LGSx SDK

2026.0.0 (2026.0.0+5b1e762)

Generated by Doxygen 1.13.2

Send comments on this topic to LGSx SDK Support

tls.sdksupport@leica-geosystems.com

Chapter 1

Documentation

- Release notes
- Leica LGSx SDK Getting Started Guide
- SDK Examples

1.1 Release notes

- Release Notes 2026.0.0 (July 7, 2026)
- Release Notes 2025.0.1 (November 12, 2025)
- Release Notes 2025.0.0 (October 1, 2025)
- Release Notes 2024.0.1 (April 22, 2024)
- Release Notes 2024.0.0 (February 2, 2024)

1.1.1 Release Notes 2026.0.0

Product	Leica LGSx SDK 2026.0.0
Date	July 7, 2026
Developed by	Reality Capture Software Product Management

1.1.1.1 Important notes

- The Sensor Info API is stable. However, sensor information is only available when present in the source LGSx file. For existing files, sensor-related fields may be unavailable or return default values.

1.1.1.2 Changelog

1.1.1.2.1 Added

- Metadata type `MetadataType_DOUBLEARRAY` for arbitrary length double arrays.
- Non-truncating API alternatives for all public functions that previously used fixed-size `wchar_t` output buffers:
  - **Buffer overflow fixes** (critical): `wcscpy` replaced with safe copy in model node and model name handling.
  - **Pattern C getters** (call after MoveNext, before End): `Lgsx_ReaderGetCurrentSetupName()`, `Lgsx_ReaderGetCurrentTargetLabel()`, `Lgsx_ReaderGetCurrentSetupCameraName()`.
  - **"2" variants** (atomic alternatives): `Lgsx_ReaderMoveNextRun2()`, `Lgsx_ReaderMoveNextCoordSystems2()`, `Lgsx_ReaderGetModelNodeInfo2()`, `Lgsx_ReaderGetModelInfo2()`.
  - **AssetHandle API**: new handle type `AssetHandle` with `Lgsx_ReaderGetGeoTagAssetHandle()`, `Lgsx_ReaderGetAssetHandle()`, `Lgsx_AssetRelease()`, and getters `Lgsx_AssetGetName()`, `Lgsx_AssetGetType()`, `Lgsx_AssetGetMimeType()`, `Lgsx_AssetGetFilename()`, `Lgsx_AssetGetUrl()`, `Lgsx_AssetGetGuid()`, `Lgsx_AssetGetId()`, `Lgsx_AssetGetCreationTime()`, `Lgsx_AssetGetModificationTime()`, `Lgsx_AssetIsExternalUrl()`, `Lgsx_AssetGetMetadata()`, `Lgsx_AssetHandleReadImage()`, `Lgsx_AssetHandleReadBinary()`.
  - **Function-level "2" variants**: `Lgsx_WriterGetLastError2()`, `Lgsx_GetProductVersion2()`, `Lgsx_ReaderGetLUTImage2()`, `Lgsx_ReaderGetLUTImageOfSetup2()`, `Lgsx_ReaderGetSetupCameraImage2()`.
  - All returned `wchar_t**` strings must be freed with `LgsxUtil_FreeMem()`. All returned `const wchar_t**` strings are read-only and bound to the handle lifetime.
- `Lgsx_MetaGetDoubleArraySize()` function to query the size of a double array in metadata.
- `Lgsx_MetaGetDoubleArray()` function to retrieve double array values from metadata.
- Cubemap face index enumeration `CubemapFaceIndex`
- Cubemap functions `Lgsx_ReaderGetCubemapImageBytes()`, `Lgsx_ReaderGetCubemapFaceMetadata()`, `Lgsx_ReaderGetCubemapMetadata()`, `Lgsx_ReaderGetCubemapFaceSize()`, `Lgsx_ReaderGetCubemapMaxLevel()`, `Lgsx_ReaderGetCubemapImageType()`, `Lgsx_ReaderGetCubemapImageBytes`.
- `Lgsx_GetCubemapFaceOrientation()` to retrieve the default orientation quaternion for a specific cubemap face.
- `LgsxUcsExample` sample application demonstrating User Coordinate System (UCS) functionality.
- `LgsxExtractCubemapsExample` sample application demonstrating how to extract cubemap images as bytes, without a need to decode.
- `LgsxSetupIndexExample` sample application demonstrating how to read setup index (`PM_SETUPIDX`) from point data and count points per setup, with subsampling support.
- `Lgsx_ReaderEndTargets()` added to finish target enumeration.
- `Lgsx_ReaderEndGeoTags()` added to finish geotag enumeration.
- `Lgsx_ReaderEndSitemaps()` added to finish sitemap enumeration.
- `Lgsx_ReaderEndAssets()` added to finish asset enumeration.

- Lgsx_ReaderEndSetupCameras() added to finish setup camera enumeration.
- Lgsx_ReaderEndCoordSystems() added to finish coordinate system enumeration.
- GeotagAnchorType type added.
- Lgsx_GeoTagGetId() added for the handle-based GeoTag API.
- Handle-based sitemap API with new types SitemapHandle, SitemapItemHandle, SitemapItemType, SitemapImageType, and CYSITEMAPIMAGECORNERS2D:
  - Enumeration: Lgsx_ReaderEnumSitemaps(), Lgsx_ReaderGetSitemapHandle(), Lgsx_ReaderEndSitemaps(), Lgsx_SitemapRelease().
  - Per-sitemap accessors: Lgsx_SitemapGetId(), Lgsx_SitemapGetGuid(), Lgsx_SitemapGetName(), Lgsx_SitemapGetOrderingIndex(), Lgsx_SitemapGetIsMaster(), Lgsx_SitemapGetMetadata(), Lgsx_SitemapGetActiveCoordSystemId().
  - Per-item accessors: Lgsx_SitemapEnumItems(), Lgsx_SitemapGetItem(), Lgsx_SitemapItemGetId(), Lgsx_SitemapItemGetType(), Lgsx_SitemapItemGetGuid(), Lgsx_SitemapItemGetPose().
  - Per-image accessors: Lgsx_SitemapHasImage(), Lgsx_SitemapImageGetHandle(), Lgsx_SitemapImageGetMetadata(), Lgsx_SitemapImageRead(), Lgsx_SitemapImageReadBinary(), Lgsx_SitemapImageGetIsBlank(), Lgsx_SitemapImageGetCorners().
- UCS accessors for current coordinate system during enumeration: Lgsx_ReaderGetCurrentCoordSystemId(), Lgsx_ReaderGetCurrentCoordSystemGuid().
- Lgsx_AssetHandleReadImageBinary() reads raw encoded image bytes (jpg/png/jxr) from any AssetHandle, stripping SDK metadata. The caller can verify the asset is an image beforehand using Lgsx_AssetGetMimeType().
- LgsxExtractSitemapsExample sample application demonstrating sitemap enumeration, sitemap item inspection, and decoded sitemap image extraction.
- Field pose added to CYTRAJECTORYINFO.
- Lgsx_ReaderRunGetGuid() function to get run GUID.
- Point cloud file-like access API: Lgsx_ReaderPointCloudFileOpen(), Lgsx_PointCloudFileClose(), Lgsx_PointCloudFileGetSize(), Lgsx_PointCloudFileRead(), PointCloudFileHandle.
- EnumPointsConfigHandle and associated functions (Lgsx_InitEnumPointsConfig(), Lgsx_ReleaseEnumPointsConfig(), setters) provide a handle-based configuration for point enumeration—including an optional memory-hint to limit reader memory usage.
- Lgsx_ReaderEnumPointsEx2() now accepts an EnumPointsConfigHandle instead of individual parameters, enabling future extension without signature changes.
- Lgsx_MetaIsEmpty() function to check if a metadata object contains any entries.
- SensorInfo handle type SensorInfoHandle and functions Lgsx_ReaderHasSetupSensorInfo(), Lgsx_ReaderGetSetupSensorInfo(), Lgsx_SensorInfoGetGuid(), Lgsx_SensorInfoGetDeviceInfoCaptureTime(), Lgsx_SensorInfoGetManufacturer(), Lgsx_SensorInfoGetScannerType(), Lgsx_SensorInfoGetSerialNumber(), Lgsx_SensorInfoGetArticleNumber(), Lgsx_SensorInfoGetHardwareVersion(), Lgsx_SensorInfoGetFirmwareVersion(), Lgsx_SensorInfoGetDeviceInfoSnapshot(), Lgsx_SensorInfoRelease() to handle sensor info associated with setup.

- ProcessingInfo handle type `ProcessingInfoHandle` and functions `Lgsx_ReaderEnumSetupProcessingInfos()`, `Lgsx_ReaderGetSetupProcessingInfo()`, `Lgsx_ProcessingInfoRelease()`, `Lgsx_ProcessingInfoGetGuid()`, `Lgsx_ProcessingInfoGetProcessingTimestamp()`, `Lgsx_ProcessingInfoGetAppName()`, `Lgsx_ProcessingInfoGetAppVersion()`, `Lgsx_ProcessingInfoHasLibraryVersion()`, `Lgsx_ProcessingInfoGetLibraryVersion()`, `Lgsx_ProcessingInfoGetOrderIndex()`, `Lgsx_ProcessingInfoGetPipelineInfo()` to handle processing info associated with setup.
- Run-level SensorInfo functions `Lgsx_ReaderHasRunSensorInfo()`, `Lgsx_ReaderGetRunSensorInfo()` to access sensor info through the current run's phantom setup.
- Run-level ProcessingInfo functions `Lgsx_ReaderEnumRunProcessingInfos()`, `Lgsx_ReaderGetRunProcessingInfo()` to access processing info through the current run's phantom setup.
- Scan handle type `ScanHandle` and functions `Lgsx_ReaderEnumSetupScans()`, `Lgsx_ReaderGetSetupScan()`, `Lgsx_ScanRelease()`, `Lgsx_ScanGetGuid()`, `Lgsx_ScanGetScanType()`, `Lgsx_ScanGetScanRole()`, `Lgsx_ScanGetPointCount()`, `Lgsx_ScanGetCreationTimestamp()`, `Lgsx_ScanGetModificationTimestamp()`, `Lgsx_ScanHasWindow()`, `Lgsx_ScanGetWindowMinAzimuth()`, `Lgsx_ScanGetWindowMinElevation()`, `Lgsx_ScanGetWindowMaxAzimuth()`, `Lgsx_ScanGetWindowMaxElevation()`, `Lgsx_ScanGetCaptureTimestamp()`, `Lgsx_ScanGetScanDensity()`, `Lgsx_ScanGetCaptureInfoSnapshot()` to access scan metadata (type, role, point count, window, capture info) per setup.
- Run-level Scan functions `Lgsx_ReaderEnumRunScans()`, `Lgsx_ReaderGetRunScan()` to access scans through the current run's phantom setup.
- `LgsxSensorProcessingInfoExample` sample application demonstrating how to read SensorInfo and ProcessingInfo data associated with setups and runs.
- String conversion utility functions for safer UTF conversion: bounded-buffer variants `LgsxUtil_A2WEx()` and `LgsxUtil_W2AEx()` add explicit output-size parameters for UTF-8/UTF-16 conversion, and allocating variants `LgsxUtil_AllocA2W()` and `LgsxUtil_AllocW2A()` allocate the required output buffer and return it to the caller. Buffers returned by the allocating variants must be freed using `LgsxUtil_FreeMem()`.

1.1.1.2.2 Changed

- `Lgsx_SitemapImageGetCorners()` extended with automatic fixes of corner positions, controlled by flags (`pFlags` parameter).
- SDK sample applications were refactored from one monolithic reader example into focused reader examples (`LgsxProjectInfoExample`, `LgsxSetupsExample`, `LgsxRunsExample`, `LgsxPointCloudExample`, `LgsxGeoTagsExample`, `LgsxAssetsExample`, `LgsxTargetsExample`, and `LgsxSitemapsExample`). `LgsxUcsExample` and `LgsxSetupIndexExample` remain cohesive advanced examples. Package verification scripts, test package execution, and getting-started sample indexing were updated accordingly.
- `Lgsx_ReaderGetProjectInfo()` no longer fails if project thumbnail is missing.
- Sitemap API replaced: the experimental cursor-based functions and structs have been removed (see Removed) and replaced by the handle-based sitemap API (see Added).
- `Lgsx_GeoTagGetSetupIndex()` renamed to `Lgsx_GeoTagGetAnchor()` and extended, as the name was misleading and not providing all required info.

- `WinDllExport.hpp` replaced by `LgsxExport.h` (export macros) and `LgsxPlatform.h` (platform types). Update includes accordingly.
- `LinearAlgebra.h` renamed to `LinearAlgebra.hpp`.
- All public C headers (`.h`) are now valid C11. Struct and enum declarations use `typedef` form. `PROPERTY_NAME_LENGTH` is now a `#define`.
- `LgsxPlatform.h` is now a pure C header; C++ standard library includes are no longer provided transitively.
- `M3DPinholeDistortion` enum extracted from `M3DPINHOLE` struct and renamed to `M3DPINHOLEDISTORTION`. Enum values renamed with `M3DPINHOLEDISTORTION_` prefix: e.g. `distortionK1` -> `M3DPINHOLEDISTORTION_K1`, `distortionElements` -> `M3DPINHOLEDISTORTION_COUNT`.

1.1.1.2.3 Removed

- Deprecated cursor-based sitemap functions: `Lgsx_ReaderMoveNextSitemap()`, `Lgsx_ReaderMoveNextSitemapImage()`, `Lgsx_ReaderMoveNextSiteMapImage()`, `Lgsx_ReaderGetSitemap()`, `Lgsx_ReaderGetSitemapImage()`, `Lgsx_ReaderEnumSitemapItems()`, `Lgsx_ReaderEndSitemapItems()`, `Lgsx_ReaderMoveNextSitemapItem()`. Use the handle-based sitemap API instead.
- Deprecated structs `CYSITEMAP` and `CYSITEMAPIMAGE`. The handle-based API returns individual fields via dedicated getters.

1.1.1.2.4 Fixed

- `Lgsx_ReaderGetAssetHandle()` now keeps at most one temp file on disk at a time and returns a non-zero error code on extraction failure. More information in documentation for `Lgsx_ReaderGetAssetHandle()`.
- `Lgsx_ReaderEnumPointsEx()` now correctly honors the `subSample` parameter for point cloud subsampling.
- `Lgsx_ReaderEnumTarget()` follows the documentation and returns targets for the whole project and not just for current setup.
- `Lgsx_ReaderEnumTargetOfSetup()` does not require the setup to be in currently enumerated setup list.
- `Lgsx_ReaderEnumGeoTags()` no longer requires prior enumeration of setups and is no longer affected by stage of the setup enumeration.
- `Lgsx_ReaderEnumGeoTagsOfSetup()` no longer requires prior enumeration of setups and is no longer affected by context of the setup enumeration.
- `Lgsx_ReaderEnumGeoTagsOfSetup()` documentation was clarified that it returns geotags visible or related to setup, not attached to setup.
- `Lgsx_ReaderEnumSetupCamera()` documentation was clarified, that it returns cameras for current setup in enumeration and not in the project.
- `Lgsx_ReaderGetLUTImage()` and `Lgsx_ReaderGetLUTImageOfSetup()` documentation corrected: `typeName` is an output parameter (the SDK writes the type name into the caller-provided buffer), not an input. The stale "no workaround" warnings were updated to reference `Lgsx_ReaderGetLUTImage2()` and `Lgsx_ReaderGetLUTImageOfSetup2()` respectively.

- Multiple documentation fixes across public headers: corrected parameter directions, removed phantom entries referencing non-existent parameters, and fixed recurring typos.
- GUID contract clarification in API documentation: GUID-returning getters may return an empty string when a source object has no persisted GUID (including `Lgsx_AssetGetGuid()`, `Lgsx_GeoTagGetGuid()`, `Lgsx_ReaderGetSetupGuid()`, and `Lgsx_ReaderRunGetGuid()`).

1.1.1.3 Additional Notes

Existing owners of LGS files will need to convert them to the LGSx format before opening or importing the LGSx content. The Leica LGS Converter Tool is available online (free of charge without any license).

1.1.1.4 Packaging

1. Windows Package (ZIP) - Made for Windows 10 and 11 (64-bit) using Visual Studio 2022 (19.43.34809.0).
2. Linux Package (TGZ) - Made for Ubuntu 22.04 LTS version (available on the Microsoft App Store) using gcc11.

Both packages include all applicable libraries, header and source files, sample programs, LGSx dataset, documentation (release notes, API documentation, Getting Started Guide).

Only the new LGSx format is supported. LGSx allows access to HSPC point clouds and other datasets.

The latest documentation is available on the `Reality Capture SDK site`.

1.1.1.5 Licensing

The LGSx SDK is available only to partners who have signed up for the Geosystems Partners Network (GPN).

A developer license key will be provided to enable the API, allowing Licensee developers to access the available functionality. The developer key must not be distributed with the integrated product.

A deployment license key will be provided to enable the API in the distributed integrated product.

The Licensee must have the LGSx SDK installed to start coding against the API.

1.1.2 Release Notes 2025.0.1

Product	Leica LGSx SDK 2025.0.1 (patch release)
Date	November 12, 2025
Developed by	Reality Capture Software Product Management

1.1.2.1 Changelog

1.1.2.1.1 Added

- Function `Lgsx_GetMigratedFieldGuid` to return GUIDs of migrated fields `Category`, `Label`, `TagIndex`

1.1.2.1.2 Fixed

- Migration (automatically updates the database schema when opening files created with older software versions): Attachments now preserve their UUIDs and creation times, LinkRef in GeoTags correctly handles both external URLs and local references, and GeoTags modification times remain unchanged
- LgsxReaderExample (example application): Fixes related to non-ASCII characters handling

1.1.2.1.3 Other

- The Linux flavor of library no longer exports dependency symbols. This ensures a cleaner symbol table and prevents unintentional symbol exposure to avoid dependenies conflicts.

1.1.2.2 Additional Notes

Existing owners of LGS files will need to convert them to the LGSx format before opening or importing the LGSx content. The Leica LGS Converter Tool is available online (free of charge without any license).

1.1.2.3 Packaging

1. Windows Package (ZIP) - Made for Windows 10 and 11 (64-bit) using Visual Studio 2022 (19.43.34809.0).
2. Linux Package (TGZ) - Made for Ubuntu 22.04 LTS version (available on the Microsoft App Store) using gcc11.

Both packages include all applicable libraries, header and source files, sample programs, LGSx dataset, documentation (release notes, API documentation, Getting Started Guide).

Only the new LGSx format is supported. LGSx allows access to HSPC point clouds and other datasets.

The latest documentation is available on the `Reality Capture SDK site`.

1.1.2.4 Licensing

The LGSx SDK is available only to partners who have signed up for the Geosystems Partners Network (GPN).

A developer license key will be provided to enable the API, allowing Licensee developers to access the available functionality. The developer key must not be distributed with the integrated product.

A deployment license key will be provided to enable the API in the distributed integrated product.

The Licensee must have the LGSx SDK installed to start coding against the API.

