Z axis.



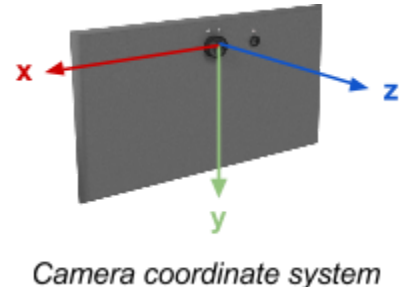
Sample device coordinate system

Note that different devices may default to portrait or landscape orientation, and many applications can rotate images when the screen is rotated. The device's coordinate system is always attached to the physical device as it rotates, so **DevicePose** **does not account** for the possible rotation of the image pixels as the device changes from portrait orientation to landscape. This rotation should be handled by **CameraPose** element.

Camera coordinate system

The camera coordinate system is the same for all cameras, regardless of the device. Looking out from the camera into the field of view:

- The origin is at the center of the field of view, in the center of the lens.
- X is aligned with the x-axis of the image, i.e. it aligns with rows of pixels.
- Y is aligned with the y-axis of the image, i.e. it aligns with columns of pixels.
- Z points outward along the camera's [optical axis](#), pointing directly away from the user into the center of the camera's field of view.



The pose of each camera specifies its position relative to the device origin, and its rotation relative to the device coordinate system. In the example device shown, Camera 0 is a few centimeters along the y axis from the device origin, and the axes are rotated 180° around the x axis. The example also has a depth sensor a few centimeters horizontally from Camera 0. This is Camera 1; its position is a few centimeters along the x and y axes from the device origin, and it has the same orientation as Camera 0.

Information on camera positioning is available from the device manufacturer, or possibly in a configuration file on the device. It may be tempting in a simple single-camera application to use zeroes for the camera positions, implying that the camera is at the device origin. But obviously this would not work in an application that uses multiple cameras, such as depth photography, especially if the image is captured at a close distance (say, a few yards or less).

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.

Applications that consume XDM might typically 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](#).

Examples

To record the camera pose with respect to the world, on a typical smartphone **the creator application must:**

1. Retrieve the device orientation with respect to the world, which returns a 3x3 matrix. Convert this to axis-angle format, and store that as the **DevicePose** orientation.
2. Retrieve the latitude, longitude, and altitude from the sensors. Store these as the **DevicePose** position.
3. Using the manufacturer's data on the position of each camera relative to the device origin, store the relative position in the **CameraPose**.
4. Using the manufacturer's data on the camera orientation, store the transform matrix in axis-angle format as the orientation data in the **CameraPose**.

To determine the camera pose relative to the world, if needed for a user application, on a typical smartphone **the consumer device must:**

1. Convert the stored device position from axis-angle to a 4x4 matrix, with 1 for each position value. Convert the stored **CameraPose** from axis-angle to a 4x4 matrix.
2. Multiply these matrices to obtain the camera's orientation relative to the world, and its position relative to the device origin.
3. Add the specified latitude, longitude, and altitude to the relative position (which is usually too small to be significant). This gives you the camera's position relative to the world.

The creator app needs to know the coordinate system of the capture device in order to generate poses. The consumer app usually does not need to know this; it just needs the poses to perform tasks such as:

- Orient multiple images to each other using the camera poses with the device as a frame of reference. (For instance, with depth photography or camera arrays.)
- Tell the user where a photograph was taken and which way the camera was facing. (For instance, to add photos to map data or create a visual tour guide.)

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 ("1.01", "1.21", 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
Container Signature	string	No DEPRECATED in v1.01 <i>Note: had been required in 1.0.</i>	N/A	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
Profiles	Sequence of Profile (rdf:Seq)	No	N/A	Describes the intended purpose(s) of the image and its metadata. If the fields for more than one use case are present, all the applicable profiles should be listed.	No change
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., position on the Earth and orientation relative to gravity and magnetic north.	No change
VendorInfo	VendorInfo	No		Vendor information for the device	No change

Profile

The **Profile** element describes vendor information for a camera or a device.

- The namespace URI is <http://ns.xdm.org/photos/1.0/profile/>.
- The default namespace prefix is **Profile**.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Type	string	Yes		One of DepthPhoto or VRPhoto	No change
CameraIndices	Sequence of integers (rdf:Seq)	Depends on use case	N/A	Indicates the cameras that will be used in the profile. See the respective profile description for the intended use of each camera.	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 [pose](#) (i.e., *position* and *orientation*) of the device that captured the image with respect to the world coordinate system.

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

The raw data used to determine the device pose typically comes from GPS and IMU sensors. See [Poses and Coordinate Systems](#) for important notes on how XDM represents position and orientation data, and how the world, device, and camera coordinate systems work together.