1.1.3 Release Notes 2025.0.0

Product	Leica LGSx SDK 2025.0.0 (added support for the updated GeoTag data scheme)
Date	October 1, 2025
Developed by	Reality Capture Software Product Management

1.1.3.1 Changelog

1.1.3.1.1 Added

- Additional documentation for multiple API elements, including data types and functions.
- `LgsxUtil_StrDupA()`/`LgsxUtil_StrDupW()` functions for duplicating strings.
- **Experimental** support for sitemap images (status: work in progress), including: `SitemapImageType`, `CYSITEMAP`, `CYSITEMAPIMAGE`, `Lgsx_ReaderMoveNextSitemap()`, `Lgsx_ReaderGetSitemap()`, `Lgsx_ReaderGetSitemapImage()`, `Lgsx_ReaderEnumSetupsForSitemap()`, `Lgsx_ReaderMoveNextSetupForSitemap()`.
- Versions of multiple functions that duplicate strings to avoid truncating the output: `Lgsx_ReaderGetLastError2()`, `Lgsx_ReaderGetProjectDescription2()`, `Lgsx_ReaderGetImageLayerNames2()`, `Lgsx_GetAssetAsImage2()`, `Lgsx_ReaderMoveNextCoordSystems2()`.
- Function to check if a file is password-protected: `Lgsx_ReaderHasPassword()`.
- Support for retrieving point cloud points in strict order (if possible) via `Lgsx_ReaderEnumPointsEx()`.
- Project Assets extended with mime-type, url and id attributes.
- New API for reading the Assets: `Lgsx_AssetReadImage()`, `Lgsx_AssetReadBinary()`, `Lgsx_ReaderAssetGetGuid()`, `Lgsx_AssetHasType()`.
- New API for reading the Setups: `Lgsx_ReaderGetSetupGuid()`.
- New API for GeoTags: `Lgsx_ReaderGetGeoTag()`, `Lgsx_GeoTagRelease()`, `Lgsx_GeoTagGetGuid()`, `Lgsx_GeoTagGetName()`, `Lgsx_GeoTagHasDescription()`, `Lgsx_GeoTagGetDescription()`, `Lgsx_GeoTagGetPosition()`, `Lgsx_GeoTagGetCameraPosition()`, `Lgsx_GeoTagGetSetupIndex()`, `Lgsx_GeoTagGetMetadata()`, `Lgsx_GeoTagGetThumbnail()`, `Lgsx_GeoTagEnumAssets()`, `Lgsx_ReaderGetGeoTagAsset()`, `Lgsx_ReaderGetGeoTagAssetThumbnail()`, `Lgsx_GeoTagHasThumbnail()`, `Lgsx_GeoTagEnumCategoryEntries()`, `Lgsx_GeoTagGetCategoryEntry()`, `Lgsx_GeoTagEnumFieldEntries()`, `Lgsx_GeoTagGetFieldEntry()`, `Lgsx_GeoTagGetRelativePosition()`, `Lgsx_GeoTagGetCameraRelativePosition()`.
- Deprecated API for Geotags: `Lgsx_ReaderGetGeoTagName()`, `Lgsx_ReaderHasGeoTagDescription()`, `Lgsx_ReaderGetGeoTagDescription()`.
- New API for Categories: `Lgsx_ReaderEnumCategories()`, `Lgsx_ReaderGetCategory()`, `Lgsx_CategoryRelease()`, `Lgsx_CategoryGetGuid()`, `Lgsx_CategoryGetName()`, `Lgsx_CategoryGetPriority()`, `Lgsx_CategoryEnumValues()`, `Lgsx_CategoryGetValue()`, `Lgsx_CategoryValueGetValue()`, `Lgsx_CategoryValueGetPriority()`, `Lgsx_CategoryValueHasColor()`, `Lgsx_CategoryValueGetColor()`, `Lgsx_CategoryEntryGetCategory()`, `Lgsx_CategoryEntryHasValue()`, `Lgsx_CategoryEntryGetValue()`, `Lgsx_CategoryValueGetGuid()`.

- New API for Fields: Lgsx_ReaderEnumFields(), Lgsx_ReaderGetField(), Lgsx_FieldRelease(), Lgsx_FieldGetGuid(), Lgsx_FieldGetName(), Lgsx_FieldGetPriority(), Lgsx_FieldEntryGetField(), Lgsx_FieldEntryGetValue().
- New API for metadata: enumerate without name truncation Lgsx_MetaMoveNext2(), getting string value without truncation Lgsx_MetaGetString2().
- Fields: id, creation and modification timestamp to CYASSET.
- Fields: cameraPosition, mimeType, url, id, creation and modification timestamp to CYGEOTAG.
- New API for getting creation/modification timestamps: Lgsx_GeoTagGetCreationTimestamp(), Lgsx_GeoTagGetModificationTimestamp(), Lgsx_FieldGetCreationTimestamp(), Lgsx_FieldGetModificationTimestamp(), Lgsx_CategoryGetCreationTimestamp(), Lgsx_CategoryGetModificationTimestamp(), Lgsx_CategoryValueGetCreationTimestamp(), Lgsx_CategoryValueGetModificationTimestamp().

1.1.3.1.2 Changed

- Functions for metadata (Lgsx_Meta*) accepts also longer names. No API changes required due to C array decay.

1.1.3.1.3 Deprecated

- Lgsx_ReaderEnumSiteMaps() renamed to Lgsx_ReaderEnumSitemaps().
- Lgsx_ReaderMoveNextSiteMapImage() renamed to Lgsx_ReaderMoveNextSitemapImage().
- CYGEOTAG replaced with the new API for GeoTags.
- Lgsx_ReaderMoveNextGeoTag() replaced with Lgsx_ReaderGetGeoTag().
- Lgsx_MetaMoveNext() replaced with Lgsx_MetaMoveNext2().

1.1.3.1.4 Removed

- Multiple unused type definitions: AnyHandle, GeomPtr, FilterPtr, QueryPtr, ClusterPtr, ObjectListPtr.

1.1.3.1.5 Fixed

- Implemented missing functions in Linux packages: Lgsx_GetProductVersion(), Lgsx_InitProductVersion(), Lgsx_InitProductVersionEx(), Lgsx_InitProductName(), Lgsx_InitProductBuildNumber().
- Issues with handling files without point clouds.

1.1.3.1.6 Other

- Headers now include Doxygen-formatted documentation.
- Various cleanup and fixes related to UTF character handling.

1.1.3.2 Additional Notes

Existing owners of LGS files will need to convert them to the LGSx format before opening or importing the LGSx content. The Leica LGS Converter Tool is available online (free of charge without any license).

1.1.3.3 Packaging

1. Windows Package (ZIP) - Made for Windows 10 and 11 (64-bit) using Visual Studio 2022 (19.43.34809.0).
2. Linux Package (TGZ) - Made for Ubuntu 22.04 LTS version (available on the Microsoft App Store) using gcc11.

Both packages include all applicable libraries, header and source files, sample programs, LGSx dataset, documentation (release notes, API documentation, Getting Started Guide).

Only the new LGSx format is supported. LGSx allows access to HSPC point clouds and other datasets.

The latest documentation is available on the Reality Capture SDK site.

1.1.3.4 Licensing

The LGSx SDK is available only to partners who have signed up for the Geosystems Partners Network (GPN).

A developer license key will be provided to enable the API, allowing Licensee developers to access the available functionality. The developer key must not be distributed with the integrated product.

A deployment license key will be provided to enable the API in the distributed integrated product.

The Licensee must have the LGSx SDK installed to start coding against the API.

1.1.4 Release Notes 2024.0.1

Product	Leica LGSx SDK 2024.0.1 (Minor patch release)
Date	April 22, 2024
Developed by	Reality Capture Software Product Management

1.1.4.1 What's New

- Evaluation Read-Only license key is valid until July 1, 2024.
- The 'Samples' folder was updated with another example program.
- The release is targeting integrators who are interested in open/import LGSx files only. This release is provided in two distributions:

1.1.4.2 Packaging

1. Windows InstallShield
2. Linux Package (tgz) - Made for Ubuntu 20.04.6 LTS version (Available on Microsoft App Store).

Both include all the applicable libs, header and source files, sample programs, LGSx dataset, release notes, API documentation, and Getting Started Guide.

Only the new LGSx format is supported. LGSx allows access to HSPC point clouds and other datasets.

Latest documentations are available on `Reality Capture SDK site`.

1.1.4.3 Fixed Issues

Several small fixes were made to allow more LGSx file types to be opened/imported.

1.1.4.4 Known Issues

Existing owners of LGS files will need to convert to LGSx format prior to opening/importing the LGSx content. Leica LGS Converter Tool is available online (Free of charge w/o any license).

1.1.4.5 Licensing

The LGSx SDK is available only to partners who signed up for the Geosystems Partners Network (GPN).

A developer license key will be provided to enable the API so that Licensee developers can access the available functionality in the API. The developer key shall not be distributed with the Integrated Product.

A deployment license key will be provided to enable the API in the distributed Integrated Product.

The Licensee will need to have the LGSx SDK installed to start coding against the API.

1.1.5 Release Notes 2024.0.0

Product	Leica LGSx SDK 2024.0 (First Release to 3rd Party Partners)
Date	February 2, 2024
Developed by	Reality Capture Software Product Management

1.1.5.1 What's New

The first release is targeting integrators who are interested in open/import LGSx files only.

1.1.5.2 Packaging

This release is provided in two distributions:

1. Windows InstallShield
2. Linux Package (tgz) - Made for Ubuntu 20.04.6 LTS version (Available on Microsoft App Store).

Both include all the applicable libs, header and source files, sample programs, LGSx dataset, release notes, API documentation, and Getting Started Guide.

Only the new LGSx format is supported. LGSx allows access to HSPC point clouds and other datasets.

Latest documentations are available on `Reality Capture SDK site`.

1.1.5.3 Fixed Issues

None

1.1.5.4 Known Issues

Existing owners of LGS files will need to convert to LGSx format prior to opening/importing the LGSx content. Leica LGS Converter Tool is available online (Free of charge w/o any license).

1.1.5.5 Licensing

The LGSx SDK is available only to partners who signed up for the Geosystems Partners Network (GPN).

A developer license key will be provided to enable the API so that Licensee developers can access the available functionality in the API. The developer key shall not be distributed with the Integrated Product.

A deployment license key will be provided to enable the API in the distributed Integrated Product.

The Licensee will need to have the LGSx SDK installed to start coding against the API.

1.2 Leica LGSx SDK Getting Started Guide

1.2.1 Overview

This guide is intended for Windows and Linux developers. Each LGSx SDK release includes one or more sample programs that demonstrate the available API functions.

1.2.2 Windows Integration

The following steps outline the process for utilizing the LGSx SDK:

1. Extract the provided .zip file containing the LGSx SDK. The extracted file structure should appear as follows:

   ```
   \acknowledgements
   \bin
   \doc
   \include
   \lib
   \samples
   VC_redist.x64.exe
   ```

   After extraction, install VC_redist.x64.

2. The full SDK documentation is located in the `\doc` folder.

3. A sample program executable, LgsxProjectInfoExample.exe, is located in the `\bin` folder. This sample can be used to output key metadata from an LGSx file. A sample LGSx file, testMobile.lgsx, is included in the `\samples` folder. To test the executable, use the following command:

   ```
   LgsxProjectInfoExample -i ../samples/testMobile.lgsx
   ```

4. The following steps describe how to build and run the provided sample programs from source code. Visual Studio 2022 and CMake 3.16 (or newer) are required.

   From the command line, starting in the extracted LGSx SDK root folder:

   ```
   cd samples
   cmake -S . -B ./msvc2022_64
   ```

   This command generates a Visual Studio solution file in a new targets folder located at `samples\msvc2022_64`.

   ```
   cd msvc2022_64
   LgsxSdk-Examples.sln
   ```

   This command navigates to the new targets folder and launches Visual Studio with the newly created solution. The sample program can be built and run from within Visual Studio.

1.2.3 Linux Integration

1. Extract the provided LGSx SDK package:

   ```
   sudo tar -xvzf <LGSx SDK Package>.tar.gz
   ```

   The extracted directory structure should appear as follows:

   ```
   /acknowledgements
   /bin
   /doc
   /include
   /lib
   /samples
   ```

2. The full SDK documentation is located in the doc folder.

3. The LGSx package includes a sample application, LgsxProjectInfoExample, which outputs key project metadata from an LGSx file. Both the source and executable for provided examples can be found in the SDK. The following steps outline how to run the executable, starting in the LGSx SDK root directory:

   ```
   cd bin
   ./LgsxProjectInfoExample -i ../samples/testMobile.lgsx
   ```

4. To build provided examples from source code, gcc 11 and CMake 3.16` (or newer) are required.

   A. Setting up the environment

   The required development tools can be installed using the following commands:

   ```
   sudo apt-get update
   sudo apt-get install --no-install-recommends -y \
       build-essential \
       gcc-multilib g++-multilib \
       cmake ninja-build
   ```


Install the necessary library dependencies using:

```
sudo apt-get install --no-install-recommends -y \
    libgomp1 libcurl4 libgl1 libglu1
```

B. Building and running examples

Use the following steps to build and run provided examples (starting in the LGSx SDK root directory):

```
cd samples cmake -S . -B ./build cd build cmake --build .
```

This process creates a targets directory samples/build and builds provided examples executables there.

To test LgsxProjectInfoExample, use the same command as previously:

```
./LgsxProjectInfoExample -i ../testMobile.lgsx
```

1.2.4 WSL Integration

The following steps outline the process for utilizing the LGSx SDK under Windows Subsystem for Linux (WSL):

1. Enable "Virtualization in BIOS."
2. Enable Hyper-V in Windows.
3. Install WSL. Install the supported Ubuntu release from Microsoft Store (Ubuntu 22.04.5 LTS).

1.2.4.1 LgsxSensorProcessingInfoExample

An example focused on reading and displaying SensorInfo and ProcessingInfo data associated with setups and runs. This example iterates through all TLS setups and mobile runs, printing detailed sensor and processing information for each context.

Sample file: testMobile.lgsx

Basic usage:

```
LgsxSensorProcessingInfoExample -i testMobile.lgsx
```

Key features demonstrated:

- Reading SensorInfo for TLS setups (`Lgsx_ReaderHasSetupSensorInfo`, `Lgsx_ReaderGetSetupSensorInfo`)
- Reading ProcessingInfo for TLS setups (`Lgsx_ReaderEnumSetupProcessingInfos`, `Lgsx_ReaderGetSetupProcessingInfo`)
- Reading SensorInfo for runs (`Lgsx_ReaderHasRunSensorInfo`, `Lgsx_ReaderGetRunSensorInfo`)
- Reading ProcessingInfo for runs (`Lgsx_ReaderEnumRunProcessingInfos`, `Lgsx_ReaderGetRunProcessingInfo`)
- Displaying all SensorInfo fields: scanner type, serial number, firmware/hardware version, manufacturer, article number, capture timestamp, and device info snapshot metadata
- Displaying all ProcessingInfo fields: application name/version, processing timestamp, library version, order index, and pipeline info metadata

1.2.5 Large Point Cloud Considerations

When working with large point cloud files, the SDK may open a significant number of file descriptors. This is particularly important on Linux and Unix-like systems where file descriptor limits can impact application performance and stability.

1.2.5.1 File Descriptor Limits

Large point cloud files can require many open files simultaneously. For example, processing a point cloud with 7.66 billion points distributed across 342 setups **peaked at 6150 simultaneously open file descriptors**, far exceeding typical system defaults (usually 1024).

Checking Current Limits (Linux/Unix):

```
# Check soft limit
ulimit -n

# Check hard limit
ulimit -Hn

# Check process limits
cat /proc/sys/fs/file-max
cat /proc/[PID]/limits

# Check actual usage (replace [PID] with process ID)
ls /proc/[PID]/fd | wc -l
```

Increasing File Descriptor Limits (Linux/Unix):

To increase limits temporarily (for current session):

```
ulimit -n 8192   # Set soft limit to 8192
```

To increase limits permanently, edit `/etc/security/limits.conf`:

```
* soft nofile 8192
username soft nofile 8192
```

Or edit `/etc/systemd/system.conf` for system-wide limits:

```
DefaultLimitNOFILE=8192
```

For systemd services, edit the service file:

```
[Service]
LimitNOFILE=8192
```

On Windows: Windows typically has much higher file descriptor limits and rarely encounters this issue. However, ensure your system has adequate resources for large file access.

1.2.5.2 Point Cloud Size and File Descriptor Impact

File descriptor usage is **primarily correlated with point cloud size** (number of points), not the number of setups. The SDK partitions large point clouds into multiple files for efficient access and processing.

Size-to-descriptor relationship:

- **7.66 billion points**: Peak of **6150 file descriptors** (range: 46-6150)
- **Point cloud partitioning**: Large point datasets are split into multiple files (chunks/tiles), requiring an open file descriptor for each chunk being processed
- **Recommended minimum soft limit for large point cloud processing:** 8192

Before processing large point cloud files, especially those with billions of points, ensure your system's file descriptor limit is set appropriately to prevent "too many open files" errors. Use the checking commands above to monitor actual usage during processing.

Rule of thumb: If processing files with point counts in the billions, set the soft file descriptor limit to at least 8192.

1.2.6 SDK Know-How

1. Most SDK functionality is located in the include/LgsxSdk/LgsxReader.h header file. This file should be included in the application.
2. All sample programs follow a standard workflow pattern. Refer to the LgsxProjectInfoExample main function for the typical initialization and cleanup sequence. Additional details can be found in the SDK documentation.

1.2.7 Examples

The SDK package includes a curated set of sample programs shipped under `samples/` (source) and `bin/` (executables).

The examples are organized as:

- Fundamentals for first-time SDK usage.
- Focused reader examples for individual data domains.
- Advanced examples for end-to-end workflows.
- Extraction utilities for export scenarios.

The full examples catalog and task-oriented snippets are maintained in the dedicated Examples documentation section generated from shipped source code.

1.3 SDK Examples

1.3.1 Examples Overview

The SDK provides executable samples in `bin/` and matching source code in `samples/`.

Use the examples in this order:

- Start with Fundamentals to learn the baseline SDK lifecycle.
- Continue with Focused Reader Examples for specific data domains.
- Use Advanced Examples for coordinate transforms and point attribution.
- Use Extraction Utilities when exporting point clouds or imagery.

The catalog page contains per-example implementation notes:

- what each example does,
- which SDK APIs are used,
- and which command lines are typical.

The snippets page contains focused code sections with short explanations of what each section does and why it is relevant.

1.3.2 SDK Examples Catalog

Use this catalog to choose examples by task and data domain. Each entry summarizes behavior, workflow, key SDK APIs, and typical command usage.

1.3.2.1 Fundamentals

Essential onboarding flows: initialize the SDK, open a project, and inspect core metadata safely.

1.3.2.1.1 LgsxLicenseDiagnosticsExample

- Source file: `LgsxLicenseDiagnosticsExample.cpp`
- What it does: Exercises SDK version, license initialization, and license diagnostics APIs.
- Description: This example demonstrates license and product-version diagnostics without opening a project file. It calls legacy and extended licensing APIs, including named-license checks, load/unload operations, and global error reporting helpers. Use this when validating deployment environment and license behavior.
- Key SDK APIs: Lgsx_GetProductVersion, Lgsx_GetProductVersion2, Lgsx_InitProductVersion, Lgsx_InitProductVersionEx, Lgsx_InitProductName, Lgsx_InitProductBuildNumber, Lgsx_InitLicense, Lgsx_InitLicenseEx2, Lgsx_IsLicensed, Lgsx_LicenseDaysLeft, Lgsx_LoadLicense, Lgsx_UnloadLicense, Lgsx_IsLicensedExA, Lgsx_LicenseDaysLeftExA, Lgsx_LoadLicenseExA, Lgsx_UnloadLicenseExA, Lgsx_GetLastError, Lgsx_GetLastErrorNo
- Typical usage: `LgsxLicenseDiagnosticsExample -i testMobile.lgsx [-l license.txt]`

1.3.2.1.2 LgsxMetadataRoundtripExample

- Source file: `LgsxMetadataRoundtripExample.cpp`
- What it does: Demonstrates metadata roundtrip operations for scalar, vector, nested, and array APIs.
- Description: This example reads project metadata from the input LGSx file, clones metadata, checks/removes keys on a cloned copy, and performs typed reads including string and double-array access patterns when available. Use this as a focused reference for metadata container manipulation and validation.
- Key SDK APIs: Lgsx_ReaderGetProjectInfo, Lgsx_MetaClone, Lgsx_MetaReset, Lgsx_MetaMoveNext2, Lgsx_MetaHas, Lgsx_MetaRemove, Lgsx_MetaGetString, Lgsx_MetaGetDoubleArraySize, Lgsx_MetaGetDoubleArray
- Typical usage: `LgsxMetadataRoundtripExample -i testMobile.lgsx`

1.3.2.1.3 LgsxOpenFileExample

- Source file: `LgsxOpenFileExample.cpp`
- What it does: Minimal project open and close lifecycle with robust error handling.
- Description: This walkthrough demonstrates the minimal lifecycle every reader-based tool should implement. It validates command-line inputs, opens the project with password support, and exits early on open failures. After a successful open it prints a simple confirmation, then closes the reader explicitly to release file locks. Use this structure as the baseline skeleton before adding domain-specific processing logic.
- Key SDK APIs: Lgsx_ReaderOpen, Lgsx_ReaderClose
- Typical usage: `LgsxOpenFileExample -i testMobile.lgsx`

1.3.2.1.4 LgsxProjectInfoExample

- Source file: `LgsxProjectInfoExample.cpp`
- What it does: Reads project metadata and prints both highlighted and full metadata views.
- Description: This example is a metadata-first inspection flow that helps you validate a project before heavier processing. It opens the project, retrieves the root project metadata handle, then reads selected keys such as project name and version. After printing highlighted values, it emits a complete metadata dump so you can discover additional available fields. Use this pattern when building validators, preflight checks, or metadata-driven import pipelines.
- Key SDK APIs: Lgsx_ReaderGetProjectInfo, Lgsx_MetaGetString2, Lgsx_FreeHandle
- Typical usage: `LgsxProjectInfoExample -i testMobile.lgsx`

1.3.2.2 Focused Reader Examples

Task-focused reader workflows for specific project domains like setups, runs, point clouds, geotags, assets, targets, and sitemaps.

1.3.2.2.1 LgsxProjectDescriptionExample

- Source file: `LgsxProjectDescriptionExample.cpp`
- What it does: Demonstrates project metadata and migrated field GUID resolution.
- Description: Retrieves project description, error context, and maps deprecated field types to their current GUIDs.
- Key SDK APIs: Lgsx_ReaderGetLastError, Lgsx_ReaderGetProjectDescription, Lgsx_ReaderGetProjectDescription2, Lgsx_GetMigratedFieldGuid
- Typical usage: `LgsxProjectDescriptionExample -i testMobile.lgsx`

1.3.2.2.2 LgsxModelInfoExample

- Source file: `LgsxModelInfoExample.cpp`
- What it does: Demonstrates model node hierarchy and model asset metadata retrieval.
- Description: Enumerates model nodes, resolves parent-child relationships, and extracts model info with full names and asset references.
- Key SDK APIs: Lgsx_ReaderGetModelNodes, Lgsx_ReaderGetModelNodeInfo, Lgsx_ReaderGetModelNodeInfo2, Lgsx_ReaderGetModelInfo, Lgsx_ReaderGetModelInfo2
- Typical usage: `LgsxModelInfoExample -i testMobile.lgsx`

1.3.2.2.3 LgsxRunScansExample

- Source file: `LgsxRunScansExample.cpp`
- What it does: Demonstrates run enumeration and per-run scan traversal with timestamps.
- Description: Enumerates runs and their associated scans, extracting creation/modification metadata and scan type information.
- Key SDK APIs: Lgsx_ReaderEnumRuns, Lgsx_ReaderMoveNextRun, Lgsx_ReaderEnumRunScans, Lgsx_ReaderGetRunScan, Lgsx_ScanGetCreationTimestamp, Lgsx_ScanGetModificationTimestamp, Lgsx_ScanGetCaptureInfoSnapshot
- Typical usage: `LgsxRunScansExample -i testMobile.lgsx`

1.3.2.2.4 LgsxSetupsExample

- Source file: `LgsxSetupsExample.cpp`
- What it does: Enumerates setups with metadata, targets, related geotags, and image-layer content.
- Description: This tutorial focuses on setup-centric exploration for static and mixed datasets. It enumerates setups, prints setup metadata, then traverses related targets and geotags to show cross-linked entities. The example also inspects available image layers and reads representative panorama data for quick visual validation. Use this flow when building setup summaries, registration checks, or setup-level content browsers.
- Key SDK APIs: Lgsx_ReaderEnumSetups, Lgsx_ReaderMoveNextSetup, Lgsx_ReaderEnumTargetOfSetup, Lgsx_ReaderEnumGeoTagsOfSetup, Lgsx_ReaderGetPanoImage
- Typical usage: `LgsxSetupsExample -i testMobile.lgsx`

1.3.2.2.5 LgsxRunsExample

- Source file: `LgsxRunsExample.cpp`
- What it does: Enumerates runs, trajectory vertices, setup cameras, and LUT image context.
- Description: This example explains how to inspect mobile run structure from trajectory to setup camera attachments. It enumerates runs, prints trajectory samples and metadata, then drills into per-run setup and camera relationships. The flow also validates LUT/image availability so you can verify that camera-linked imagery is consistent. Use this as a foundation for trajectory QA, mobile capture diagnostics, and run-level reporting tools.
- Key SDK APIs: Lgsx_ReaderEnumRuns, Lgsx_ReaderMoveNextRun2, Lgsx_ReaderEnumSetupCamera, Lgsx_ReaderGetLUTImage2, Lgsx_ReaderEnumSetupCameraOfSetup
- Typical usage: `LgsxRunsExample -i testMobile.lgsx`

1.3.2.2.6 LgsxPointCloudExample

- Source file: `LgsxPointCloudExample.cpp`
- What it does: Reads point-cloud metadata and streams chunked point fields based on property masks.
- Description: This walkthrough demonstrates scalable point-cloud reading with dynamic field selection. It queries available property types, allocates matching typed buffers, and configures chunked iteration to control memory use. During iteration it reads field blocks, prints representative values, and reports metadata needed for downstream interpretation. Use this structure for analytics pipelines, custom exporters, or preprocessing jobs over large scan datasets.
- Key SDK APIs: Lgsx_ReaderGetMetadata, Lgsx_ReaderQueryPropertyTypes, Lgsx_ReaderEnumPoints, Lgsx_ReaderMoveNextFields, Lgsx_ReaderGetPropertyTypeInfo
- Typical usage: `LgsxPointCloudExample -i testMobile.lgsx`

1.3.2.2.7 LgsxGeoTagsExample

- Source file: `LgsxGeoTagsExample.cpp`
- What it does: Enumerates geotags, categories, fields, values, and attachment relationships.
- Description: This example demonstrates a complete geotag traversal from schema discovery to per-tag value extraction. It starts by building category and field catalogs, then iterates geotags and resolves anchor context for each entry. For every geotag it prints typed values, category mapping, and linked asset information to show full semantic context. Use this workflow when implementing annotation viewers, issue-tracking overlays, or QA rules around tagged entities.
- Key SDK APIs: Lgsx_ReaderEnumGeoTags, Lgsx_ReaderGetGeoTag, Lgsx_GeoTagGetAnchor, Lgsx_GeoTagGetCategoryEntry, Lgsx_ReaderGetGeoTagAssetHandle
- Typical usage: `LgsxGeoTagsExample -i testMobile.lgsx`

1.3.2.2.8 LgsxAssetsExample

- Source file: `LgsxAssetsExample.cpp`
- What it does: Enumerates project assets and inspects metadata, local files, and external URLs.
- Description: This tutorial shows how to traverse every project asset and classify how each payload is referenced. It first enumerates asset IDs, then resolves each handle and prints metadata together with local filenames or external URLs. The flow highlights how to branch behavior for embedded files versus remote resources during export or synchronization. Use this as a template for asset audits, dependency reports, and custom download or packaging tools.
- Key SDK APIs: Lgsx_ReaderEnumAssets, Lgsx_ReaderGetAssetHandle, Lgsx_AssetGetMetadata, Lgsx_AssetGetFilename, Lgsx_AssetGetUrl
- Typical usage: `LgsxAssetsExample -i testMobile.lgsx`

1.3.2.2.9 LgsxTargetsExample

- Source file: `LgsxTargetsExample.cpp`
- What it does: Enumerates project targets and reports geometry, labels, and associated metadata.
- Description: This walkthrough demonstrates target-focused inspection for quality and reporting scenarios. It enumerates targets, prints geometric attributes and labels, then resolves metadata attached to each target object. The flow is useful when comparing target distributions, validating naming conventions, and auditing target completeness. Use it as a starting point for target QA dashboards or downstream registration support tools.
- Key SDK APIs: Lgsx_ReaderEnumTarget, Lgsx_ReaderMoveNextTarget, Lgsx_ReaderGetCurrentTargetLabel, Lgsx_FreeHandle
- Typical usage: `LgsxTargetsExample -i testMobile.lgsx`

1.3.2.2.10 LgsxSitemapsExample

- Source file: `LgsxSitemapsExample.cpp`
- What it does: Enumerates sitemaps, sitemap items, and sitemap image access paths including active UCS context.
- Description: This example shows how to read sitemap topology together with image content options. It enumerates sitemap handles, resolves item lists, and prints item identity/type data needed for overlay reconstruction. For imagery, it demonstrates both decoded image reads and raw binary extraction so you can choose your own decode path. Use this approach for map viewers, background-map exporters, and sitemap integrity checks.
- Key SDK APIs: Lgsx_ReaderEnumSitemaps, Lgsx_ReaderGetSitemapHandle, Lgsx_SitemapEnumItems, Lgsx_SitemapImageRead, Lgsx_SitemapImageReadBinary
- Typical usage: `LgsxSitemapsExample -i testMobile.lgsx`

1.3.2.3 Advanced Examples

Composite workflows that combine multiple reader APIs to solve higher-level engineering tasks.

1.3.2.3.1 LgsxUcsExample

- Source file: `LgsxUcsExample.cpp`
- What it does: Demonstrates UCS listing and coordinate transformations for setups, geotags, and point clouds.
- Description: This tutorial demonstrates how to inspect available coordinate systems and transform project entities into a chosen UCS. It can list all coordinate systems, select one by name, and then apply conversions to setups, geotags, and sampled points. The example prints transformed coordinates so you can verify orientation, scale, and translation behavior end to end. Use this as a baseline for coordinate-validation tools, GIS integration, and custom transform pipelines.
- Key SDK APIs: Lgsx_ReaderEnumCoordSystems, Lgsx_ReaderMoveNextCoordSystems2, Lgsx_ReaderEnumSetups, Lgsx_ReaderEnumGeoTags, Lgsx_ReaderEnumPoints
- Typical usage: `LgsxUcsExample -i testUCS.lgsx -a | LgsxUcsExample -i testUCS.lgsx -u <ucs-name>`

1.3.2.3.2 LgsxSetupIndexExample

- Source file: `LgsxSetupIndexExample.cpp`
- What it does: Uses PM_SETUPIDX to attribute points to setups and waypoints, with optional subsampling.
- Description: This advanced example computes point ownership per setup using PM_SETUPIDX for mixed TLS and mobile datasets. It builds lookup maps for setups and run waypoints, then streams point attributes in chunks to keep memory bounded. During accumulation it aggregates counts per setup index and prints hierarchical summaries for attribution analysis. Use this pattern when you need reproducible per-setup attribution from raw point streams.
- Key SDK APIs: Lgsx_ReaderEnumPointsEx, Lgsx_ReaderQueryPropertyTypes, Lgsx_ReaderGetPropertyTypeInfo, Lgsx_ReaderEnumSetups, Lgsx_ReaderEnumRuns
- Typical usage: `LgsxSetupIndexExample -i testMobile.lgsx | LgsxSetupIndexExample -i testMobile.lgsx --subsample 0.1`

1.3.2.3.3 LgsxSensorProcessingInfoExample

- Source file: `LgsxSensorProcessingInfoExample.cpp`
- What it does: Demonstrates reading SensorInfo and ProcessingInfo for setups and runs.
- Description: This example shows how to read SensorInfo and ProcessingInfo data associated with setups (TLS and mobile) and runs using the LGSx SDK. It iterates all TLS setups, then all runs with their mobile setups, printing detailed SensorInfo and ProcessingInfo for each context. Use this to inspect sensor metadata such as serial number, scanner type, firmware version, and processing pipeline details including application name, version, and order index.
- Key SDK APIs: Lgsx_ReaderEnumSetups, Lgsx_ReaderMoveNextSetup, Lgsx_ReaderHasSetupSensorInfo, Lgsx_ReaderGetSetupSensorInfo, Lgsx_ReaderEnumSetupProcessingInfos, Lgsx_ReaderGetSetupProcessingInfo, Lgsx_ReaderEnumRuns, Lgsx_ReaderMoveNextRun, Lgsx_ReaderHasRunSensorInfo, Lgsx_ReaderGetRunSensorInfo, Lgsx_ReaderEnumRunProcessingInfos, Lgsx_ReaderGetRunProcessingInfo
- Typical usage: `LgsxSensorProcessingInfoExample -i <file.lgsx>`

1.3.2.4 Extraction Utilities

Export-oriented examples that transform project data into external files or image payloads.

1.3.2.4.1 LgsxExtractCubemapsExample

- Source file: `LgsxExtractCubemapsExample.cpp`
- What it does: Extracts cubemap image bytes for TLS and kinematic setups into output folders.
- Description: This extraction flow exports cubemap faces for both TLS setups and kinematic runs. It discovers available image layers, applies optional layer filtering, and iterates all requested resolution levels. For each face it creates a deterministic output path and writes raw image bytes so no additional decoding is required. Use this pattern when preparing imagery for rendering pipelines, offline previews, or texture-processing workflows.
- Key SDK APIs: Lgsx_ReaderGetImageLayerNames, Lgsx_ReaderGetCubemapImageType, Lgsx_ReaderGetCubemapMaxLev Lgsx_ReaderGetCubemapImageBytes, Lgsx_ReaderEnumRuns
- Typical usage: `LgsxExtractCubemapsExample -i testMobile.lgsx -o output/ | LgsxExtractCubemapsExample -i testMobile.lgsx -o output/ --layer-name Camera`

1.3.2.4.2 LgsxExtractSitemapsExample

- Source file: `LgsxExtractSitemapsExample.cpp`
- What it does: Extracts sitemap images and metadata, including decoded and binary image payloads.
- Description: This tutorial exports complete sitemap deliverables, including metadata, item listings, and image payloads. It iterates each sitemap, writes structured text summaries, and then exports background/overview/acceptance images. The flow supports both decoded reads and raw binary image output to match different downstream integration needs. Use this approach for packaging map artifacts for external viewers, review workflows, or archival outputs.
- Key SDK APIs: Lgsx_ReaderEnumSitemaps, Lgsx_ReaderGetSitemapHandle, Lgsx_SitemapEnumItems, Lgsx_SitemapImageRead, Lgsx_SitemapImageReadBinary
- Typical usage: `LgsxExtractSitemapsExample -i testMobile.lgsx -o sitemaps_↩ output`

1.3.2.4.3 LgsxExtractPointCloudExample

- Source file: `LgsxExtractPointCloudExample.cpp`
- What it does: Extracts point cloud content from LGS or LGSx into HSPC output.
- Description: This example presents two practical strategies for writing project point clouds to HSPC output. The first path performs direct reader-driven extraction, while the second streams through the point-cloud file API. Both paths share argument parsing and validation so you can switch extraction mode without changing external usage. Use this as a reference when you need reproducible point export with optional control over read/write internals.
- Key SDK APIs: Lgsx_ReaderExtractPointCloud, Lgsx_ReaderPointCloudFileOpen, Lgsx_PointCloudFileGetSize, Lgsx_PointCloudFileRead, Lgsx_PointCloudFileClose
- Typical usage: `LgsxExtractPointCloudExample -i testMobile.lgsx -o testMobile.hspca | LgsxExtractPointCloudExample -i testMobile.lgsx -o testMobile.hspca -f`

1.3.3 SDK Examples Snippets

This page highlights short code sections that implement key example workflows, with a brief explanation of what each snippet demonstrates.

1.3.3.1 Fundamentals

1.3.3.1.1 LgsxLicenseDiagnosticsExample

License diagnostics workflow

- Description: Executes default and optional named-license checks/load-unload flows and reports global SDK error state.

```
  const int initLegacy = Lgsx_InitLicense(license.c_str(), L"");
  const int initExtended = Lgsx_InitLicenseEx2(license.c_str(), L"", L"",
L"LgsxLicenseDiagnosticsExample");
  Console() « "Init results: InitLicense=" « initLegacy
            « ", InitLicenseEx2=" « initExtended « std::endl;

  const int isLicensed = Lgsx_IsLicensed();
  const int daysLeft = Lgsx_LicenseDaysLeft();
  Console() « "Default license status: isLicensed=" « isLicensed
            « ", daysLeft=" « daysLeft « std::endl;

  const int loadStatus = Lgsx_LoadLicense();
  const int unloadStatus = Lgsx_UnloadLicense();
  Console() « "Default load/unload status: load=" « loadStatus
            « ", unload=" « unloadStatus « std::endl;

#ifdef _WIN32
  const std::string licenseUtf8 = ToStdString(license.c_str());
  using IsLicensedExAFn = int (*)(const char*, const char*);
  using LicenseDaysLeftExAFn = int (*)(const char*, const char*);
  using LoadLicenseExAFn = int (*)(const char*, const char*);
  using UnloadLicenseExAFn = int (*)(const char*);

  const IsLicensedExAFn isLicensedExA = ResolveOptionalLgsxApi<IsLicensedExAFn>("Lgsx_IsLicensedExA");
  const LicenseDaysLeftExAFn licenseDaysLeftExA =
    ResolveOptionalLgsxApi<LicenseDaysLeftExAFn>("Lgsx_LicenseDaysLeftExA");
  const LoadLicenseExAFn loadLicenseExA =
ResolveOptionalLgsxApi<LoadLicenseExAFn>("Lgsx_LoadLicenseExA");
  const UnloadLicenseExAFn unloadLicenseExA =
ResolveOptionalLgsxApi<UnloadLicenseExAFn>("Lgsx_UnloadLicenseExA");

  if (isLicensedExA != nullptr && licenseDaysLeftExA != nullptr)
  {
```



```
    const int namedLicensed = isLicensedExA(licenseUtf8.c_str(), "");
    const int namedDaysLeft = licenseDaysLeftExA(licenseUtf8.c_str(), "");
    Console() « "Named license status: isLicensedExA=" « namedLicensed
              « ", daysLeftExA=" « namedDaysLeft « std::endl;
  }
  else
  {
    Console() « "Named license status APIs are unavailable in linked SDK binary." « std::endl;
  }

  if (loadLicenseExA != nullptr && unloadLicenseExA != nullptr)
  {
    const int namedLoad = loadLicenseExA(licenseUtf8.c_str(), "");
    const int namedUnload = unloadLicenseExA(licenseUtf8.c_str());
    Console() « "Named load/unload status: loadExA=" « namedLoad
              « ", unloadExA=" « namedUnload « std::endl;
  }
  else
  {
    Console() « "Named load/unload APIs are unavailable in linked SDK binary." « std::endl;
  }
#else
  Console() « "Named license *ExA diagnostics are unavailable on this platform." « std::endl;
#endif

  Console() « "Global last error no=" « Lgsx_GetLastErrorNo()
            « ", message=[" « ToOsString(Lgsx_GetLastError()) « "]"
            « std::endl;
```

1.3.3.1.2 LgsxMetadataRoundtripExample

Metadata clone and typed reads

- Description: Shows key metadata operations: clone mutation checks plus typed string and double-array reads.

```
  const std::string stringKey = FindFirstKeyByType(root, MetadataType_STRING);
  if (!stringKey.empty())
  {
    Console() « "Has key '" « ToOsString(stringKey.c_str()) « "' before remove = " «
Lgsx_MetaHas(cloned, stringKey.c_str()) « std::endl;
    Lgsx_MetaRemove(cloned, stringKey.c_str());
    Console() « "Has key '" « ToOsString(stringKey.c_str()) « "' after remove = " «
Lgsx_MetaHas(cloned, stringKey.c_str()) « std::endl;

    wchar_t valueBuffer[256] = {0};
    const int getStringStatus = Lgsx_MetaGetString(root, stringKey.c_str(), valueBuffer, 256);
    Console() « "GetString status=" « getStringStatus
              « " key='" « ToOsString(stringKey.c_str()) « "' value=[" « ToOsString(valueBuffer) « "]"
« std::endl;
  }
  else
  {
    Console() « "No string metadata key found in project metadata." « std::endl;
  }

  const std::string arrayKey = FindFirstKeyByType(root, MetadataType_DOUBLEARRAY);
  if (!arrayKey.empty())
  {
    int arraySize = 0;
    const int sizeStatus = Lgsx_MetaGetDoubleArraySize(root, arrayKey.c_str(), &arraySize);
    Console() « "GetDoubleArraySize status=" « sizeStatus
              « " key='" « ToOsString(arrayKey.c_str()) « "' count=" « arraySize « std::endl;

    if (sizeStatus == 0 && arraySize > 0)
    {
      std::vector<double> values(static_cast<size_t>(arraySize), 0.0);
      int valuesCount = arraySize;
      const int getArrayStatus = Lgsx_MetaGetDoubleArray(root, arrayKey.c_str(), values.data(),
&valuesCount);
      Console() « "GetDoubleArray status=" « getArrayStatus
                « " key='" « ToOsString(arrayKey.c_str()) « "' count=" « valuesCount « std::endl;
    }
  }
  else
  {
    Console() « "No double-array metadata key found in project metadata." « std::endl;
  }
```


1.3.3.1.3 LgsxOpenFileExample

Reader open and close lifecycle

- Description: Opens an LGSx file and demonstrates safe reader cleanup. Shows the baseline lifecycle pattern for building SDK tools.

```
LgsxReaderPtr reader = nullptr;
const int openResult = OpenReaderChecked(args.mInputFileName, args.mPassword, &reader);
if (openResult != 0)
{
  return openResult;
}

Console() « "Open-file flow completed." « std::endl;

CloseReaderChecked(&reader);
Console() « "Reader closed." « std::endl;
```