When providing the position, all three of the position fields are required, and when providing the orientation, all four orientation fields are required.

Position data shows location on the Earth. **Rotation (orientation) data** shows the device orientation relative to the ENU world coordinate system, in [axis-angle](#) format as described under [Poses and Coordinate Systems](#).

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

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	No change

			should make an effort to normalize [x,y,z] (into a unit vector), but readers should not expect the rotation axis to be normalized.	
RotationAngle	real	Yes, if providing orientation.	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

Camera

The **Camera** element describes a camera or image sensor. (This element is a child of the **Cameras** element.) A **Camera** includes at least one audio, depth map, image, and/or point cloud.

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

If there are multiple cameras, the sequence of **Camera** elements is important. The first **Camera** that appears in the data structure is referred to in this documentation as Camera 0. In a simple case (e.g., a smart phone or tablet), Camera 0 is always the camera that captured the container image; the rest of the sequence is arbitrary. In the case of a complex device such as a 360-degree camera or planar camera array, Camera 0 is at the discretion of the creator app, but it's recommended to use the manufacturer's designated "primary" or "first" camera if one exists.

All images, depth maps, and point clouds in a single **Camera** element are presumed to be rectified to each other, i.e., have the same pose, proportions, and field of view. Any additional **Camera** element should either include accurate pose data for that camera relative to the device, or have no pose data, indicating that its images have already been rectified to the Camera 0 image. If multiple **Camera** elements are present in a file and have differing **CameraPose** elements, then XDM consumer apps should assume that the additional images are not [rectified](#) to the container image, and be prepared to work with the images on that basis.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Audio	Audio	Depends on use case		The Audio property of this Camera	No Change
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 or EquirectModel	No		The imaging model of this Camera. Currently PerspectiveModel and EquirectModel are the only supported model types.	If image scaled or cropped, update accordingly
Pose	CameraPose	Depends on use case		The pose of this Camera. The poses of Camera elements are all relative to the device.	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 the device.

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

See [Poses and Coordinate Systems](#) for important notes on how XDM represents position and orientation data, and how the world, device, and camera coordinate systems work together.

The position and orientation of each camera relative to the device are based on information provided by the manufacturer. Image-creation apps need this information in order to create the XDM file. Image consumers do not need this information; they just need the pose data.

If it is not possible to know the camera pose relative to the device, both are assumed to have the same pose, and the **CameraPose** is set to identity (no difference).

Position data shows the x, y, z distance between the center of the camera lens and the device origin, in meters. **Rotation (orientation) data** shows the camera orientation relative to the device orientation, in [axis-angle](#) format as described under [Poses and Coordinate Systems](#).

When providing the position, all three of the position fields are required, and when providing the orientation, all four orientation fields are required.

The table below shows the components of the **CameraPose** element.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
PositionX	real	Yes, if providing position.	0	The x position in meters, relative to the device.	No change
PositionY	real	Yes, if providing position.	0	The y position in meters, relative to the device.	No change
PositionZ	real	Yes, if providing position.	0	The z position in meters, relative to the device.	No change
RotationAxisX	real	Yes, if providing orientation.	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

RotationAxisY	real	Yes, if providing orientation.	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	Yes, if providing orientation.	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 not expect the rotation axis to be normalized.	No change
RotationAngle	real	Yes, if providing orientation.	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 the device, 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	N/A	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

Audio

The **Audio** element describes a base64-encoded audio file.

- The namespace URI is <http://ns.xdm.org/photos/1.0/audio/>.
- The default namespace prefix is **Audio**.

The Audio element can be used if a camera records sound along with its captured image, and the sound is to be played back when the image is viewed. Another use case is when there is a separate microphone array in addition to a set of cameras. The recorded sound from each microphone would be represented in a distinct Camera element.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Mime	string	Yes	N/A	The mime type for the base64 string containing the audio content, e.g., "audio/mp4" or "audio/mpeg3"	No change
Data	string	Yes	N/A	The base64-encoded audio data	No change

Equirect Model

The `EquirectModel` element describes an [equirectangular](#) imaging model for a panoramic or spherical photo.

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

This model is adapted from the [Google PhotoSphere XMP metadata](#) specification.