1.3.3.1.4 LgsxProjectInfoExample

CLI and reader bootstrap

- Description: Performs shared CLI, licensing, and guarded execution plumbing. Shows how to structure a reusable host program for reader-based tools.

```
CommonCliArguments args;
ArgumentParser parser;

AddCommonSwitches(parser, args);
AddCommonLicenseAndPasswordParameters(parser, args);
AddCommonInputParameter(parser, args, true);

switch (FinalizeCommonArguments(parser, argc, argv, args))
{
case ExampleParseResult::SUCCESS:
  break;
case ExampleParseResult::STOP:
  return 0;
case ExampleParseResult::FAIL:
default:
  return 1;
}

Console() « "LGSx ProjectInfo example started. Logging is "
          « (args.mEnableLogging ? "enabled." : "disabled.") « std::endl;

int hRes = InitializeSdkWithLicense(args, L"LgsxProjectInfoExample");
if (hRes != 0)
{
  return hRes;
}

try
{
  hRes = LgsxProjectInfoExample(args);
}
catch (const std::exception& e)
{
  Console() « "Unhandled exception: " « e.what() « std::endl;
  hRes = 1;
}
catch (...)
{
  Console() « "Unhandled unknown exception." « std::endl;
  hRes = 1;
}

Lgsx_Uninitialize();
Console() « "Finished LGSx ProjectInfo example." « std::endl;
```

Project metadata readout

- Description: Retrieves project information and formats key metadata fields. Shows how to access project metadata before moving to deeper workflows.

```
LgsxReaderPtr reader = nullptr;
const int openResult = OpenReaderChecked(args.mInputFileName, args.mPassword, &reader);
if (openResult != 0)
{
  return openResult;
}

LgsxMetadataPtr projectInfo = nullptr;
const int projectInfoResult =
  Lgsx_ReaderGetProjectInfo(reader, &projectInfo, nullptr, nullptr, nullptr, nullptr, nullptr);
if (projectInfoResult != 0 || projectInfo == nullptr)
{
  Console() « "Failed to query project information." « std::endl;
  CloseReaderChecked(&reader);
  return 1;
}

Console() « "Highlighted project metadata:" « std::endl;
PrintProjectMetaValue(projectInfo, "  Project:       ", "Project");
PrintProjectMetaValue(projectInfo, "  ProjectVersion:", "ProjectVersion");
PrintProjectMetaValue(projectInfo, "  Publisher:     ", "Publisher");
PrintProjectMetaValue(projectInfo, "  Company:       ", "Company");
PrintProjectMetaValue(projectInfo, "  CompanyUrl:    ", "Company URL");

Console() « "All project metadata:" « std::endl;
PrintMetadata(projectInfo, 1);

Lgsx_FreeHandle(projectInfo);
CloseReaderChecked(&reader);
```

1.3.3.2 Focused Reader Examples

1.3.3.2.1 LgsxProjectDescriptionExample

Project description and migrated GUID mapping

- Description: Queries project description with fixed and allocated buffers and resolves migrated metadata fields to GUIDs.

```
char err[256] = {0};
Lgsx_ReaderGetLastError(reader, err, 256);
Console() « "ReaderGetLastError=[" « ToOsString(err) « "]" « std::endl;

wchar_t desc[512] = {0};
Console() « "GetProjectDescription=" « Lgsx_ReaderGetProjectDescription(reader, desc, 512)
          « " value=[" « ToOsString(desc) « "]" « std::endl;

wchar_t* desc2 = nullptr;
Console() « "GetProjectDescription2=" « Lgsx_ReaderGetProjectDescription2(reader, &desc2);
if (desc2 != nullptr)
{
  Console() « " value=[" « ToOsString(desc2) « "]";
  LgsxUtil_FreeMem((void**)&desc2);
}
Console() « std::endl;

for (int i = 0; i < 3; ++i)
{
  const char* guid = nullptr;
  Console() « "GetMigratedFieldGuid(" « i « ")="
            « Lgsx_GetMigratedFieldGuid(static_cast<LgsxMigratedMetaKeyField>(i), &guid)
            « " guid=[" « ToOsString(guid != nullptr ? guid : "") « "]"
            « std::endl;
}
```

1.3.3.2.2 LgsxModelInfoExample

Model node and model info traversal

- Description: Shows node-to-model traversal using paired legacy/extended node and model info APIs.

```
Lgsx_ReaderGetModelNodeInfo(reader, nodeIds[i], &n1, &childIds);
Lgsx_ReaderGetModelNodeInfo2(reader, nodeIds[i], &n2, &childIds, &fullName);

if (n1.modelId != 0)
{
  CYMODELINFO m1 = {};
  CYMODELINFO m2 = {};
  LgsxMetadataPtr meta1 = nullptr;
  LgsxMetadataPtr meta2 = nullptr;
  uint64_t* refAssets = nullptr;
  wchar_t* modelName = nullptr;

  Lgsx_ReaderGetModelInfo(reader, n1.modelId, &m1, &meta1, &refAssets);
  Lgsx_ReaderGetModelInfo2(reader, n1.modelId, &m2, &meta2, &refAssets, &modelName);

  if (meta1 != nullptr)
  {
    Lgsx_FreeHandle(meta1);
  }
  if (meta2 != nullptr)
  {
    Lgsx_FreeHandle(meta2);
  }
  if (modelName != nullptr)
  {
    LgsxUtil_FreeMem((void**)&modelName);
  }
  if (refAssets != nullptr)
  {
    LgsxUtil_FreeMem((void**)&refAssets);
  }
}
```

1.3.3.2.3 LgsxRunScansExample

Run scan timestamp traversal

- Description: Demonstrates the core run-scan loop and timestamp extraction for each scan.

```
int scanCount = 0;
Lgsx_ReaderEnumRunScans(reader, &scanCount);
Console() « "Run " « runId « " scans=" « scanCount « std::endl;

for (int s = 0; s < scanCount; ++s)
{
  ScanHandle scan = {};
  if (Lgsx_ReaderGetRunScan(reader, s, &scan) != 0)
  {
    continue;
  }

  uint64_t created = 0;
  uint64_t modified = 0;
  LgsxMetadataPtr capture = nullptr;
  Lgsx_ScanGetCreationTimestamp(scan, &created);
  Lgsx_ScanGetModificationTimestamp(scan, &modified);
  Lgsx_ScanGetCaptureInfoSnapshot(scan, &capture);

  Console() « "  Scan " « s
            « " created=" « FormatTimestamp(created)
            « " modified=" « FormatTimestamp(modified)
            « " hasCapture=" « (capture != nullptr)
            « std::endl;

  Lgsx_ScanRelease(&scan);
}

if (runMeta != nullptr)
{
  Lgsx_FreeHandle(runMeta);
}
```

1.3.3.2.4 LgsxSetupsExample

Setup image fallback

- Description: Reads pano imagery first, then falls back to cube imagery. Shows a resilient image-access flow for setup visualization.

```
wchar_t layers[128] = {};
const int layerCount = Lgsx_ReaderGetImageLayerNames(reader, layers, 128);
Console() « "    Setup " « setupId « " image layers=" « layerCount;
if (layerCount > 0)
{
  Console() « " names=[" « ToOsString(layers) « "]";
}
Console() « std::endl;

int width = 0;
int height = 0;
int widthInBytes = 0;
ImagePixelType pixelType = {};
unsigned char* panoImage = nullptr;

const int panoResult = Lgsx_ReaderGetPanoImage(
  reader,
  &width,
  &height,
  &widthInBytes,
  &pixelType,
  &panoImage);
if (panoResult == 0)
{
  Console() « "      Pano image " « width « "x" « height « std::endl;
  return;
}

SCyRgdBdyTxf txf = {};
unsigned char* cubeFaces[6] = {};
const int cubeResult = Lgsx_ReaderGetCubeImage(
  reader,
  L"Camera",
  &txf,
  &width,
  &height,
  &widthInBytes,
  &pixelType,
  &cubeFaces[0]);
if (cubeResult == 0)
{
  Console() « "      Cube image " « width « "x" « height « std::endl;
}
```

Setup traversal workflow

- Description: Executes setup enumeration and setup-related data traversal. Shows how setup-centric features are connected in project data.

```
int PrintSetups(LgsxReaderPtr reader)
{
  int setupCount = 0;
  const int enumResult = Lgsx_ReaderEnumSetups(reader, &setupCount);
  if (enumResult != 0)
  {
    Console() « "Failed to enumerate setups." « std::endl;
    return 1;
  }

  Console() « "Setup count = " « setupCount « std::endl;
  if (setupCount == 0)
  {
    return 0;
  }

  uint64_t setupId = 0;
  CYSETUPINFO setup = {};
  LgsxMetadataPtr metadata = nullptr;
  while (Lgsx_ReaderMoveNextSetup(reader, &setupId, &setup, &metadata) == 0)
  {
    wchar_t* setupName = nullptr;
    Lgsx_ReaderGetCurrentSetupName(reader, &setupName);

    Console() « "  Setup " « setup.setupIndex
              « " [" « ToOsString(setupName != nullptr ? setupName : L"") « "]"
              « " id=" « setupId
              « " time=" « setup.time
              « " position=" « setup.pose.mDx « "," « setup.pose.mDy « "," « setup.pose.mDz
              « std::endl;

    LgsxUtil_FreeMem((void**)&setupName);

    if (metadata != nullptr)
```



```
    {
      PrintMetadata(metadata, 2);
      Lgsx_FreeHandle(metadata);
      metadata = nullptr;
    }

    PrintSetupTargets(reader, setupId);
    PrintSetupRelatedGeoTags(reader, setupId);
    PrintImageLayers(reader, setupId);
  }

  Lgsx_ReaderEndSetups(reader);
  return 0;
}
```

1.3.3.2.5 LgsxRunsExample

Run setup-camera traversal

- Description: Traverses run setups and drills into setup-camera subelements. Shows nested mobile-run traversal with graceful image fallback.

```
int setupCount = 0;
const int enumResult = Lgsx_ReaderEnumSetups(reader, &setupCount);
Console() « "  Setup count in Run " « runId « " = " « setupCount « std::endl;
if (enumResult != 0 || setupCount <= 0)
{
  return;
}

uint64_t setupId = 0;
CYSETUPINFO setup = {};
while (Lgsx_ReaderMoveNextSetup(reader, &setupId, &setup, nullptr) == 0)
{
  wchar_t* setupName = nullptr;
  Lgsx_ReaderGetCurrentSetupName(reader, &setupName);
  Console() « "    Setup " « setup.setupIndex
            « " [" « ToOsString(setupName != nullptr ? setupName : L"") « "]"
            « " id=" « setupId
            « " time=" « setup.time
            « " vIdx=" « setup.vertexIndex
            « std::endl;
  LgsxUtil_FreeMem((void**)&setupName);

  int setupCameraCount = 0;
  const int enumCameraResult = Lgsx_ReaderEnumSetupCamera(reader, &setupCameraCount);
  if (enumCameraResult == 0 && setupCameraCount > 0)
  {
    if (totalSetupCameras != nullptr)
    {
      *totalSetupCameras += setupCameraCount;
    }
    if (setupsWithCameras != nullptr)
    {
      setupsWithCameras->insert(setupId);
    }

    uint64_t setupCameraId = 0;
    CYSETUPCAMERAINFO setupCameraInfo = {};
    while (Lgsx_ReaderMoveNextSetupCamera(reader, &setupCameraId, &setupCameraInfo, nullptr) == 0)
    {
      PrintSetupCameraInfoBrief(reader, setupCameraId);
    }
    Lgsx_ReaderEndSetupCameras(reader);
  }
  else
  {
    PrintSetupImageFallback(reader, setupId);
  }
}

Lgsx_ReaderEndSetups(reader);
```

Setup-camera integrity check

- Description: Verifies setup-camera totals with enum-by-setup traversal. Shows a practical consistency check for run camera completeness.

```
if (expectedSetupCameraCount <= 0)
{
  return;
}

int remainingCount = expectedSetupCameraCount;
for (const uint64_t setupId : setupsWithCameras)
{
  int setupCameraCount = 0;
  if (Lgsx_ReaderEnumSetupCameraOfSetup(reader, setupId, &setupCameraCount) != 0)
  {
    continue;
  }

  uint64_t setupCameraId = 0;
  CYSETUPCAMERAINFO setupCameraInfo = {};
  while (Lgsx_ReaderMoveNextSetupCamera(reader, &setupCameraId, &setupCameraInfo, nullptr) == 0)
  {
    --remainingCount;
  }
  Lgsx_ReaderEndSetupCameras(reader);
}

if (remainingCount == 0)
{
  Console() « "  Setup-camera enum-by-setup integrity check: OK" « std::endl;
}
else
{
  Console() « "  Setup-camera enum-by-setup integrity check: FAILED, delta="
            « remainingCount « std::endl;
}
```

Run traversal workflow

- Description: Performs run traversal and prints trajectory plus camera details. Shows how mobile/kinematic content is structured and how to validate it.

```
int PrintRuns(LgsxReaderPtr reader)
{
  int runCount = 0;
  const int enumResult = Lgsx_ReaderEnumRuns(reader, &runCount);
  if (enumResult != 0)
  {
    Console() « "Failed to enumerate runs." « std::endl;
    return 1;
  }

  Console() « "Run count = " « runCount « std::endl;
  if (runCount == 0)
  {
    return 0;
  }

  uint64_t runId = 0;
  int vertexCount = 0;
  CYTRAJECTORYINFO trajectory = {};
  CYTRAJECTORYVERTEX* trajectoryPath = nullptr;
  LgsxMetadataPtr metadata = nullptr;
  wchar_t* jobName = nullptr;
  wchar_t* runName = nullptr;
  int totalSetupCameras = 0;
  std::set<uint64_t> setupsWithCameras;

  while (Lgsx_ReaderMoveNextRun2(
           reader,
           &runId,
           &trajectory,
           &vertexCount,
           &trajectoryPath,
           &metadata,
           &jobName,
           &runName) == 0)
  {
    char* runGuid = nullptr;
    if (Lgsx_ReaderRunGetGuid(reader, &runGuid) != 0)
    {
      runGuid = nullptr;
    }

    Console() « "Run " « runId;
```



```
    if (runGuid != nullptr)
    {
      Console() « " guid=[" « ToOsString(runGuid) « "]";
    }

    const wchar_t* effectiveJobName = (jobName != nullptr) ? jobName : trajectory.jobName;
    if (effectiveJobName != nullptr && effectiveJobName[0] != 0)
    {
      Console() « " job=[" « ToOsString(effectiveJobName) « "]";
    }

    const wchar_t* effectiveRunName = (runName != nullptr) ? runName : trajectory.runName;
    Console() « " name=[" « ToOsString(effectiveRunName != nullptr ? effectiveRunName : L"") « "]";
    Console() « " vertices=" « vertexCount « std::endl;

    if (metadata != nullptr)
    {
      PrintMetadata(metadata, 2);
      Lgsx_FreeHandle(metadata);
      metadata = nullptr;
    }

    PrintLutImageSummary(reader);
    PrintVertexSample(trajectoryPath, vertexCount);
    PrintRunSetupsAndSubelements(reader, runId, &totalSetupCameras, &setupsWithCameras);

    LgsxUtil_FreeMem((void**)&runGuid);
    LgsxUtil_FreeMem((void**)&jobName);
    LgsxUtil_FreeMem((void**)&runName);
  }

  Lgsx_ReaderEndRuns(reader);
  Console() « "  Total setup-camera count seen in runs = " « totalSetupCameras « std::endl;
  VerifySetupCameraEnumeration(reader, totalSetupCameras, setupsWithCameras);
  return 0;
}
```

1.3.3.2.6 LgsxPointCloudExample

Property-mask field wiring

- Description: Maps property-mask bits to typed point buffers before streaming. Shows how to bind optional fields and query byte widths safely.

```
void* pointData[16] = {};
if ((propertyMask & PM_XYZ_DATA) != 0)
{
  pointData[0] = xyz.data();
}
if ((propertyMask & PM_INTENSITY_DATA) != 0)
{
  pointData[1] = intensity.data();
}
if ((propertyMask & PM_COLOR_DATA) != 0)
{
  pointData[2] = color.data();
}
if ((propertyMask & PM_NORMAL_DATA) != 0)
{
  pointData[3] = normal.data();
}

int classificationBytes = 0;
if ((propertyMask & PM_CLASSIFICATION) != 0
    && Lgsx_ReaderGetPropertyTypeInfo(reader, PM_CLASSIFICATION, L"", &classificationBytes) == 0)
{
  pointData[4] = classification.data();
  Console() « "Classification property size = " « classificationBytes « " bytes" « std::endl;
}

int setupIndexBytes = 0;
if ((propertyMask & PM_SETUPIDX) != 0
    && Lgsx_ReaderGetPropertyTypeInfo(reader, PM_SETUPIDX, L"", &setupIndexBytes) == 0)
{
  pointData[5] = setupIndices.data();
  Console() « "Setup-index property size = " « setupIndexBytes « " bytes" « std::endl;
}

if (Lgsx_ReaderEnumPoints(reader, propertyMask, 1, true) != 0)
{
  Console() « "Failed to start point-cloud enumeration." « std::endl;
  return 0;
}
```


Chunked point iteration

- Description: Streams points in bounded chunks and reports progress plus totals. Shows a memory-safe loop pattern for large point-cloud datasets.

```
  uint64_t totalRead = 0;
  int totalChunksRead = 0;
  while (true)
  {
    const int pointsRead = Lgsx_ReaderMoveNextFields(reader, chunkSize, BitmapFormat__RGBA,
pointData);
    if (pointsRead <= 0)
    {
      break;
    }

    Console() « "Read " « pointsRead « " points in chunk." « std::endl;
    if (totalRead == 0 && pointData[0] != nullptr)
    {
      Console() « "First point = "
                « FormatDouble(xyz[0]) « ","
                « FormatDouble(xyz[1]) « ","
                « FormatDouble(xyz[2]) « std::endl;
    }

    totalRead += static_cast<uint64_t>(pointsRead);
    ++totalChunksRead;
    if (maxReadChunks > 0 && totalChunksRead >= maxReadChunks)
    {
      Console() « "... additional chunks skipped" « std::endl;
      break;
    }
  }

  if (totalRead == 0)
  {
    Console() « "No points returned from query." « std::endl;
  }
  else
  {
    Console() « "Total points read = " « totalRead « std::endl;
  }
```

Point streaming workflow

- Description: Configures point-field buffers and executes chunked point iteration. Shows scalable patterns for large point-cloud reads.

```
  LgsxReaderPtr reader = nullptr;
  const int openResult = OpenReaderChecked(args.mInputFileName, args.mPassword, &reader);
  if (openResult != 0)
  {
    return openResult;
  }

  uint64_t pointCount = 0;
  CYBBOX bbox = {};
  LgsxMetadataPtr metadata = nullptr;
  if (Lgsx_ReaderGetMetadata(reader, &pointCount, &bbox, &metadata) != 0)
  {
    Console() « "Failed to query project metadata for point cloud." « std::endl;
    CloseReaderChecked(&reader);
    return 1;
  }

  Console() « "Total point count = " « pointCount « std::endl;
  Console() « "Bounding box: "
            « FormatDouble(bbox.minCorner.x) « ","
            « FormatDouble(bbox.minCorner.y) « ","
            « FormatDouble(bbox.minCorner.z) « " -> "
            « FormatDouble(bbox.maxCorner.x) « ","
            « FormatDouble(bbox.maxCorner.y) « ","
            « FormatDouble(bbox.maxCorner.z) « std::endl;
  if (metadata != nullptr)
  {
    PrintMetadata(metadata, 1);
    Lgsx_FreeHandle(metadata);
  }

  const int propertyMask = Lgsx_ReaderQueryPropertyTypes(reader);
```



```
  Console() « "Point-cloud property mask = 0x" « std::hex « propertyMask « std::dec « std::endl;

  if ((propertyMask & PM_CLASSIFICATION) != 0)
  {
    PrintClassificationModels(reader);
  }

  Console() « "Reading points..." « std::endl;
  ReadPointCloudFields(reader, propertyMask);
  CloseReaderChecked(&reader);
```