The imaging model assumes a standard camera with no distortion model.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
CroppedAreaLeft Pixels	integer	Yes	N/A	Column where the left edge of the image was cropped from the full-sized image.	Rescale if the size of the container image has changed.
CroppedAreaTop Pixels	integer	Yes	N/A	Row where the top edge of the image was cropped from the full-sized image.	Rescale if the size of the container image has changed.
CroppedAreaImageWidthPixels	integer	Yes	N/A	Original width in pixels of the image. Equal to the actual image's width for unedited images.	Change to match the container image's width
CroppedAreaImageHeightPixels	integer	Yes	N/A	Original height in pixels of the image. Equal to the actual image's height for unedited images.	Change to match the container image's height
FullImageWidthPixels	integer	Yes	N/A	Original full width from which the image was cropped. If only a partial photosphere was captured, this specifies the width of what the full photosphere would have been.	Rescale if the size of the container image has changed.
FullImageHeightPixels	integer	Yes	N/A	Original full width from which the image was cropped. If only a partial photosphere was captured, this specifies the width of what the full photosphere would have been.	Rescale if the size of the container image has changed.

Fisheye Model

The `FisheyeModel` element describes the imaging model of a camera with a [fisheye lens](#). This follows the [OpenCV fisheye model](#).

- The namespace URI is <http://ns.xdm.org/photos/1.0/fisheyemodel/>.
- The default namespace prefix is `FisheyeModel`.

The imaging model assumes that a fisheye distortion model will be used.

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 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	
PixelAspectRatio	real	No	1.0	The aspect ratio of the X scale factor over the Y scale factor.	
DistortionModel	BrownsDistortion or MeshDistortion	No		The lens distortion model. Currently <code>BrownsDistortion</code> and <code>MeshDistortion</code> are the only supported types.	

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 imaging 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 .	

Brown's Distortion

The `BrownsDistortion` element describes the [Brown-Conrady lens distortion model](#).

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

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Parameters	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.	No change

Mesh Distortion

The **MeshDistortion** element describes the distortion model used by the Ricoh Theta fisheye lens.

- The namespace URI is <http://ns.xdm.org/photos/1.0/meshdistortion/>.
- The default namespace prefix is **MeshDistortion**.

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Width	integer	Yes	N/A	The row width of the core distortion values' matrix.	No change
Height	integer	Yes	N/A	The height of the core distortion values' matrix.	No change
Grid	Sequence of reals (rdf:Seq)	No		The values in the mesh grid, in a single array. Consumer apps should use the above parameters to convert this back into a matrix.	No change

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.

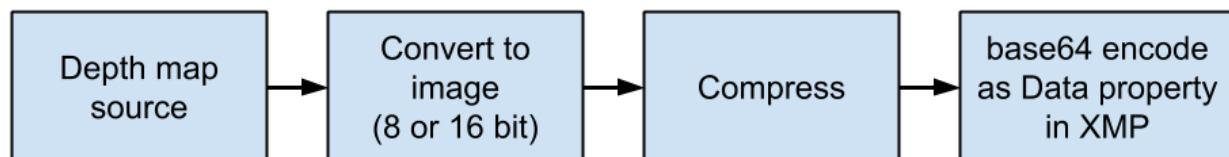
The primary **DepthMap** data associated with the container image must not have any holes.

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). These apps should either store the **DepthMap** under Camera 0 along with the primary image.

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:

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

Depth map definition

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Format	string	Yes	N/A	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 and image are cropped to have the same aspect ratio as needed. The reference depth map must not have any holes, as it may be used for image processing by applications.	Scale: No change as long as aspect ratios match. Crop: Decode data into an image, crop to matching ratio, then re-encode.
Mime	string	Yes	N/A	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

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:

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

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

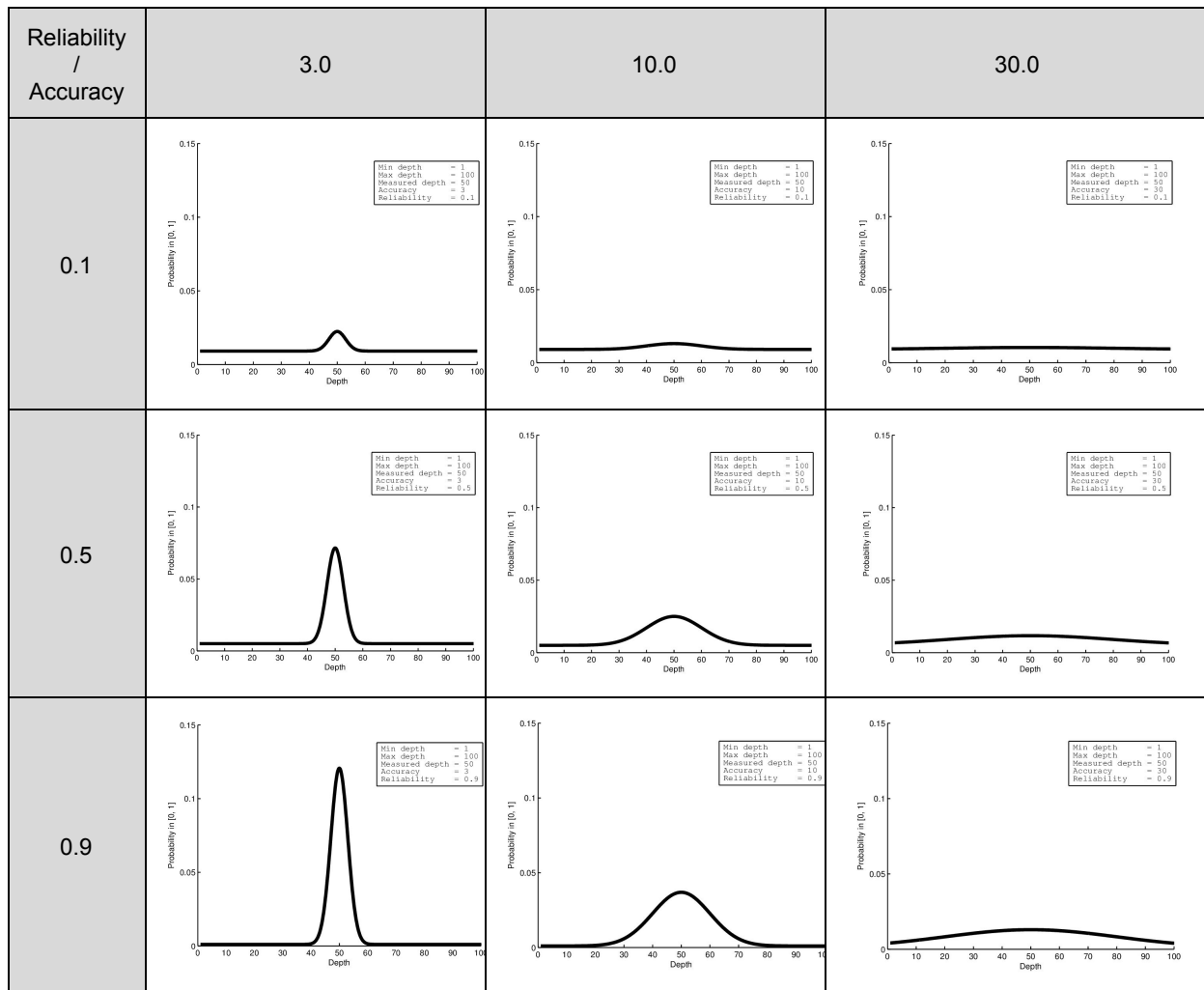
For example, let's say a pixel has minDepth 0.1 meters, maxDepth 10 meters, measured depth 2 meters, accuracy 0.3 meters, and reliability 0.2. Then we know there is a 20% chance (.02 reliability) that its measurement is correct (an inlier), with its true depth being a Gaussian distribution $N(2 \text{ meters}, 0.3 \text{ meters})$, 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.

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.

Please refer to the charts in [Appendix 2](#).



Noise model definition

Name	Type	Required	Default Value	Property Content	If Container Image Modified
Accuracy	RDF:Seq of pairs of reals (depth, std deviation) in pair-major order, e.g., d_0, s_0, d_1, s_1	Yes	N/A	Inlier model ($D^*(x)$) discretized as a table of pairs, where D is in depth units, e.g., D=1 m, $\sigma(D)=0.1$ m D=3 m, $\sigma(D)=0.3$ m ...	No change
Reliability	Image	Yes	N/A	The probability of a given pixel x being an inlier $p_{\text{inlier}}(x)$	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	N/A	Outlier model D_{\min} , in depth units	No change
MaxDepth	real	Yes	N/A	Outlier model D_{\max} , in depth units	No change