1.3.3.2.7 LgsxGeoTagsExample

Geotag schema discovery

- Description: Enumerates category definitions and value catalogs with metadata. Shows how to discover geotag schema before reading tag instances.

```
  int categoryCount = 0;
  if (Lgsx_ReaderEnumCategories(reader, &categoryCount) != 0)
  {
    Console() « "Failed to enumerate categories." « std::endl;
    return;
  }

  Console() « "Category count = " « categoryCount « std::endl;
  for (int index = 0; index < categoryCount; ++index)
  {
    CategoryHandle category = {};
    if (Lgsx_ReaderGetCategory(reader, index, &category) != 0)
    {
      continue;
    }

    const char* categoryGuid = nullptr;
    const wchar_t* categoryName = nullptr;
    int categoryPriority = 0;
    uint64_t creationTimestamp = 0;
    uint64_t modificationTimestamp = 0;
    Lgsx_CategoryGetGuid(category, &categoryGuid);
    Lgsx_CategoryGetName(category, &categoryName);
    Lgsx_CategoryGetPriority(category, &categoryPriority);
    Lgsx_CategoryGetCreationTimestamp(category, &creationTimestamp);
    Lgsx_CategoryGetModificationTimestamp(category, &modificationTimestamp);

    Console() « "  Category #" « index
              « " guid=[" « ToOsString(categoryGuid != nullptr ? categoryGuid : "") « "]"
              « " name=[" « ToOsString(categoryName != nullptr ? categoryName : L"") « "]"
              « " priority=" « categoryPriority
              « " created=" « FormatTimestamp(creationTimestamp)
              « " modified=" « FormatTimestamp(modificationTimestamp)
              « std::endl;

    int valueCount = 0;
    if (Lgsx_CategoryEnumValues(category, &valueCount) == 0 && valueCount > 0)
    {
      Console() « "    Values=" « valueCount « std::endl;
      for (int valueIndex = 0; valueIndex < valueCount; ++valueIndex)
      {
        CategoryValueHandle categoryValue = {};
        if (Lgsx_CategoryGetValue(category, valueIndex, &categoryValue) != 0)
        {
          continue;
        }

        const char* valueGuid = nullptr;
        const wchar_t* valueText = nullptr;
        int valuePriority = 0;
        uint64_t valueCreationTimestamp = 0;
        uint64_t valueModificationTimestamp = 0;
        if (Lgsx_CategoryValueGetGuid(categoryValue, &valueGuid) != 0
            || Lgsx_CategoryValueGetValue(categoryValue, &valueText) != 0
            || Lgsx_CategoryValueGetPriority(categoryValue, &valuePriority) != 0
            || Lgsx_CategoryValueGetCreationTimestamp(categoryValue, &valueCreationTimestamp) != 0
            || Lgsx_CategoryValueGetModificationTimestamp(categoryValue, &valueModificationTimestamp)
!= 0)
        {
          continue;
        }

        bool hasColor = false;
```



```
      CYRGBA valueColor = {0, 0, 0, 0};
      if (Lgsx_CategoryValueHasColor(categoryValue, &hasColor) == 0 && hasColor)
      {
        Lgsx_CategoryValueGetColor(categoryValue, &valueColor);
      }

      Console() « "      Value #" « valueIndex
                « " guid=[" « ToOsString(valueGuid != nullptr ? valueGuid : "") « "]"
                « " text=[" « ToOsString(valueText != nullptr ? valueText : L"") « "]"
                « " priority=" « valuePriority
                « " created=" « FormatTimestamp(valueCreationTimestamp)
                « " modified=" « FormatTimestamp(valueModificationTimestamp);
      if (hasColor)
      {
        Console() « " color=("
                  « static_cast<int>(valueColor.r) « ","
                  « static_cast<int>(valueColor.g) « ","
                  « static_cast<int>(valueColor.b) « ","
                  « static_cast<int>(valueColor.a) « ")";
      }
      Console() « std::endl;
    }
  }

  Lgsx_CategoryRelease(&category);
}
```

Geotag traversal workflow

- Description: Executes full geotag traversal with category and attachment inspection. Shows how geotag data is modeled and how entities are related.

```
LgsxReaderPtr reader = nullptr;
const int openResult = OpenReaderChecked(args.mInputFileName, args.mPassword, &reader);
if (openResult != 0)
{
  return openResult;
}

PrintCategoryCatalog(reader);
PrintFieldCatalog(reader);
PrintGeoTags(reader);

CloseReaderChecked(&reader);
```

1.3.3.2.8 LgsxAssetsExample

Asset transport branch

- Description: Branches asset handling for external URLs versus local files. Shows how to consume both transport modes in one traversal.

```
if (isExternal)
{
  const wchar_t* url = nullptr;
  Lgsx_AssetGetUrl(assetHandle, &url);
  Console() « "  Asset " « assetId
            « " name=[" « ToOsString(name != nullptr ? name : L"") « "]"
            « " type=[" « ToOsString(type != nullptr ? type : L"") « "]"
            « " mime=[" « ToOsString(mimeType != nullptr ? mimeType : L"") « "]"
            « " url=[" « ToOsString(url != nullptr ? url : L"") « "]"
            « std::endl;
}
else
{
  const wchar_t* filename = nullptr;
  Lgsx_AssetGetFilename(assetHandle, &filename);
  Console() « "  Asset " « assetId
            « " name=[" « ToOsString(name != nullptr ? name : L"") « "]"
            « " type=[" « ToOsString(type != nullptr ? type : L"") « "]"
            « " mime=[" « ToOsString(mimeType != nullptr ? mimeType : L"") « "]"
            « " file=[" « ToOsString(filename != nullptr ? filename : L"") « "]"
            « std::endl;

  if (filename != nullptr && filename[0] != 0)
  {
```



```
        FILE* tempFile = fopen(_A(filename), "rb");
        if (tempFile != nullptr)
        {
          fseek(tempFile, 0L, SEEK_END);
          const long fileSize = ftell(tempFile);
          fclose(tempFile);
          Console() « "    temp file size=" « fileSize « std::endl;
        }
      }
    }
```

Asset enumeration workflow

- Description: Traverses all assets and prints transport plus metadata details. Shows how to consume mixed local-file and external-URL asset references.

```
LgsxReaderPtr reader = nullptr;
const int openResult = OpenReaderChecked(args.mInputFileName, args.mPassword, &reader);
if (openResult != 0)
{
  return openResult;
}

PrintProjectAssets(reader);
CloseReaderChecked(&reader);
```

1.3.3.2.9 LgsxTargetsExample

Target enumeration workflow

- Description: Iterates targets and prints key geometric attributes. Shows a direct template for building target QA or reporting tools.

```
void PrintTargets(LgsxReaderPtr reader)
{
  int targetCount = 0;
  if (Lgsx_ReaderEnumTarget(reader, &targetCount) != 0)
  {
    Console() « "Failed to enumerate targets." « std::endl;
    return;
  }

  Console() « "Target count = " « targetCount « std::endl;
  uint64_t targetId = 0;
  CYTARGETINFO targetInfo = {};
  LgsxMetadataPtr metadata = nullptr;
  while (Lgsx_ReaderMoveNextTarget(reader, &targetId, &targetInfo, &metadata) == 0)
  {
    wchar_t* targetLabel = nullptr;
    Lgsx_ReaderGetCurrentTargetLabel(reader, &targetLabel);

    Console() « "  Target id=" « targetId
              « " name=[" « ToOsString(targetLabel != nullptr ? targetLabel : L"") « "]"
              « " type=" « targetInfo.type
              « " origin=" « targetInfo.origin.x « ","
              « targetInfo.origin.y « ","
              « targetInfo.origin.z
              « " normal=" « targetInfo.normal.x « ","
              « targetInfo.normal.y « ","
              « targetInfo.normal.z
              « " radius=" « targetInfo.radius
              « std::endl;

    LgsxUtil_FreeMem((void**)&targetLabel);

    if (metadata != nullptr)
    {
      PrintMetadata(metadata, 2);
      Lgsx_FreeHandle(metadata);
      metadata = nullptr;
    }
  }

  Lgsx_ReaderEndTargets(reader);
}
```


1.3.3.2.10 LgsxSitemapsExample

Sitemap image read modes

- Description: Reads decoded sitemap images and inspects original binary payloads. Shows when to use decoded pixels versus binary transfer data.

```
const SitemapImageType imageTypes[] = {
  SitemapImageType_Background,
  SitemapImageType_Overview,
  SitemapImageType_Acceptance
};
for (const SitemapImageType imageType : imageTypes)
{
  bool hasImage = false;
  if (Lgsx_SitemapHasImage(sitemap, imageType, &hasImage) != 0 || !hasImage)
  {
    continue;
  }

  int width = 0;
  int height = 0;
  int widthInBytes = 0;
  ImagePixelType pixelType = {};
  unsigned char* imageBytes = nullptr;
  if (Lgsx_SitemapImageRead(
        reader,
        sitemap,
        imageType,
        &width,
        &height,
        &widthInBytes,
        &pixelType,
        &imageBytes) == 0)
  {
    Console() << "    Image (" << SitemapImageTypeName(imageType) << "): "
              << width << "x" << height
              << " bytes/row=" << widthInBytes
              << " pixelType=" << pixelType
              << std::endl;
  }

  bool isBlank = false;
  if (Lgsx_SitemapImageGetIsBlank(reader, sitemap, imageType, &isBlank) == 0)
  {
    Console() << "    Image " << SitemapImageTypeName(imageType)
              << " isBlank=" << (isBlank ? "true" : "false")
              << std::endl;
  }

  CYSITEMAPIMAGECORNERS2D corners = {};
  if (Lgsx_SitemapImageGetCorners(reader, sitemap, imageType, &corners, nullptr) == 0)
  {
    Console() << "      Corners: topLeft(" << corners.topLeft.x << "," << corners.topLeft.y
              << ") topRight(" << corners.topRight.x << "," << corners.topRight.y
              << ") bottomRight(" << corners.bottomRight.x << "," << corners.bottomRight.y
              << ") bottomLeft(" << corners.bottomLeft.x << "," << corners.bottomLeft.y << ")"
              << std::endl;
  }

  wchar_t* originalImageType = nullptr;
  int originalWidth = 0;
  int originalHeight = 0;
  int originalSize = 0;
  unsigned char* originalBytes = nullptr;
  if (Lgsx_SitemapImageReadBinary(
        reader,
        sitemap,
        imageType,
        &originalImageType,
        &originalWidth,
        &originalHeight,
        &originalBytes,
        &originalSize) != 0)
  {
    Console() << "    Original-image-bytes path (" << SitemapImageTypeName(imageType)
              << ") is not implemented yet." << std::endl;
  }

  if (originalImageType != nullptr)
  {
    LgsxUtil_FreeMem((void**)&originalImageType);
  }
}
```


Sitemap traversal workflow

- Description: Traverses sitemap content and inspects sitemap image and item metadata. Shows how map overlays and sitemap entities are represented.

```
LgsxReaderPtr reader = nullptr;
const int openResult = OpenReaderChecked(args.mInputFileName, args.mPassword, &reader);
if (openResult != 0)
{
  return openResult;
}

const int result = PrintSitemaps(reader);
CloseReaderChecked(&reader);
```

1.3.3.3 Advanced Examples

1.3.3.3.1 LgsxUcsExample

Point-field chunk streaming

- Description: Streams point chunks from the active point enumeration and reads XYZ/intensity/color payloads via Lgsx_ReaderMoveNextFields.

```
Console() « "Point Cloud (first " « maxPoints « " points):" « std::endl;

const int chunkSize = maxPoints * POINT_CLOUD_CHUNK_MULTIPLIER;
Point3D* points = new Point3D[chunkSize];
float* intensities = new float[chunkSize];
int* colors = new int[chunkSize];
float* normals = new float[NORMAL_VECTOR_COMPONENTS * chunkSize];

void* ppData[16];
std::memset(ppData, 0, sizeof(ppData));
ppData[0] = points;
ppData[1] = intensities;
ppData[2] = colors;
ppData[3] = normals;

int pointCount = 0;
while (pointCount < maxPoints)
{
  int fill = Lgsx_ReaderMoveNextFields(reader, chunkSize, BitmapFormat__RGBA, ppData);
  if (fill == 0)
  {
    break;
  }

  int pointsToProcess = (fill < (maxPoints - pointCount)) ? fill : (maxPoints - pointCount);

  for (int i = 0; i < pointsToProcess; ++i)
  {
    Vector3D prjPoint(points[i].X, points[i].Y, points[i].Z);
    Vector3D ucsPoint = ApplyUcsTransformation(prjPoint, coordTxf);

    Console() « "  Point " « (pointCount + i) « ":" « std::endl;
    Console() « "    - [prj] X=" « prjPoint.X « ", Y=" « prjPoint.Y « ", Z=" « prjPoint.Z «
std::endl;
    Console() « "    - [ucs] X=" « ucsPoint.X « ", Y=" « ucsPoint.Y « ", Z=" « ucsPoint.Z «
std::endl;

    if (intensities[i] >= 0.0f)
    {
      Console() « "    - intensity: " « intensities[i] « std::endl;
    }
  }

  pointCount += pointsToProcess;

  if (pointCount >= maxPoints)
  {
    break;
  }
}
```

UCS coordinate-system enumeration

- Description: Enumerates available coordinate systems and reads name/WKT/transform data using reader UCS APIs.

```
  Console() « "List of available UCS:" « std::endl;

  // Use JsReader APIs to get more info (WKT string)
  int numCs = 0;
  int hRes = Lgsx_ReaderEnumCoordSystems(pReader, &numCs);
  if (hRes == 0 && numCs > 0)
  {
    wchar_t* csName = nullptr;
    wchar_t* csWKT = nullptr;
    RigidTxf txf;
    bool isActive;
    while (0 == Lgsx_ReaderMoveNextCoordSystems2(pReader, &isActive, &csName, &csWKT,
(SCyRgdBdyTxf*)&txf))
    {
      if (isActive)
      {
        Console() « "+ ";
      }
      else
      {
        Console() « "- ";
      }
      Console() « ToOsString(csName != nullptr ? csName : L"") « std::endl;

      if ((csWKT != nullptr) && csWKT[0] != 0)
      {
        Console() « "    WKT: " « ToOsString(csWKT) « std::endl;
      }

      LgsxUtil_FreeMem((void**)&csName);
      LgsxUtil_FreeMem((void**)&csWKT);
    }

    Lgsx_ReaderEndCoordSystems(pReader);
  }
```

UCS lookup by name

- Description: Enumerates coordinate systems and resolves a specific UCS name with transform metadata using reader APIs.

```
  // Use JsReader APIs to get more info (WKT string)
  int numCs = 0;
  int hRes = Lgsx_ReaderEnumCoordSystems(pReader, &numCs);
  if (hRes == 0 && numCs > 0)
  {
    wchar_t* csName = nullptr;
    wchar_t* csWKT = nullptr;
    bool isActive;
    SCyRgdBdyTxf tmpTxf;
    while (0 == Lgsx_ReaderMoveNextCoordSystems2(pReader, &isActive, &csName, &csWKT, &tmpTxf))
    {
      auto name = ToOsString(csName != nullptr ? csName : L"");

      if (name == selectedUcsName)
      {
        Console() « "Found UCS: " « name « std::endl;
        Console() « "- translation: " « tmpTxf.mDx « "," « tmpTxf.mDy « "," « tmpTxf.mDz « std::endl;
        Console() « "- rotation: " « tmpTxf.mU « "," « tmpTxf.mV « "," « tmpTxf.mW « "," « tmpTxf.mS «
std::endl;

        if ((csWKT != nullptr) && csWKT[0] != 0)
        {
          Console() « "- WKT: " « ToOsString(csWKT) « std::endl;
        }

        Convert(txf, tmpTxf);

        LgsxUtil_FreeMem((void**)&csName);
        LgsxUtil_FreeMem((void**)&csWKT);

        return true;
      }

      LgsxUtil_FreeMem((void**)&csName);
      LgsxUtil_FreeMem((void**)&csWKT);
    }

    Lgsx_ReaderEndCoordSystems(pReader);
  }
```


Setup geotag traversal

- Description: Enumerates geotags and filters by setup anchor, then reads geotag identity and position through GeoTag handle APIs.

```
int numTags = 0;
int hRes = Lgsx_ReaderEnumGeoTags(pReader, &numTags);

if (hRes != 0 || numTags == 0)
{
  return;
}

int cntPrintGeotag = 0;
for (int i = 0; i < numTags; i++)
{
  GeoTagHandle geotag;
  if (0 != Lgsx_ReaderGetGeoTag(pReader, i, &geotag))
  {
    continue;
  }

  GeotagAnchorType anchorType;
  uint64_t anchorId;
  if (0 != Lgsx_GeoTagGetAnchor(geotag, &anchorType, &anchorId))
  {
    Lgsx_GeoTagRelease(&geotag);
    continue;
  }

  // Only print geotags for this specific setup
  if (anchorType != GeotagAnchorType_TlsSetup || anchorId != setupId)
  {
    Lgsx_GeoTagRelease(&geotag);
    continue;
  }

  if (cntPrintGeotag == 0)
  {
    Console() « "  - Geotags:" « std::endl;
  }

  cntPrintGeotag++;

  const char* geoTagGuid = nullptr;
  const wchar_t* name = nullptr;
  PNT3D position;

  if (0 == Lgsx_GeoTagGetGuid(geotag, &geoTagGuid) &&
      0 == Lgsx_GeoTagGetName(geotag, &name) &&
      0 == Lgsx_GeoTagGetPosition(geotag, &position))
  {
    PrintSingleGeotag(geotag, geoTagGuid, name, position, coordTxf,
                      setupCache, setupCacheCount, anchorId, "    ");
  }

  Lgsx_GeoTagRelease(&geotag);
}
```

Setup enumeration and name retrieval

- Description: Enumerates setups, reads setup identity/pose, and retrieves full setup names with current-setup APIs.

```
Console() « "Setups:" « std::endl;

// Single pass: cache all setups with their data
SetupCache* setupCache = nullptr;
int setupCacheCount = 0;

int numSetup = 0;
int hRes = Lgsx_ReaderEnumSetups(pReader, &numSetup);
if (hRes == 0 && numSetup > 0)
{
  setupCache = new SetupCache[numSetup];

  uint64_t setupId;
  CYSETUPINFO setup;
  LgsxMetadataPtr pMetadata;
  LgsxMetadataPtr* ppMetadata = &pMetadata;

  while (0 == Lgsx_ReaderMoveNextSetup(pReader, &setupId, &setup, ppMetadata))
```



```
  {
    setupCache[setupCacheCount].setupId = setupId;
    setupCache[setupCacheCount].setupIndex = setup.setupIndex;

    // Get the name using non-truncating API
    wchar_t* pSetupName = nullptr;
    Lgsx_ReaderGetCurrentSetupName(pReader, &pSetupName);
    setupCache[setupCacheCount].name = pSetupName != nullptr ? pSetupName : L"";
    LgsxUtil_FreeMem((void**)&pSetupName);

    // Store the raw pose
    setupCache[setupCacheCount].pose = setup.pose;

    // Convert and store the rigid transformation
    Convert(setupCache[setupCacheCount].rigidPose, setup.pose);

    setupCacheCount++;
  }

  Lgsx_ReaderEndSetups(pReader);
}
```

Point enumeration configuration

- Description: Configures and starts point enumeration with EnumPointsConfig APIs before handing off to chunk streaming.

```
int properties = Lgsx_ReaderQueryPropertyTypes(pReader);
if ((properties & PM_XYZ_DATA) != 0)
{
  EnumPointsConfigHandle config{};
  Lgsx_InitEnumPointsConfig(&config);
  Lgsx_EnumPointsConfigSetDesiredTypes(config, properties);
  Lgsx_EnumPointsConfigSetSubSample(config, 1);
  Lgsx_EnumPointsConfigSetVisible(config, true);
  Lgsx_EnumPointsConfigSetStrictOrder(config, true);
  int hRes = Lgsx_ReaderEnumPointsEx2(pReader, config);
  Lgsx_ReleaseEnumPointsConfig(&config);
  if (0 == hRes)
  {
    PrintPointCloudWithUcs(pReader, coordTxf, numPointsToPrint);
  }
}
else
{
  Console() « "Point cloud data not available in this project." « std::endl;
}
```

CLI and license bootstrap

- Description: Parses UCS-related CLI options and performs explicit SDK license initialization. Shows a complete standalone runtime setup for transformation tools.

```
Arguments args;

switch(ParseArguments(argc, argv, args))
{
  case ParseArgumentsResult::SUCCESS:
    break;
  case ParseArgumentsResult::STOP:
    return 0;
  case ParseArgumentsResult::FAIL:
  default:
    return 1;
}

  Console() « "LGSx UCS example started. Logging is " « (args.m_enableLogging ? "enabled." :
"disabled.") « std::endl;

  std::wstring license = GetSdkLicense(args.m_licenseFileName);
  if (license.empty())
  {
    Console() « "Cannot prepare license." « std::endl;
    return 1;
  }

  int hRes = Lgsx_Initialize(L"", args.m_enableLogging);
  if (hRes != 0)
  {
    Console() « "LGSx SDK initialization failed." « std::endl;
```



```
    return 1;
  }
  Console() « "Lgsx_InitalizeEx called." « std::endl;

  // see License.cpp for expiration date of LgsxLicenseKey license string
  hRes = Lgsx_InitLicenseEx(license.c_str(), L"", L"LgsxUcsExample");
  Console() « "Lgsx_InitLicenseEx called." « std::endl;
  if (hRes != 0)
  {
    Console() « "Bad SDK license key or license expired." « std::endl;
    Lgsx_Uninitialize();
    return 1;
  }

  try
  {
    hRes = LgsxUcsExample(args.m_inputFileName, args.m_password, args.m_ucsName, args.m_processAllUcs,
args.m_numPointsToPrint);
  }
  catch (const std::exception& e)
  {
    Console() « "**Exception caught in LGS reader test: " « e.what() « std::endl;
    hRes = 1;
  }
  catch (...)
  {
    Console() « "**Unknown exception caught in LGSx UCS test." « std::endl;
    hRes = 1;
  }

  Lgsx_Uninitialize();
  Console() « std::endl « "Finished LGSx UCS test." « std::endl;
```

UCS reporting workflow

- Description: Runs UCS-dependent coordinate conversion and object reporting. Shows how to align project data to user-defined coordinate systems.

```
  const auto wideInFile = ToWideString(inFile.c_str());
  const wchar_t* lgsFile = wideInFile.c_str();

  if (FileExists(inFile) == 0)
  {
    Console() « std::endl « "=======Error: File not found: " « ToOsString(lgsFile) « " ========" «
std::endl;
    return 1;
  }

  // Make sure that if the file requires a password, that a password was provided.
  bool filePasswordProtected = false;
  const int result = Lgsx_ReaderHasPassword(lgsFile, &filePasswordProtected);
  if (result != 0)
  {
    Console() « "=======Error: " « ToOsString(lgsFile) « " check if password is required failed." «
std::endl;
    return 1;
  }
  if (filePasswordProtected && password.empty())
  {
    Console() « "=======Error: " « ToOsString(lgsFile) « " requires a password, but none was
provided." « std::endl;
    return 1;
  }

  int status = 0;
  Console() « std::endl « "=======Begin " « ToOsString(lgsFile) « " ========" « std::endl;
  LgsxReaderPtr reader = Lgsx_ReaderCreate(&status);
  status = Lgsx_ReaderOpen(reader, lgsFile, ToStdString(password.c_str()).c_str());
  if (status != 0)
  {
    char* errorMsg = nullptr;
    Lgsx_ReaderGetLastError2(reader, &errorMsg);
    Console() « "Error opening " « ToOsString(lgsFile) « ": " « status « std::endl;
    Console() « "Details: " « (errorMsg != nullptr ? errorMsg : "Unknown error") « std::endl;
    LgsxUtil_FreeMem((void**)&errorMsg);
    Lgsx_FreeHandle(reader);
    return 1;
  }

  Console() « "Opened " « ToOsString(lgsFile) « " - successful" « std::endl;

  int returnValue = 0;

  try
```



```
{
  if (processAllUcs)
  {
    PrintObjectsPositionsForAllUcs(reader, numPointsToPrint);
  }
  else if (ucsName.empty())
  {
    PrintUcsList(reader);
  }
  else
  {
    PrintObjectsPositions(reader, ucsName, numPointsToPrint);
  }
}
catch (const std::exception& e)
{
  Console() « "!!!Caught unhandled exception: " « e.what() « std::endl;
  returnValue = 1;
}
catch (...)
{
  Console() « "!!!Caught unhandled exception." « std::endl;
  returnValue = 1;
}

// Make sure the reader is closed so the related document is closed
status = Lgsx_ReaderClose(reader);
Console() « std::endl « "»Closed" « std::endl « std::endl;

// You need to free reader handle in time - leaving it around may cause problem later
Lgsx_FreeHandle(reader);
```

1.3.3.3.2 LgsxSetupIndexExample

Setup-index clamped mapping

- Description: Reverses setup-index encoding with the trajectory scale and start index, then clamps the resolved vertex index to the valid [0, vertexCount-1] range. Use this when PM_SETUPIDX values fall between sampled trajectory vertices and you need a deterministic attribution target.

```
int ClampSetupIndexToVertex(
  int setupIndex,
  const RunInfo& run)
{
  // Apply reverse mapping formula
  double rawVertexIndex = static_cast<double>(setupIndex - run.startSetupIndex) / run.scaleFactor;
  int vertexIndex = static_cast<int>(rawVertexIndex);

  // Clamp to valid vertex range [0, vertexCount-1]
  if (vertexIndex >= run.vertexCount)
  {
    vertexIndex = run.vertexCount - 1;
  }
  else if (vertexIndex < 0)
  {
    vertexIndex = 0;
  }

  return vertexIndex;
```

PM_SETUPIDX field streaming

- Description: Configures point-field enumeration and streams PM_SETUPIDX chunks. Shows how to read setup-index values from point blocks and attribute them incrementally.

```
  // Allocate buffers for point data
  Point3D* points = new Point3D[CHUNK_SIZE];
  uint32_t* setupidx = new uint32_t[CHUNK_SIZE];

  // Set up ppData array for Lgsx_ReaderMoveNextFields
  void* ppData[16];
  std::memset(ppData, 0, sizeof(ppData));
  ppData[0] = points;     // PM_XYZ_DATA at index 0
  ppData[5] = setupidx;   // PM_SETUPIDX at index 5

  // Convert percentage to subsample rate
```



```
  int subsampleRate = (subsamplePercent > 0.0) ? static_cast<int>(1.0 / subsamplePercent) : 1;
  if (subsampleRate < 1) subsampleRate = 1;

  // Start point enumeration
  int desiredTypes = PM_XYZ_DATA | PM_SETUPIDX;
  EnumPointsConfigHandle config{};
  Lgsx_InitEnumPointsConfig(&config);
  Lgsx_EnumPointsConfigSetDesiredTypes(config, desiredTypes);
  Lgsx_EnumPointsConfigSetSubSample(config, subsampleRate);
  Lgsx_EnumPointsConfigSetVisible(config, true);
  int hRes = Lgsx_ReaderEnumPointsEx2(reader, config);
  Lgsx_ReleaseEnumPointsConfig(&config);
  if (hRes != 0)
  {
    Console() << "Error starting point enumeration: " << hRes << std::endl;
    delete[] points;
    delete[] setupidx;
    return 1;
  }

  Console() << std::endl << "Reading points (" << (subsamplePercent * 100) << "% of points)..." <<
std::endl;

  totalPoints = 0;
  unknownSetupPoints = 0;

  while (true)
  {
    int fill = Lgsx_ReaderMoveNextFields(reader, CHUNK_SIZE, BitmapFormat__RGBA, ppData);
    if (fill == 0)
    {
      break;
    }

    // Attribute each point
    for (int i = 0; i < fill; ++i)
    {
      int idx = static_cast<int>(setupidx[i]);

      if (!AttributePoint(idx, tlsSetupMap, runMap))
      {
        unknownSetupPoints++;
      }
    }

    totalPoints += fill;

    // Progress indicator every 1 million points
    if (totalPoints % 1000000 < static_cast<uint64_t>(fill))
    {
      Console() << "  Processed " << totalPoints << " points..." << std::endl;
    }
  }

  delete[] points;
  delete[] setupidx;
```

Setup-index attribution workflow

- Description: Computes per-setup point ownership from PM_SETUPIDX point attributes. Shows the end-to-end orchestration that combines setup/run maps with point streaming.

```
  const auto wideInFile = ToWideString(inFile.c_str());
  const wchar_t* lgsFile = wideInFile.c_str();

  if (FileExists(inFile) == 0)
  {
    Console() << std::endl << "=======Error: File not found: " << ToOsString(lgsFile) << " ========" <<
std::endl;
    return 1;
  }

  // Check if file requires a password
  bool filePasswordProtected = false;
  const int result = Lgsx_ReaderHasPassword(lgsFile, &filePasswordProtected);
  if (result != 0)
  {
    Console() << "=======Error: " << ToOsString(lgsFile) << " check if password is required failed." <<
std::endl;
    return 1;
  }
  if (filePasswordProtected && password.empty())
  {
```



```
    Console() « "=======Error: " « ToOsString(lgsFile) « " requires a password, but none was
provided." « std::endl;
    return 1;
  }

  int status = 0;
  Console() « std::endl « "=======Begin " « ToOsString(lgsFile) « " ========" « std::endl;
  LgsxReaderPtr reader = Lgsx_ReaderCreate(&status);
  status = Lgsx_ReaderOpen(reader, lgsFile, ToStdString(password.c_str()).c_str());
  if (status != 0)
  {
    char* errorMsg = nullptr;
    Lgsx_ReaderGetLastError2(reader, &errorMsg);
    Console() « "Error opening " « ToOsString(lgsFile) « ": " « status « std::endl;
    Console() « "Details: " « (errorMsg != nullptr ? errorMsg : "Unknown error") « std::endl;
    LgsxUtil_FreeMem((void**)&errorMsg);
    Lgsx_FreeHandle(reader);
    return 1;
  }

  Console() « "Opened " « ToOsString(lgsFile) « " - successful" « std::endl;

  int returnValue = 0;

  try
  {
    // Build setup and run lookup maps
    std::map<int, SetupInfo> tlsSetupMap;
    std::map<uint64_t, RunInfo> runMap;
    BuildSetupAndRunMaps(reader, tlsSetupMap, runMap);

    if (tlsSetupMap.empty() && runMap.empty())
    {
      Console() « "No setups or runs found. Cannot determine point-to-setup mapping." « std::endl;
      returnValue = 0;
    }
    else
    {
      // Count points per setup
      returnValue = CountPointsPerSetup(reader, tlsSetupMap, runMap, subsamplePercent);
    }
  }
  catch (const std::exception& e)
  {
    Console() « "!!!Caught unhandled exception: " « e.what() « std::endl;
    returnValue = 1;
  }
  catch (...)
  {
    Console() « "!!!Caught unhandled exception." « std::endl;
    returnValue = 1;
  }

  // Close the reader
  status = Lgsx_ReaderClose(reader);
  Console() « std::endl « "»Closed" « std::endl « std::endl;

  Lgsx_FreeHandle(reader);
```

1.3.3.3.3 LgsxSensorProcessingInfoExample

SensorInfo API readout

- Description: Retrieves core SensorInfo fields and the optional device-info snapshot handle.

```
  const char* guid;
  if (0 != Lgsx_SensorInfoGetGuid(info, &guid))
  {
    Console() « "    Error reading sensor info guid" « std::endl;
    return;
  }

  const wchar_t* serial;
  if (0 != Lgsx_SensorInfoGetSerialNumber(info, &serial))
  {
    Console() « "    Error reading sensor info serial number" « std::endl;
    return;
  }

  const wchar_t* scannerType;
```



```
if (0 != Lgsx_SensorInfoGetScannerType(info, &scannerType))
{
  Console() « "    Error reading sensor info scanner type" « std::endl;
  return;
}

const wchar_t* hardwareVersion;
if (0 != Lgsx_SensorInfoGetHardwareVersion(info, &hardwareVersion))
{
  Console() « "    Error reading sensor info hardware version" « std::endl;
  return;
}

const wchar_t* firmwareVersion;
if (0 != Lgsx_SensorInfoGetFirmwareVersion(info, &firmwareVersion))
{
  Console() « "    Error reading sensor info firmware version" « std::endl;
  return;
}

const wchar_t* manufacturer;
if (0 != Lgsx_SensorInfoGetManufacturer(info, &manufacturer))
{
  Console() « "    Error reading sensor info manufacturer" « std::endl;
  return;
}

const wchar_t* articleNumber;
if (0 != Lgsx_SensorInfoGetArticleNumber(info, &articleNumber))
{
  Console() « "    Error reading sensor info article number" « std::endl;
  return;
}

uint64_t timestamp = 0;
if (0 != Lgsx_SensorInfoGetDeviceInfoCaptureTime(info, &timestamp))
{
  Console() « "    Error reading sensor info capture timestamp" « std::endl;
  return;
}

Console() « "    SensorInfo " « guid « std::endl;
Console() « "      Scanner type:     [" « ToOsString(scannerType) « "]" « std::endl;
Console() « "      Serial number:    [" « ToOsString(serial) « "]" « std::endl;
Console() « "      Firmware version: [" « ToOsString(firmwareVersion) « "]" « std::endl;
Console() « "      Hardware version: [" « ToOsString(hardwareVersion) « "]" « std::endl;
Console() « "      Manufacturer:     [" « ToOsString(manufacturer) « "]" « std::endl;
Console() « "      Article number:   [" « ToOsString(articleNumber) « "]" « std::endl;
Console() « "      Capture time:     [" « FormatTimestamp(timestamp) « "]" « std::endl;

LgsxMetadataPtr snapshot = nullptr;
Lgsx_SensorInfoGetDeviceInfoSnapshot(info, &snapshot);
```

ProcessingInfo API readout

- Description: Retrieves processing provenance fields, optional library version, and the pipeline metadata handle.

```
if (0 != Lgsx_ProcessingInfoGetGuid(info, &guid))
{
  Console() « "    Error reading ProcessingInfo guid for #" « index « std::endl;
  return;
}
if (0 != Lgsx_ProcessingInfoGetProcessingTimestamp(info, &processingTimestamp))
{
  Console() « "    Error reading ProcessingInfo timestamp for #" « index « std::endl;
  return;
}
if (0 != Lgsx_ProcessingInfoGetAppName(info, &appName))
{
  Console() « "    Error reading ProcessingInfo app name for #" « index « std::endl;
  return;
}
if (0 != Lgsx_ProcessingInfoGetAppVersion(info, &appVersion))
{
  Console() « "    Error reading ProcessingInfo app version for #" « index « std::endl;
  return;
}
if (0 != Lgsx_ProcessingInfoHasLibraryVersion(info, &hasLibraryVersion))
{
  Console() « "    Error reading ProcessingInfo library version flag for #" « index « std::endl;
  return;
}
```



```
if (hasLibraryVersion && 0 != Lgsx_ProcessingInfoGetLibraryVersion(info, &libraryVersion))
{
  Console() « "    Error reading ProcessingInfo library version for #" « index « std::endl;
  return;
}
if (0 != Lgsx_ProcessingInfoGetOrderIndex(info, &orderIndex))
{
  Console() « "    Error reading ProcessingInfo order index for #" « index « std::endl;
  return;
}
if (0 != Lgsx_ProcessingInfoGetPipelineInfo(info, &pipelineInfo))
{
  Console() « "    Error reading ProcessingInfo pipeline info for #" « index « std::endl;
  return;
}
```

Sensor and processing info traversal

- Description: Shows the high-level setup/run traversal pattern that invokes SensorInfo and ProcessingInfo readers.

```
while (0 == Lgsx_ReaderMoveNextSetup(reader, &setupId, &setup, nullptr))
{
  Console() « std::endl « "  Setup " « setup.setupIndex
    « " [" « ToOsString(setup.name) « "] id=" « setupId « std::endl;

  PrintSetupSensorInfo(reader);
  PrintSetupProcessingInfos(reader);

  int numScans = 0;
  hRes = Lgsx_ReaderEnumSetupScans(reader, &numScans);
  if (hRes == 0 && numScans > 0)
  {
    Console() « "    Scan count = " « numScans « std::endl;
    for (int i = 0; i < numScans; i++)
    {
      ScanHandle scan{};
      if (0 != Lgsx_ReaderGetSetupScan(reader, i, &scan))
      {
        Console() « "!!! Error reading scan #" « i « std::endl;
        continue;
      }
      const char* scanGuid = nullptr;
      Lgsx_ScanGetGuid(scan, &scanGuid);

      int scanType = 0;
      Lgsx_ScanGetScanType(scan, &scanType);

      int scanRole = 0;
      Lgsx_ScanGetScanRole(scan, &scanRole);

      uint64_t pointCount = 0;
      Lgsx_ScanGetPointCount(scan, &pointCount);

      Console() « "      Scan #" « i « " " « (scanGuid != nullptr ? scanGuid : "?");
      Console() « " type=" « scanType « " role=" « scanRole;
      Console() « " points=" « pointCount;

      bool hasWindow = false;
      Lgsx_ScanHasWindow(scan, &hasWindow);
      if (hasWindow)
      {
        double minAz = 0, minEl = 0, maxAz = 0, maxEl = 0;
        Lgsx_ScanGetWindowMinAzimuth(scan, &minAz);
        Lgsx_ScanGetWindowMinElevation(scan, &minEl);
        Lgsx_ScanGetWindowMaxAzimuth(scan, &maxAz);
        Lgsx_ScanGetWindowMaxElevation(scan, &maxEl);
        Console() « " window=[" « minAz « "," « minEl « " .. " « maxAz « "," « maxEl « "]";
      }

      uint64_t captureTs = 0;
      Lgsx_ScanGetCaptureTimestamp(scan, &captureTs);
      double density = 0;
      Lgsx_ScanGetScanDensity(scan, &density);
      Console() « " captureTs=" « captureTs « " density=" « density;
      Console() « std::endl;

      Lgsx_ScanRelease(&scan);
    }
  }
}
Lgsx_ReaderEndSetups(reader);
}
```



```
  else
  {
    Console() « std::endl « "No TLS setups found." « std::endl;
    Lgsx_ReaderEndSetups(reader);
  }

  // --- Runs (mobile) ---
  int numRun = 0;
  hRes = Lgsx_ReaderEnumRuns(reader, &numRun);
  if (hRes == 0 && numRun > 0)
  {
    Console() « std::endl « "=== Runs (" « numRun « ") ===" « std::endl;
    uint64_t trajId = 0;
    int nVertex;
    CYTRAJECTORYINFO trajectory;
    CYTRAJECTORYVERTEX* trajectoryPath;
    while (0 == Lgsx_ReaderMoveNextRun(reader, &trajId, &trajectory, &nVertex, &trajectoryPath,
nullptr))
    {
      Console() « std::endl « "  Run " « trajId
        « " [" « ToOsString(trajectory.runName) « "]" « std::endl;

      Console() « "  -- Run-level SensorInfo/ProcessingInfo --" « std::endl;
      PrintRunSensorInfo(reader);
      PrintRunProcessingInfos(reader);
    }
```