Point Cloud

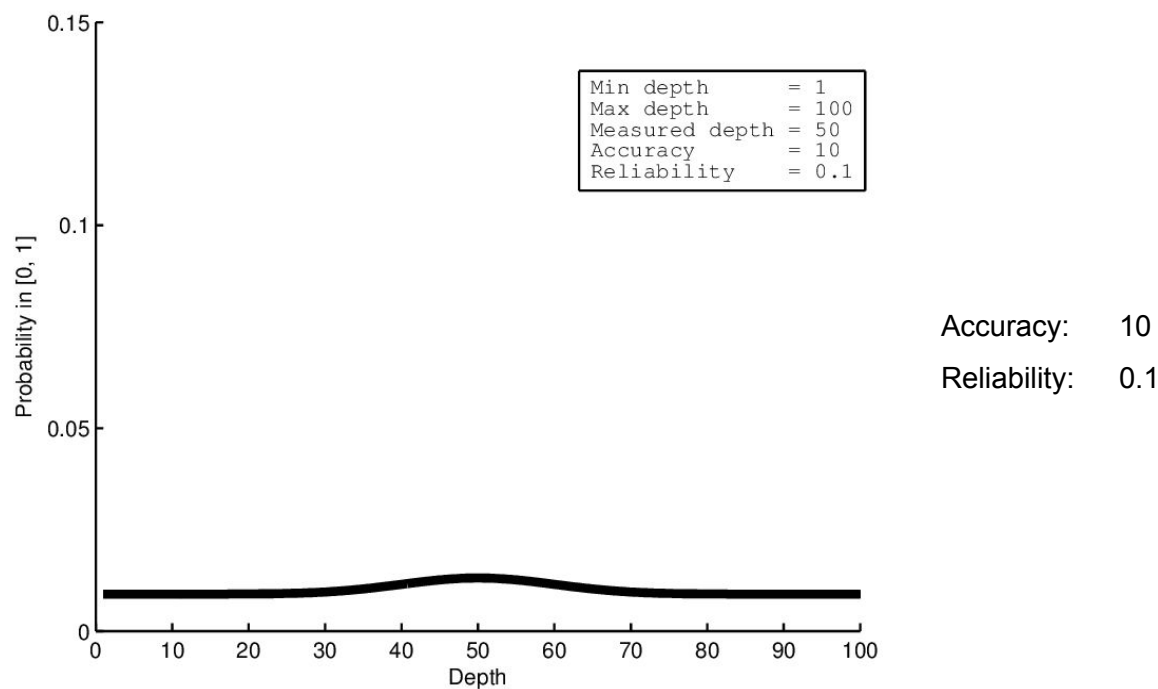
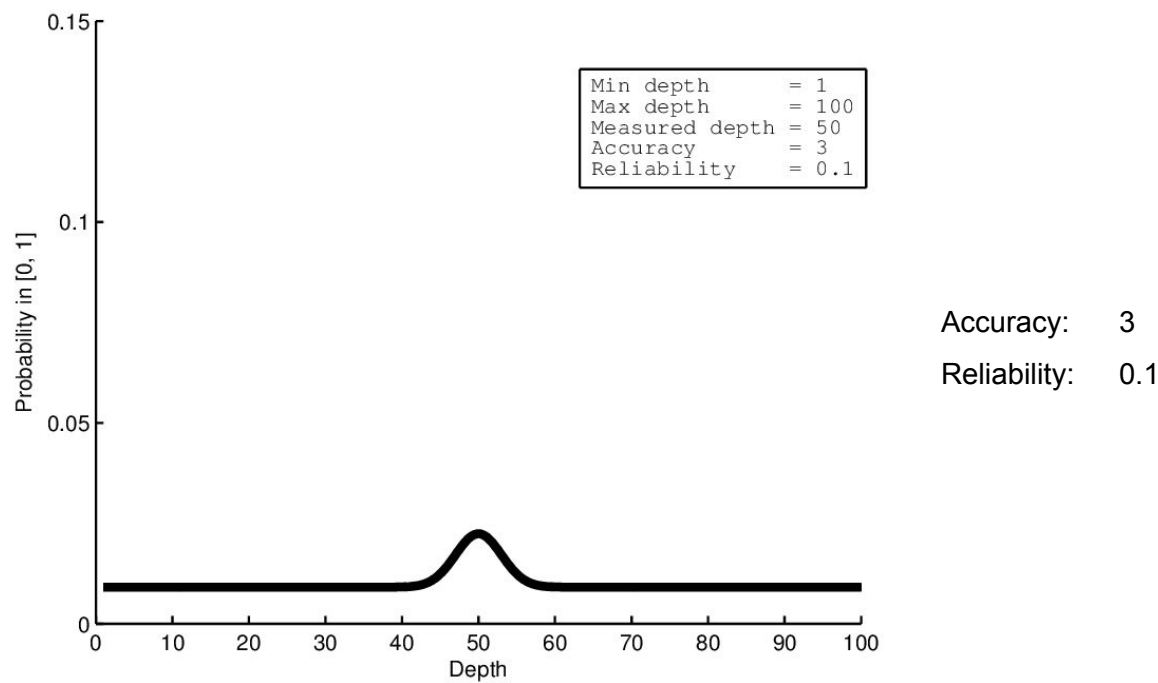
The **PointCloud** element contains properties that provide information regarding the creation and storage of a point cloud.