1.3.3.4 Extraction Utilities

1.3.3.4.1 LgsxExtractCubemapsExample

TLS cubemap API traversal

- Description: Enumerates setups and reads cubemap layers, levels, and face bytes. Shows direct use of cubemap read APIs for setup-scoped image export.

```
  // Print out all setups with targets
  int numSetup = 0;
  int hRes = Lgsx_ReaderEnumSetups(reader, &numSetup);
  if (hRes == 0 && numSetup > 0)
  {
    Console() « "TLS Setup count = " « numSetup « std::endl;

    const int maxPrintImage = 2;
    uint64_t setupId;
    CYSETUPINFO setup;
    while (0 == Lgsx_ReaderMoveNextSetup(reader, &setupId, &setup, nullptr))
    {
      wchar_t* pSetupName = nullptr;
      Lgsx_ReaderGetCurrentSetupName(reader, &pSetupName);
      Console() « "Setup ID: " « setupId « ", Name: " « ToOsString(pSetupName != nullptr ? pSetupName
: L"") « std::endl;
      LgsxUtil_FreeMem((void**)&pSetupName);

      std::filesystem::path setupDirectory = setupsDirectory / std::to_string(setupId);

      wchar_t layers[128];
      int numLayers = Lgsx_ReaderGetImageLayerNames(reader, layers, 128);
      if (numLayers > 0)
      {
        Console() « "- Image Layers (" « numLayers « "): " « ToOsString(layers) « std::endl;

        wchar_t* pLayerTokPtr;
        for (const wchar_t* layer = wcstok(layers, L",", &pLayerTokPtr); layer != nullptr; layer =
wcstok(nullptr, L",", &pLayerTokPtr))
        {
          Console() « "- Layer: " « ToOsString(layer) « std::endl;

          if (!layerName.empty() && ToOsString(layer) != layerName)
          {
            Console() « "  - Skipping layer (not matching requested layer name)." « std::endl;
            continue;
          }

          std::filesystem::path layerDirectory = setupDirectory / ToStdString(layer);

          wchar_t* pImageType;
          hRes = Lgsx_ReaderGetCubemapImageType(reader, layer, &pImageType);
          if (hRes == 0)
```



```
          {
            Console() « "- Cubemap image type: " « ToOsString(pImageType) « std::endl;

            int maxAvailableLevel = 0;
            hRes = Lgsx_ReaderGetCubemapMaxLevel(reader, layer, &maxAvailableLevel);
            if (hRes == 0)
            {
              Console() « "- Cubemap available levels: " « maxAvailableLevel « std::endl;
              if (maxLevel >= 0 && maxAvailableLevel > maxLevel)
              {
                maxAvailableLevel = maxLevel;
                Console() « "- Limiting to max level: " « maxLevel « std::endl;
              }

              for (int level = 0; level <= maxAvailableLevel; ++level)
              {
                int size[6] = {};
                unsigned char* images[6] = {};

                hRes = Lgsx_ReaderGetCubemapImageBytes(reader, layer, level, &size[0], &images[0]);
                if (hRes == 0)
                {
                  Console() « "-- Level " « level « ", image type = " « ToOsString(pImageType) «
std::endl;
                  for (int face = 0; face < 6; ++face)
                  {
                    if (images[face] != nullptr)
                    {
                      static const char* pFaceNames[6] = { "PosY", "NegY", "NegX", "PosX", "PosZ",
"NegZ" };

                      Console() « "--- Face " « pFaceNames[face] « ": image data pointer = "
                        « static_cast<void*>(images[face]) « ", size = " « size[face] « std::endl;

                      // create output directory if not exists
                      std::filesystem::create_directories(layerDirectory);

                      std::string faceFileName = std::to_string(level) + "_" + pFaceNames[face] + "."
+ ToStdString(pImageType);
                      std::filesystem::path faceFilePath = layerDirectory / faceFileName;

                      Console() « "---- Writing face image to file: " «
ToOsString(faceFileName.c_str()) « std::endl;

                      FILE* pFaceFile = fopen(faceFilePath.string().c_str(), "wb");
                      if (pFaceFile != nullptr)
                      {
                        const char* pImageData = reinterpret_cast<const char*>(images[face]);
                        size_t bytesLeft = static_cast<size_t>(size[face]);
                        while (bytesLeft > 0)
                        {
                          size_t bytesWritten = fwrite(pImageData, 1, bytesLeft, pFaceFile);
                          if (bytesWritten <= 0)
                          {
                            Console() « "---- Failed to write image data to file: " «
ToOsString(faceFileName.c_str()) « " - returned " « bytesWritten « std::endl;
                            break;
                          }
                          bytesLeft -= bytesWritten;
                          pImageData += bytesWritten;
                        }
                        fclose(pFaceFile);
                        Console() « "---- Saved face image to file: " «
ToOsString(faceFileName.c_str()) « std::endl;
                      }
                    }
                  }
                }
              }
            }

            LgsxUtil_FreeMem((void**)&pImageType);
          }
        }
      }
    }

    Lgsx_ReaderEndSetups(reader);
  }
```

Kinematic run cubemap traversal

- Description: Enumerates runs and reuses setup extraction for each trajectory scope. Shows direct run iteration before setup-level cubemap extraction.


```
  int numRun = 0;
  int hRes = Lgsx_ReaderEnumRuns(reader, &numRun);
  if (hRes == 0 && numRun > 0)
  {
    Console() « "Run count " « numRun « std::endl;

    uint64_t trajId = 0;
    int nVertex;
    CYTRAJECTORYINFO trajectory;
    CYTRAJECTORYVERTEX* trajectoryPath;
    LgsxMetadataPtr pMetadata = nullptr;
    wchar_t* pRunName = nullptr;
    while (0 == Lgsx_ReaderMoveNextRun2(reader, &trajId, &trajectory, &nVertex, &trajectoryPath,
nullptr, nullptr, &pRunName))
    {
      Console() « "Trajectory ID: " « trajId « ", Name: " « ToOsString(pRunName != nullptr ? pRunName
: L"") « std::endl;

      std::filesystem::path trajDirectory = tracksDirectory / std::to_string(trajId);
      ExtractSetupsImages(reader, trajDirectory, layerName, maxLevel);
      LgsxUtil_FreeMem((void**)&pRunName);
    }
    Lgsx_ReaderEndRuns(reader);
  }
```

Cubemap extraction loop

- Description: Opens the reader and orchestrates TLS plus run-based cubemap extraction. Shows end-to-end extraction flow and error-safe lifecycle handling.

```
  const auto wideInFile = ToWideString(inFile.c_str());
  const wchar_t* lgsFile = wideInFile.c_str();

  Console() « std::endl « "=======Begin " « ToOsString(lgsFile) « " ========" « std::endl;
  LgsxReaderPtr reader = nullptr;
  const int openResult = OpenReaderChecked(inFile, password, &reader);
  if (openResult != 0)
  {
    return openResult;
  }

  int returnValue = 0;

  try
  {
    std::filesystem::path outputDirPath = std::filesystem::path(ToStdString(outputDirectory.c_str()));

    ExtractTlsSetupsImages(reader, outputDirPath, layerName, maxLevel);
    ExtractKinematicSetupsImages(reader, outputDirPath, layerName, maxLevel);
  }
  catch (const std::exception& e)
  {
    Console() « "!!!Caught unhandled exception: " « e.what() « std::endl;
    returnValue = 1;
  }
  catch (...)
  {
    Console() « "!!!Caught unhandled exception." « std::endl;
    returnValue = 1;
  }

  // Make sure the reader is closed so the related document is closed
  CloseReaderChecked(&reader);
  Console() « std::endl « "»Closed" « std::endl « std::endl;
```

1.3.3.4.2 LgsxExtractSitemapsExample

Sitemap extraction loop

- Description: Walks each sitemap and writes metadata plus image artifacts. Shows how to build a complete sitemap export workflow for downstream visualization.

```
  LgsxReaderPtr reader = nullptr;
  const int openResult = OpenReaderChecked(inputFile, password, &reader);
  if (openResult != 0)
  {
```



```
    return openResult;
  }

  const auto outputRoot = std::filesystem::path(ToStdString(outputDirectory.c_str()));
  std::filesystem::create_directories(outputRoot);

  int sitemapCount = 0;
  if (Lgsx_ReaderEnumSitemaps(reader, &sitemapCount) != 0)
  {
    Console() « "Failed to enumerate sitemaps." « std::endl;
    CloseReaderChecked(&reader);
    return 1;
  }

  Console() « "Sitemap count = " « sitemapCount « std::endl;

  int returnValue = 0;
  for (int sitemapIndex = 0; sitemapIndex < sitemapCount; ++sitemapIndex)
  {
    SitemapHandle sitemapHandle{};
    if (Lgsx_ReaderGetSitemapHandle(reader, sitemapIndex, &sitemapHandle) != 0)
    {
      returnValue = 1;
      continue;
    }

    uint64_t sitemapId = 0;
    int orderingIndex = -1;
    bool isMaster = false;
    const char* sitemapGuid = nullptr;
    const wchar_t* sitemapName = nullptr;
    uint64_t activeCoordSystemId = 0;

    Lgsx_SitemapGetId(sitemapHandle, &sitemapId);
    Lgsx_SitemapGetGuid(sitemapHandle, &sitemapGuid);
    Lgsx_SitemapGetName(sitemapHandle, &sitemapName);
    Lgsx_SitemapGetOrderingIndex(sitemapHandle, &orderingIndex);
    Lgsx_SitemapGetIsMaster(sitemapHandle, &isMaster);
    Lgsx_SitemapGetActiveCoordSystemId(sitemapHandle, &activeCoordSystemId);

    const std::string guidSegment = (sitemapGuid != nullptr && sitemapGuid[0] != '\0')
      ? std::string(sitemapGuid)
      : ("numeric_" + std::to_string(sitemapId));
    const auto sitemapDirectory = outputRoot / guidSegment;
    std::filesystem::create_directories(sitemapDirectory);

    std::string metadata;
    metadata += "id=" + std::to_string(sitemapId) + "\n";
    metadata += "guid=" + std::string(sitemapGuid != nullptr ? sitemapGuid : "") + "\n";
    metadata += "name=" + ToStdString(sitemapName != nullptr ? sitemapName : L"") + "\n";
    metadata += "orderingIndex=" + std::to_string(orderingIndex) + "\n";
    metadata += std::string("isMaster=") + (isMaster ? "true" : "false") + "\n";
    metadata += "activeCoordSystemId=" + std::to_string(activeCoordSystemId) + "\n";

    int itemCount = 0;
    if (Lgsx_SitemapEnumItems(sitemapHandle, &itemCount) == 0)
    {
      metadata += "itemCount=" + std::to_string(itemCount) + "\n";
      for (int itemIndex = 0; itemIndex < itemCount; ++itemIndex)
      {
        SitemapItemHandle itemHandle{};
        if (Lgsx_SitemapGetItem(sitemapHandle, itemIndex, &itemHandle) != 0)
        {
          continue;
        }

        uint64_t itemId = 0;
        SitemapItemType itemType = SitemapItemType_TlsSetup;
        const char* itemGuid = nullptr;
        SCyRgdBdyTxf pose{};
        Lgsx_SitemapItemGetId(itemHandle, &itemId);
        Lgsx_SitemapItemGetType(itemHandle, &itemType);
        Lgsx_SitemapItemGetGuid(itemHandle, &itemGuid);
        Lgsx_SitemapItemGetPose(itemHandle, &pose);

        metadata += "item." + std::to_string(itemIndex) + ".id=" + std::to_string(itemId) + "\n";
        metadata += "item." + std::to_string(itemIndex) + ".guid=" + std::string(itemGuid != nullptr ?
itemGuid : "") + "\n";
        metadata += "item." + std::to_string(itemIndex) + ".type=" + SitemapItemTypeName(itemType) +
"\n";
        metadata += "item." + std::to_string(itemIndex) + ".pose=" +
          std::to_string(pose.mDx) + "," + std::to_string(pose.mDy) + "," + std::to_string(pose.mDz) +
"," +
          std::to_string(pose.mU) + "," + std::to_string(pose.mV) + "," + std::to_string(pose.mW) +
"," + std::to_string(pose.mS) + "\n";
      }
```



```
    }

    if (!WriteTextFile(sitemapDirectory / "metadata.txt", metadata))
    {
      Lgsx_SitemapRelease(&sitemapHandle);
      throw std::runtime_error("Failed to write sitemap metadata file.");
    }

    Console() « "Extracting sitemap [" « ToOsString(sitemapGuid != nullptr ? sitemapGuid : "") « "] to
"
      « ToOsString(sitemapDirectory.string().c_str()) « std::endl;

    ExtractSitemapImage(reader, sitemapHandle, SitemapImageType_Background, sitemapDirectory);
    ExtractSitemapImage(reader, sitemapHandle, SitemapImageType_Overview, sitemapDirectory);
    ExtractSitemapImage(reader, sitemapHandle, SitemapImageType_Acceptance, sitemapDirectory);

    ExtractSitemapImageBinary(reader, sitemapHandle, SitemapImageType_Background, sitemapDirectory);
    ExtractSitemapImageBinary(reader, sitemapHandle, SitemapImageType_Overview, sitemapDirectory);
    ExtractSitemapImageBinary(reader, sitemapHandle, SitemapImageType_Acceptance, sitemapDirectory);

    Lgsx_SitemapRelease(&sitemapHandle);
  }

  Lgsx_ReaderEndSitemaps(reader);
  CloseReaderChecked(&reader);
```

1.3.3.4.3 LgsxExtractPointCloudExample

Point-cloud extraction loop

- Description: Validates file type, opens reader, and runs selected extraction path. Shows the two supported extraction APIs and when to choose each one.

```
  Console() « "LGS extractor example started" « std::endl;

  const auto wideInFile = ToWideString(inFile.c_str());
  const wchar_t* lgsFile = wideInFile.c_str();

  size_t len = wcslen(lgsFile);
  if (len >= 5 && wcsicmp(lgsFile + len - 5, L".lgsx") == 0)
  {
    Console() « "LGSx file detected" « std::endl;
  }
  else if (len >= 4 && wcsicmp(lgsFile + len - 4, L".lgs") == 0)
  {
    Console() « "LGS file detected" « std::endl;
  }
  else
  {
    Console() « std::endl « "====== Error: File type is not supported:" « ToOsString(lgsFile) « "
=======" « std::endl;
    return 1;
  }

  Console() « std::endl « "======= Begin " « ToOsString(lgsFile) « " =======" « std::endl;

  LgsxReaderPtr reader = nullptr;
  const int openResult = OpenReaderChecked(inFile, password, &reader);
  if (openResult != 0)
  {
    return openResult;
  }

  if (useFileApi)
  {
    Console() « "Extracting point cloud using file API" « std::endl;
    ExtractWithFileApi(reader, outFile);
  }
  else
  {
    Console() « "Extracting point cloud using reader API" « std::endl;
    ExtractWithReader(reader, outFile);
  }

  CloseReaderChecked(&reader);
  Console() « std::endl « "» Closed" « std::endl « std::endl;
```


Chapter 2

Deprecated List

Struct CYGEOTAG

26.06.2025 - New GeoTag API available via GeoTagHandle

Global Lgsx_MetaMoveNext (LgsxMetadataPtr pMetadata, LgsxPropertyName pName, int *pType)

09.07.2025 - Lgsx_MetaMoveNext2 introduced that does not truncate the name.

Global Lgsx_ReaderGetGeoTagDescription (LgsxReaderPtr reader, uint64_t geoTagId, wchar_t **pDescPtr)

26.06.2025 - New GeoTag API available via GeoTagHandle

Global Lgsx_ReaderGetGeoTagName (LgsxReaderPtr reader, uint64_t geoTagId, wchar_t **pNamePtr)

26.06.2025 - New GeoTag API available via GeoTagHandle

Global Lgsx_ReaderHasGeoTagDescription (LgsxReaderPtr reader, uint64_t geoTagId, bool *pStatus)

26.06.2025 - New GeoTag API available via GeoTagHandle

Global Lgsx_ReaderMoveNextGeoTag (LgsxReaderPtr reader, uint64_t *geoTagId, CYGEOTAG *pGeoTag, LgsxMetadataPtr *pMetadata)

26.06.2025 - New GeoTag API available via GeoTagHandle

Global LgsxUtil_A2W (const char *value, wchar_t *tValue)

Legacy function without buffer size limit. Use LgsxUtil_A2WEx() instead.

Global LgsxUtil_W2A (const wchar_t *value, char *sValue)

Legacy function without buffer size limit. Use LgsxUtil_W2AEx() instead.

Chapter 3

Topic Index

3.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Topic Documentation

6.1 Reader Functions

Topics

- Basic Reader Functions
- Project Functions
- Asset Functions
- Categories Functions

  Categories.
- Fields Functions

  Fields.
- Migrated Fields

  Old Geotag's metadata "Category", "Label" and "TagIndex" are now automatically migrated to new Fields.
- Setup Functions
- SensorInfo Functions

  SensorInfo.
- Target Functions
- Run Functions
- GeoTag Functions
- Sitemap Functions
- Model and Model Node Functions
- Point Cloud Functions
- User Coordinate System Functions

Data Structures

- struct AssetHandle

  Asset data handle for accessing Asset data without buffer truncation.
- struct CategoryHandle

  Descriptor for a Category object.
- struct CategoryValueHandle

  Descriptor for a Category Value object.
- struct CategoryEntryHandle

  Descriptor for a Category Entry object.
- struct FieldHandle

Descriptor for a Field object.

- struct FieldEntryHandle

  Descriptor for a Field Entry object.
- struct SensorInfoHandle

  SensorInfo data handle for accessing the SensorInfo data.
- struct ProcessingInfoHandle

  ProcessingInfo data handle for accessing setup ProcessingInfo data.
- struct ScanHandle

  Scan data handle for accessing the Scan data.
- struct GeoTagHandle

  GeoTag data handle for accessing the GeoTag data.
- struct SitemapHandle

  Sitemap data handle for accessing Sitemap data without reader-level cursor state.
- struct SitemapItemHandle

  Sitemap item handle bound to the lifetime of its parent SitemapHandle.
- struct EnumPointsConfigHandle

  Configuration handle for Lgsx_ReaderEnumPointsEx2().
- struct PointCloudFileHandle

  Point Cloud handle for accessing the raw point cloud data.

Typedefs

- typedef void * **LgsxReaderPtr**

  Handle for LGSx reader object.

Sitemap corner correction flags

Bitmask flags for the `pFlags` parameter of Lgsx_SitemapImageGetCorners().

- #define **LGSX_SITEMAP_CORNERS_CALCULATE_MISSING** 0x01

  Reconstruct corners from setup/track pixel positions when corner data is absent.
- #define LGSX_SITEMAP_CORNERS_FIX_INCONSISTENT 0x02

  Detect and replace corners that are inconsistent with the recorded setup pixel positions.
- #define LGSX_SITEMAP_CORNERS_REJECT_DEGENERATE 0x04

  Treat geometrically degenerate stored corners as missing.

6.1.1 Detailed Description

6.1.2 Macro Definition Documentation

6.1.2.1 LGSX_SITEMAP_CORNERS_FIX_INCONSISTENT

```
#define LGSX_SITEMAP_CORNERS_FIX_INCONSISTENT 0x02
```

Detect and replace corners that are inconsistent with the recorded setup pixel positions.

Uses a majority-vote outlier test: if more than half of the setup correspondences have a predicted pixel error exceeding 10% of the image size (derived from the stored corners), the corners are considered invalid and are reconstructed from the correspondences.

6.1.2.2 LGSX_SITEMAP_CORNERS_REJECT_DEGENERATE

```
#define LGSX_SITEMAP_CORNERS_REJECT_DEGENERATE 0x04
```

Treat geometrically degenerate stored corners as missing.

If LGSX_SITEMAP_CORNERS_CALCULATE_MISSING is also set, reconstruction from setup/track correspondences is attempted; otherwise the function returns an error.

6.1.3 Basic Reader Functions

Functions

- LgsxReaderPtr Lgsx_ReaderCreate (int ∗rError)
  Create a Reader and return its handle.
- int Lgsx_ReaderOpen (LgsxReaderPtr reader, const wchar_t ∗pFilePath, const char ∗password)
  Open the given LGSx file with the given Reader using the given password.
- int Lgsx_ReaderClose (LgsxReaderPtr reader)
  Closes the LGSx file for this Reader.
- int Lgsx_ReaderGetLastError (LgsxReaderPtr reader, char ∗pError, int maxLen)
  Returns a string description of the last error that occurred with the given Reader.
- int Lgsx_ReaderGetLastError2 (LgsxReaderPtr reader, char ∗∗pErrorPtr)
  Returns a string description of the last error that occurred with the given Reader.
- int Lgsx_ReaderHasPassword (const wchar_t ∗pFilePath, bool ∗hasPassword)
  Checks if file is password protected.

6.1.3.1 Detailed Description

6.1.3.2 Function Documentation

6.1.3.2.1 Lgsx_ReaderClose()

```
int Lgsx_ReaderClose (
            LgsxReaderPtr reader)
```

Closes the LGSx file for this Reader.

Parameters

in	*reader*	Reader handle.

Returns

0 for success.

6.1.3.2.2 Lgsx_ReaderCreate()

```
LgsxReaderPtr Lgsx_ReaderCreate (
            int * rError)
```

Create a Reader and return its handle.

Parameters

out	*rError*	Error code.

Returns

Error code is set to 0 for success and -1 for failure.

6.1.3.2.3 Lgsx_ReaderGetLastError()

```
int Lgsx_ReaderGetLastError (
            LgsxReaderPtr reader,
            char * pError,
            int maxLen)
```

Returns a string description of the last error that occurred with the given Reader.

See also

Lgsx_ReaderGetLastError2() is a version that always returns complete data without truncation.

Parameters

in	*reader*	Reader handle.
out	*pError*	Error string buffer.
in	*maxLen*	Size of error string buffer.

Returns

0 for success.

6.1.3.2.4 Lgsx_ReaderGetLastError2()

```
int Lgsx_ReaderGetLastError2 (
            LgsxReaderPtr reader,
            char ** pErrorPtr)
```

Returns a string description of the last error that occurred with the given Reader.

Parameters

in	*reader*	Reader handle.
out	*pErrorPtr*	Pointer to store pointer to error string. Buffer must be freed using LgsxUtil_FreeMem().

Returns

0 for success.

6.1.3.2.5 Lgsx_ReaderHasPassword()

```
int Lgsx_ReaderHasPassword (
            const wchar_t * pFilePath,
            bool * hasPassword)
```

Checks if file is password protected.