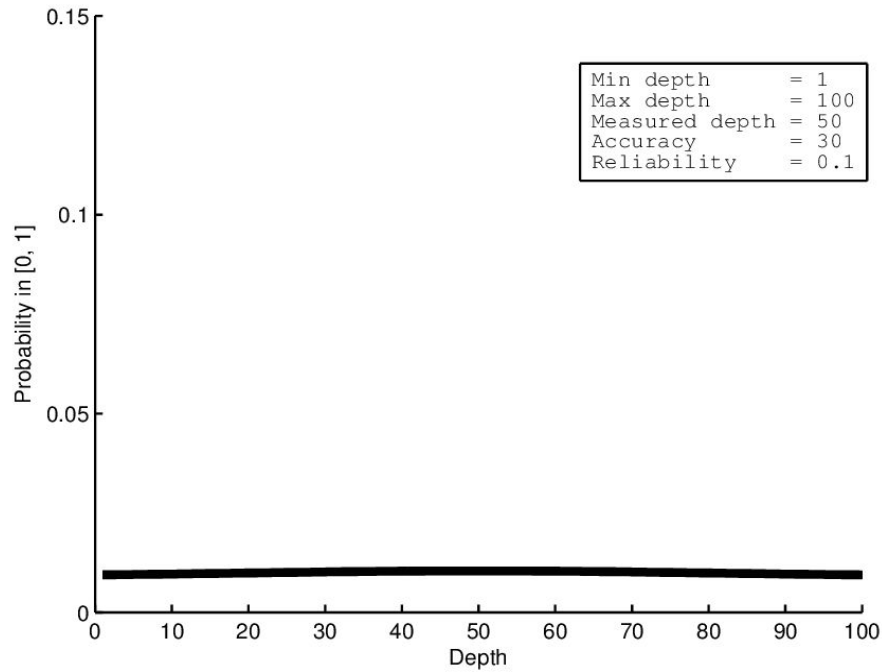
- The namespace URI is <http://ns.xdm.org/photos/1.0/pointcloud/>.
- The default namespace prefix is **PointCloud**.

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

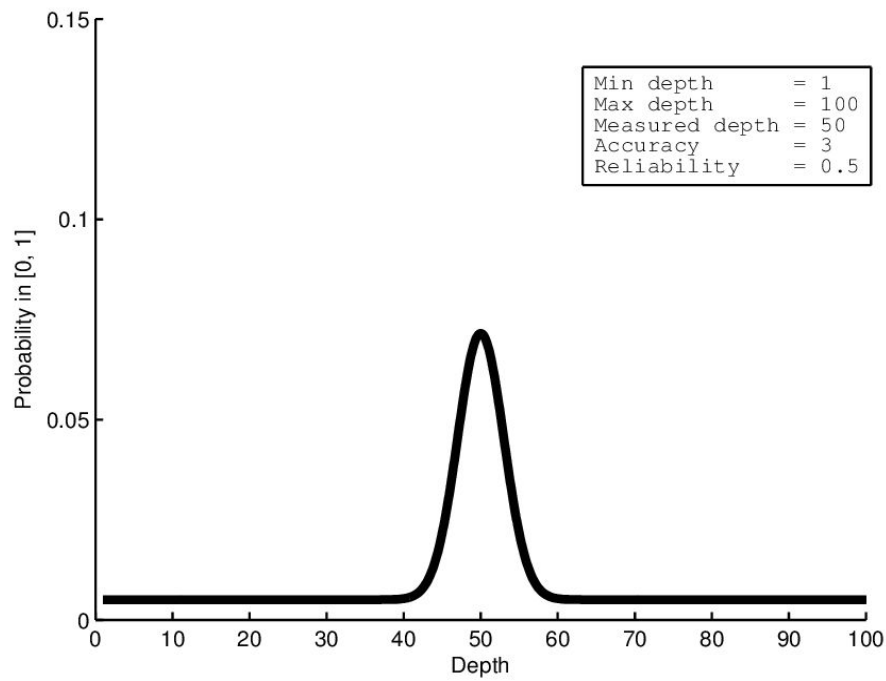
Appendix: Noise Model Examples

Here are larger views of the sample-data chart thumbnails in the [Noise Model](#) section.

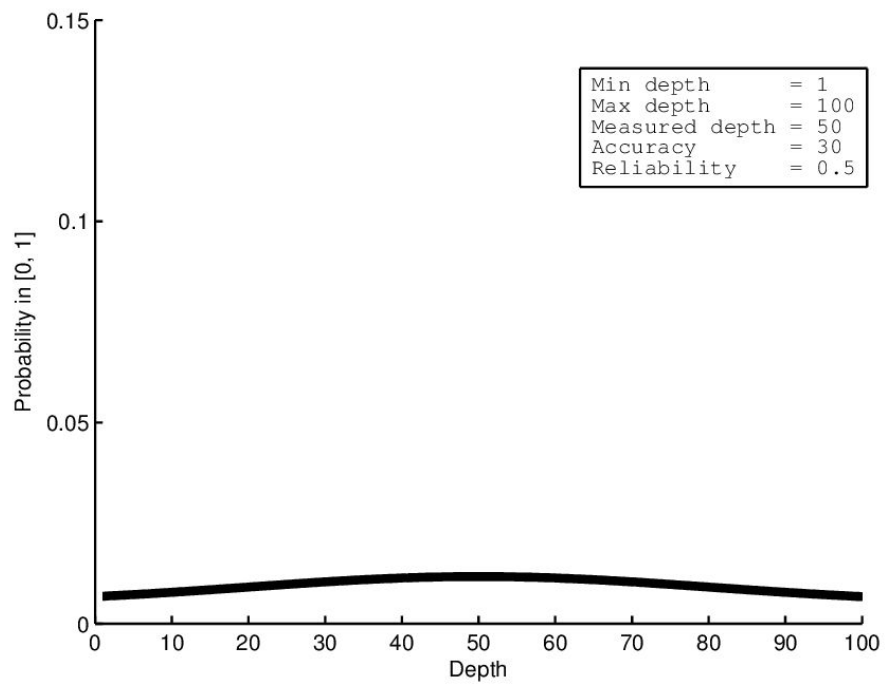
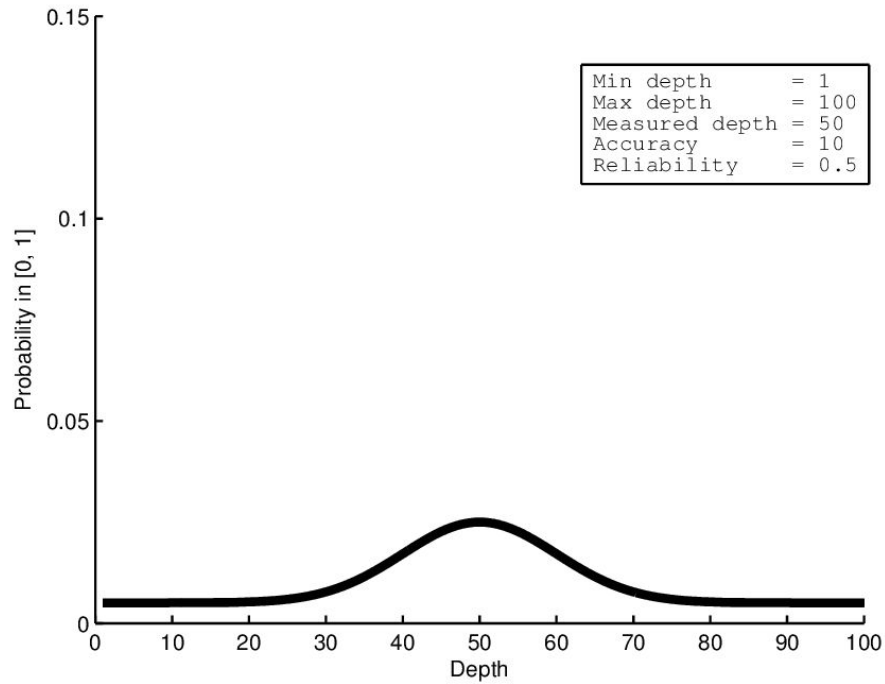


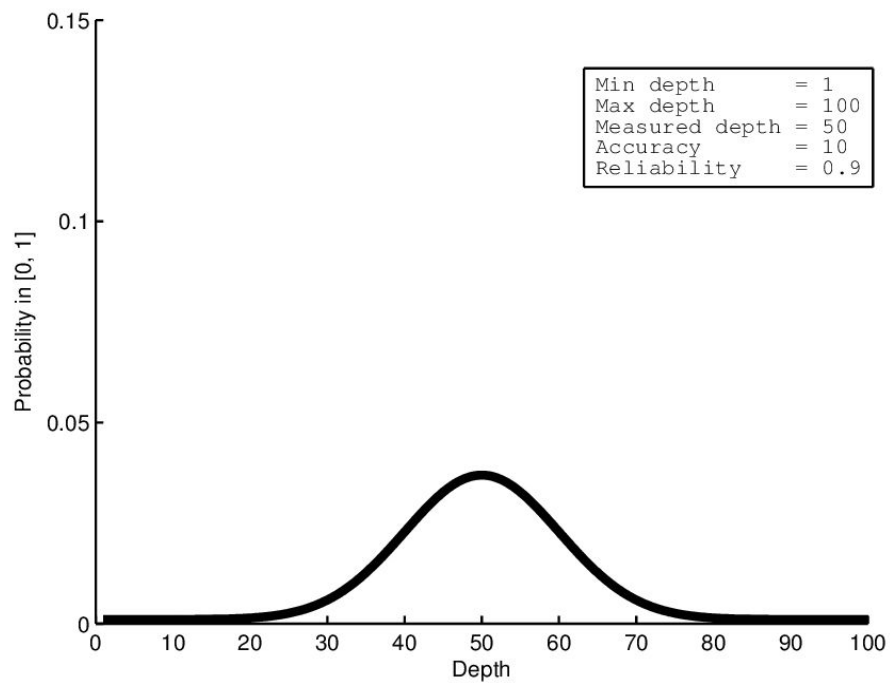
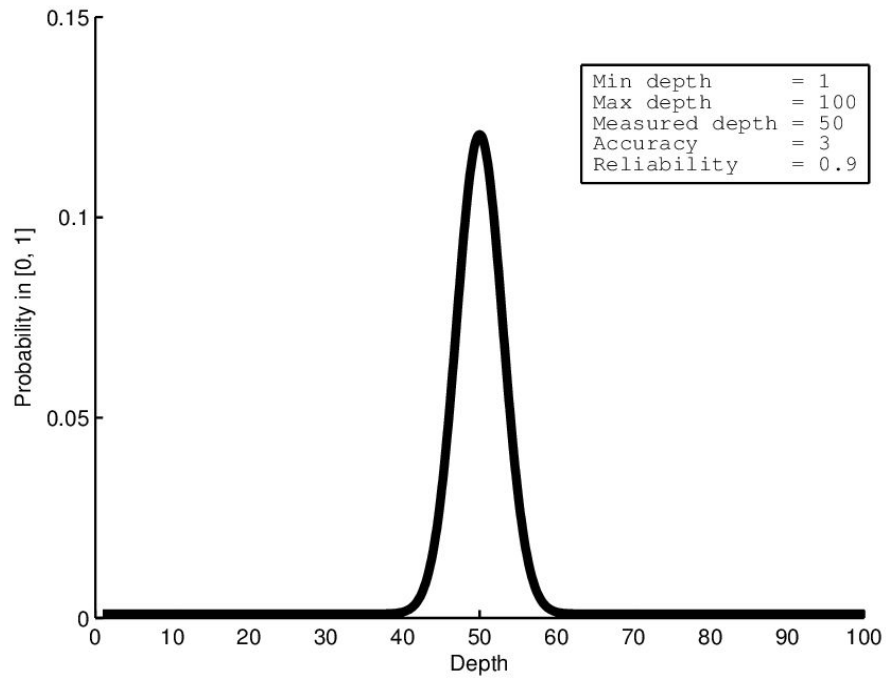


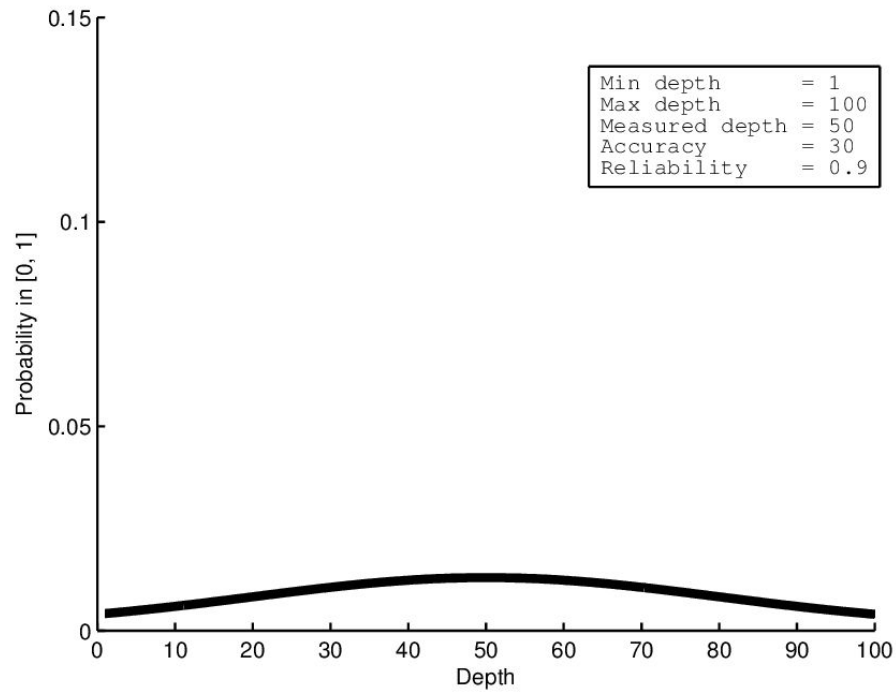
Accuracy: 30
Reliability: 0.5



Accuracy: 3
Reliability: 0.1







Accuracy: 30

Reliability: 0.9

Except as otherwise noted, the content of this document is licensed under the [Creative Commons Attribution 4.0 License](#), and code samples are licensed under the [Apache 2.0 License](#).