Parameters

in	*pFilePath*	Full path to the LGSx file to be opened.
out	*hasPassword*	Set to true if the file is password protected.

Returns

0 for success.

6.1.3.2.6 Lgsx_ReaderOpen()

```
int Lgsx_ReaderOpen (
            LgsxReaderPtr reader,
            const wchar_t * pFilePath,
            const char * password)
```

Open the given LGSx file with the given Reader using the given password.

Parameters

in	*reader*	Reader handle.
in	*pFilePath*	Full path to the LGSx file to be opened.
in	*password*	Password needed to open the LGSx file. Set to null if there is no password.

Returns

0 for success.

6.1.4 Project Functions

Functions

- int Lgsx_ReaderGetMetadata (LgsxReaderPtr reader, uint64_t ∗pPointCount, CYBBOX ∗pBbox, LgsxMetadataPtr ∗pMetadata)

  Returns the project metadata for the opened project.
- int Lgsx_ReaderGetProjectInfo (LgsxReaderPtr reader, LgsxMetadataPtr ∗pOwnerInfo, int ∗width, int ∗height, int ∗widthInBytes, ImagePixelType ∗pixelType, unsigned char ∗∗pThumbImage)

  Returns the project metadata and thumbnail for the opened project.
- int Lgsx_ReaderGetProjectDescription (LgsxReaderPtr reader, wchar_t ∗pDesc, int maxLen)

  Returns the project description string for the opened project.
- int Lgsx_ReaderGetProjectDescription2 (LgsxReaderPtr reader, wchar_t ∗∗pDescPtr)

  Returns the project description string for the opened project.

6.1.4.1 Detailed Description

6.1.4.2 Function Documentation

6.1.4.2.1 Lgsx_ReaderGetMetadata()

```
int Lgsx_ReaderGetMetadata (
            LgsxReaderPtr reader,
            uint64_t * pPointCount,
            CYBBOX * pBbox,
            LgsxMetadataPtr * pMetadata)
```

Returns the project metadata for the opened project.

Parameters

in	*reader*	Reader handle.
out	*pPointCount*	Number of points in the point cloud.
out	*pBbox*	Bounding box that contains the project.
out	*pMetadata*	Metadata object that contains additional project metadata. Metadata must be freed using Lgsx_FreeHandle().

Returns

0 for success.

6.1.4.2.2 Lgsx_ReaderGetProjectDescription()

```
int Lgsx_ReaderGetProjectDescription (
            LgsxReaderPtr reader,
            wchar_t * pDesc,
            int maxLen)
```

Returns the project description string for the opened project.

See also

Lgsx_ReaderGetProjectDescription2() is a version that always returns complete data without truncation.

Parameters

in	*reader*	Reader handle.
out	*pDesc*	Description string buffer.
in	*maxLen*	Size of error string buffer.

Returns

0 for success.

6.1.4.2.3 Lgsx_ReaderGetProjectDescription2()

```
int Lgsx_ReaderGetProjectDescription2 (
            LgsxReaderPtr reader,
            wchar_t ** pDescPtr)
```

Returns the project description string for the opened project.

Parameters

in	*reader*	Reader handle.
out	*pDescPtr*	Pointer to store pointer to description. Buffer must be freed using LgsxUtil_FreeMem().

Returns

0 for success.

6.1.4.2.4 Lgsx_ReaderGetProjectInfo()

```
int Lgsx_ReaderGetProjectInfo (
            LgsxReaderPtr reader,
            LgsxMetadataPtr * pOwnerInfo,
            int * width,
            int * height,
            int * widthInBytes,
            ImagePixelType * pixelType,
            unsigned char ** pThumbImage)
```

Returns the project metadata and thumbnail for the opened project.

If the thumbnail is not available, width, height, widthInBytes will be set to zero and pThumbImage to NULL.

Parameters

in	*reader*	Reader handle.
out	*pOwnerInfo*	Project metadata. Metadata must be freed using Lgsx_FreeHandle().
out	*width*	Thumbnail image width.
out	*height*	Thumbnail image height.
out	*widthInBytes*	Thumbnail image byte-width.
out	*pixelType*	Thumbnail image pixel type.
out	*pThumbImage*	Thumbnail image data.

Returns

0 for success.

6.1.5 Asset Functions

Data Structures

- struct AssetHandle

  Asset data handle for accessing Asset data without buffer truncation.

Functions

- int Lgsx_ReaderEnumAssets (LgsxReaderPtr reader, int ∗pNumAsset, bool bGetAll)

  Collects assets in the project and returns the count.
- int Lgsx_ReaderEndAssets (LgsxReaderPtr reader)

  Ends the current Asset enumeration.
- int Lgsx_ReaderMoveNextAsset (LgsxReaderPtr reader, uint64_t ∗assetId)

  Advance to the next Asset in the active collection and return its ID.
- int Lgsx_ReaderGetAsset (LgsxReaderPtr reader, uint64_t assetId, CYASSET ∗pAsset, LgsxMetadataPtr ∗pMetadata)

  Get Info and metadata for current asset as per Lgsx_ReaderMoveNextAsset() context.
- int Lgsx_ReaderAssetGetGuid (LgsxReaderPtr reader, const CYASSET ∗pAsset, char ∗∗pGuidPtr)

  Returns the Asset GUID for the asset passed as parameter.
- int Lgsx_GetAssetAsImage (LgsxReaderPtr reader, uint64_t assetId, wchar_t ∗pName, int maxLenName, int ∗width, int ∗height, int ∗widthInBytes, ImagePixelType ∗pixelType, unsigned char ∗∗pImage)

Get the "payload" image for current asset as per Lgsx_ReaderMoveNextAsset() context.

- int Lgsx_GetAssetAsImage2 (LgsxReaderPtr reader, uint64_t assetId, wchar_t **pNamePtr, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char **pImage)

  Get the "payload" image for current asset as per Lgsx_ReaderMoveNextAsset() context.
- int Lgsx_AssetReadImage (const CYASSET *pAsset, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **pImage)

  Get the "payload" as image for the given Asset.
- int Lgsx_AssetReadBinary (const CYASSET *pAsset, unsigned char **pData, int *pSize)

  Get the "payload" as binary data for the given Asset.
- int Lgsx_GetAssetThumbImage (LgsxReaderPtr reader, uint64_t assetId, int *width, int *height, int *width↩ InBytes, ImagePixelType *pixelType, unsigned char **pImage)

  Get the thumbnail image for current asset as per Lgsx_ReaderMoveNextAsset() context.
- int Lgsx_AssetHasType (LgsxReaderPtr reader, uint64_t assetId, bool *pHasAssetType)

  Check if an Asset has a type set.
- int Lgsx_ReaderGetAssetHandle (LgsxReaderPtr reader, uint64_t assetId, AssetHandle *pHandle)

  Acquire an Asset handle for the Asset with the given ID.
- int Lgsx_AssetRelease (AssetHandle *pHandle)

  Release the Asset handle returned by Lgsx_ReaderGetGeoTagAssetHandle() or Lgsx_ReaderGetAssetHandle().
- int Lgsx_AssetGetName (const AssetHandle handle, const wchar_t **pNamePtr)

  Get the name of the Asset.
- int Lgsx_AssetGetType (const AssetHandle handle, const wchar_t **pTypePtr)

  Get the type of the Asset.
- int Lgsx_AssetGetMimeType (const AssetHandle handle, const wchar_t **pMimeTypePtr)

  Get the MIME type of the Asset.
- int Lgsx_AssetGetFilename (const AssetHandle handle, const wchar_t **pFilenamePtr)

  Get the extracted file path of the Asset payload.
- int Lgsx_AssetGetUrl (const AssetHandle handle, const wchar_t **pUrlPtr)

  Get the external URL of the Asset (for URL-type Assets).
- int Lgsx_AssetGetGuid (const AssetHandle handle, const char **pGuidPtr)

  Get the GUID of the Asset.
- int Lgsx_AssetGetId (const AssetHandle handle, uint64_t *pId)

  Get the ID of the Asset.
- int Lgsx_AssetGetCreationTime (const AssetHandle handle, uint64_t *pCreationTime)

  Get the creation timestamp of the Asset.
- int Lgsx_AssetGetModificationTime (const AssetHandle handle, uint64_t *pModificationTime)

  Get the modification timestamp of the Asset.
- int Lgsx_AssetIsExternalUrl (const AssetHandle handle, bool *pIsExternalUrl)

  Check whether the Asset is an external URL.
- int Lgsx_AssetGetMetadata (const AssetHandle handle, LgsxMetadataPtr *pMetadata)

  Get the metadata of the Asset.
- int Lgsx_AssetHandleReadImage (const AssetHandle handle, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **pImage)

  Read the Asset payload as an image.
- int Lgsx_AssetHandleReadBinary (const AssetHandle handle, unsigned char **pData, int *pSize)

  Read the Asset payload as binary data.
- int Lgsx_AssetHandleReadImageBinary (LgsxReaderPtr reader, const AssetHandle handle, wchar_t **p↩ ImageType, int *pWidth, int *pHeight, unsigned char **pData, int *pSize)

  Read the raw encoded image bytes from the Asset, stripping SDK metadata.

6.1.5.1 Detailed Description

6.1.5.2 Function Documentation

6.1.5.2.1 Lgsx_AssetGetCreationTime()

```
int Lgsx_AssetGetCreationTime (
            const AssetHandle handle,
            uint64_t * pCreationTime)
```

Get the creation timestamp of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pCreationTime*	Pointer to store the creation timestamp.

Returns

Returns 0 for success.

6.1.5.2.2 Lgsx_AssetGetFilename()

```
int Lgsx_AssetGetFilename (
            const AssetHandle handle,
            const wchar_t ** pFilenamePtr)
```

Get the extracted file path of the Asset payload.

For URL-type Assets this returns an empty string; use Lgsx_AssetGetUrl() instead.

Parameters

in	*handle*	Asset handle.
out	*pFilenamePtr*	Pointer to store the filename buffer. Buffer is read-only, bound to the lifetime of the Asset handle. Must not be freed.

Returns

Returns 0 for success.

6.1.5.2.3 Lgsx_AssetGetGuid()

```
int Lgsx_AssetGetGuid (
            const AssetHandle handle,
            const char ** pGuidPtr)
```

Get the GUID of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pGuidPtr*	Pointer to store the GUID buffer. Buffer is read-only, bound to the lifetime of the Asset handle. Must not be freed. May be an empty string if the source object does not have a GUID.

Returns

Returns 0 for success.

6.1.5.2.4 Lgsx_AssetGetId()

```
int Lgsx_AssetGetId (
            const AssetHandle handle,
            uint64_t * pId)
```

Get the ID of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pId*	Pointer to store the Asset ID.

Returns

Returns 0 for success.

6.1.5.2.5 Lgsx_AssetGetMetadata()

```
int Lgsx_AssetGetMetadata (
            const AssetHandle handle,
            LgsxMetadataPtr * pMetadata)
```

Get the metadata of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pMetadata*	Asset metadata. Metadata is bound to the lifetime of the Asset handle. Must not be freed.

Returns

Returns 0 for success.

6.1.5.2.6 Lgsx_AssetGetMimeType()

```
int Lgsx_AssetGetMimeType (
            const AssetHandle handle,
            const wchar_t ** pMimeTypePtr)
```

Get the MIME type of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pMimeTypePtr*	Pointer to store the MIME type buffer. Buffer is read-only, bound to the lifetime of the Asset handle. Must not be freed.

Returns

Returns 0 for success.

6.1.5.2.7 Lgsx_AssetGetModificationTime()

```
int Lgsx_AssetGetModificationTime (
            const AssetHandle handle,
            uint64_t * pModificationTime)
```

Get the modification timestamp of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pModificationTime*	Pointer to store the modification timestamp.

Returns

Returns 0 for success.

6.1.5.2.8 Lgsx_AssetGetName()

```
int Lgsx_AssetGetName (
            const AssetHandle handle,
            const wchar_t ** pNamePtr)
```

Get the name of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pNamePtr*	Pointer to store the name buffer. Buffer is read-only, bound to the lifetime of the Asset handle. Must not be freed.

Returns

Returns 0 for success.

6.1.5.2.9 Lgsx_AssetGetType()

```
int Lgsx_AssetGetType (
            const AssetHandle handle,
            const wchar_t ** pTypePtr)
```

Get the type of the Asset.

Parameters

in	*handle*	Asset handle.
out	*pTypePtr*	Pointer to store the type buffer. Buffer is read-only, bound to the lifetime of the Asset handle. Must not be freed.

Returns

Returns 0 for success.

6.1.5.2.10 Lgsx_AssetGetUrl()

```
int Lgsx_AssetGetUrl (
            const AssetHandle handle,
            const wchar_t ** pUrlPtr)
```

Get the external URL of the Asset (for URL-type Assets).

Parameters

in	*handle*	Asset handle.
out	*pUrlPtr*	Pointer to store the URL buffer. Buffer is read-only, bound to the lifetime of the Asset handle. Must not be freed.

Returns

Returns 0 for success.

6.1.5.2.11 Lgsx_AssetHandleReadBinary()

```
int Lgsx_AssetHandleReadBinary (
            const AssetHandle handle,
            unsigned char ** pData,
            int * pSize)
```

Read the Asset payload as binary data.

See also

Lgsx_AssetReadBinary() for the alternative using CYASSET.

Parameters

in	*handle*	Asset handle.
out	*pData*	Pointer to store pointer to binary data. Buffer is read-only.
out	*pSize*	Number of bytes in the binary data.

Returns

Returns 0 for success.

6.1.5.2.12 Lgsx_AssetHandleReadImage()

```
int Lgsx_AssetHandleReadImage (
            const AssetHandle handle,
            int * pWidth,
            int * pHeight,
            int * pWidthInBytes,
            ImagePixelType * pPixelType,
            unsigned char ** pImage)
```

Read the Asset payload as an image.

See also

Lgsx_AssetReadImage() for the alternative using CYASSET.

Parameters

in	*handle*	Asset handle.
out	*pWidth*	Image width.
out	*pHeight*	Image height.
out	*pWidthInBytes*	Image width in bytes.
out	*pPixelType*	Pixel type.
out	*pImage*	Asset image.

Returns

Returns 0 for success.

6.1.5.2.13 Lgsx_AssetHandleReadImageBinary()

```
int Lgsx_AssetHandleReadImageBinary (
            LgsxReaderPtr reader,
            const AssetHandle handle,
            wchar_t ** pImageType,
            int * pWidth,
            int * pHeight,
            unsigned char ** pData,
            int * pSize)
```

Read the raw encoded image bytes from the Asset, stripping SDK metadata.

Returns the original image data without decoding. The caller receives the encoded bytes, the image type string, and pixel dimensions.

Note

This function may fail if the image format stored in the asset is not recognized by the implementation. Callers should fall back to the decoded-image path (Lgsx_AssetHandleReadImage()) when this function returns a non-zero error code.

Parameters

in	*reader*	Reader handle.
in	*handle*	Asset handle obtained via Lgsx_ReaderGetAssetHandle() or Lgsx_SitemapImageGetHandle().
out	*pImageType*	Image type string (e.g. L"jpg", L"png"). Free with LgsxUtil_FreeMem().
out	*pWidth*	Image width in pixels.
out	*pHeight*	Image height in pixels.
out	*pData*	Pointer to receive the raw image bytes. Free with LgsxUtil_FreeMem().
out	*pSize*	Number of bytes in the pData buffer.

Returns

Returns 0 for success.

6.1.5.2.14 Lgsx_AssetHasType()

```
int Lgsx_AssetHasType (
            LgsxReaderPtr reader,
            uint64_t assetId,
            bool * pHasAssetType)
```

Check if an Asset has a type set.

Parameters

in	*reader*	Reader handle.
in	*assetId*	Asset ID.
out	*pHasAssetType*	Pointer to store whether the asset has the type set.

Returns

Returns 0 for success.

6.1.5.2.15 Lgsx_AssetIsExternalUrl()

```
int Lgsx_AssetIsExternalUrl (
            const AssetHandle handle,
            bool * pIsExternalUrl)
```

Check whether the Asset is an external URL.

Parameters

in	*handle*	Asset handle.
out	*pIsExternalUrl*	Pointer to store the result.

Returns

Returns 0 for success.

6.1.5.2.16 Lgsx_AssetReadBinary()

```
int Lgsx_AssetReadBinary (
            const CYASSET * pAsset,
            unsigned char ** pData,
            int * pSize)
```

Get the "payload" as binary data for the given Asset.

Parameters

in	*pAsset*	Asset info.
out	*pData*	Pointer to store pointer to binary data. Buffer is read-only.
out	*pSize*	Number of bytes in the binary data.

Returns

Returns 0 for success.

6.1.5.2.17 Lgsx_AssetReadImage()

```
int Lgsx_AssetReadImage (
            const CYASSET * pAsset,
            int * pWidth,
            int * pHeight,
            int * pWidthInBytes,
            ImagePixelType * pPixelType,
            unsigned char ** pImage)
```

Get the "payload" as image for the given Asset.

Parameters

in	*pAsset*	Asset info.
out	*pWidth*	Image width.
out	*pHeight*	Image height.
out	*pWidthInBytes*	Image width in bytes.
out	*pPixelType*	Pixel type.
out	*pImage*	Asset image.

Returns

Returns 0 for success.

6.1.5.2.18 Lgsx_AssetRelease()

```
int Lgsx_AssetRelease (
            AssetHandle * pHandle)
```

Release the Asset handle returned by Lgsx_ReaderGetGeoTagAssetHandle() or Lgsx_ReaderGetAssetHandle().

Parameters

in	*pHandle*	Pointer to the Asset handle to release.

Returns

Returns 0 for success.

6.1.5.2.19 Lgsx_GetAssetAsImage()

```
int Lgsx_GetAssetAsImage (
            LgsxReaderPtr reader,
            uint64_t assetId,
            wchar_t * pName,
            int maxLenName,
            int * width,
            int * height,
            int * widthInBytes,
            ImagePixelType * pixelType,
            unsigned char ** pImage)
```

Get the "payload" image for current asset as per Lgsx_ReaderMoveNextAsset() context.

See also

Lgsx_GetAssetAsImage2() is a version that always returns complete data without truncation.

Parameters

in	*reader*	Reader handle.
in	*assetId*	Asset ID.
out	*pName*	Asset filename buffer.
in	*maxLenName*	Asset filename buffer length.
out	*width*	Image width.
out	*height*	Image height.
out	*widthInBytes*	Image width in bytes.
out	*pixelType*	Pixel type.
out	*pImage*	Asset image.

Returns

Returns 0 for success.

6.1.5.2.20 Lgsx_GetAssetAsImage2()

```
int Lgsx_GetAssetAsImage2 (
            LgsxReaderPtr reader,
            uint64_t assetId,
            wchar_t ** pNamePtr,
            int * width,
            int * height,
            int * widthInBytes,
            ImagePixelType * pixelType,
            unsigned char ** pImage)
```

Get the "payload" image for current asset as per Lgsx_ReaderMoveNextAsset() context.

Parameters

in	*reader*	Reader handle.
in	*assetId*	Asset ID.
out	*pNamePtr*	Pointer to store pointer to asset filename. Buffer must be freed using LgsxUtil_FreeMem().
out	*width*	Image width.
out	*height*	Image height.
out	*widthInBytes*	Image width in bytes.
out	*pixelType*	Pixel type.
out	*pImage*	Asset image.

Returns

Returns 0 for success.

6.1.5.2.21 Lgsx_GetAssetThumbImage()

```
int Lgsx_GetAssetThumbImage (
            LgsxReaderPtr reader,
            uint64_t assetId,
            int * width,
            int * height,
            int * widthInBytes,
            ImagePixelType * pixelType,
            unsigned char ** pImage)
```

Get the thumbnail image for current asset as per Lgsx_ReaderMoveNextAsset() context.

Parameters

in	*reader*	Reader handle.
in	*assetId*	Asset ID.
out	*width*	Image width.
out	*height*	Image height.
out	*widthInBytes*	Image width in bytes.
out	*pixelType*	Pixel type.
out	*pImage*	Asset thumbnail image.

Returns

Returns 0 for success.

6.1.5.2.22 Lgsx_ReaderAssetGetGuid()

```
int Lgsx_ReaderAssetGetGuid (
            LgsxReaderPtr reader,
            const CYASSET * pAsset,
            char ** pGuidPtr)
```

Returns the Asset GUID for the asset passed as parameter.

Parameters

in	*reader*	Reader handle.
in	*pAsset*	Pointer to the asset information structure.
out	*pGuidPtr*	Pointer to store the GUID of the asset. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.5.2.23 Lgsx_ReaderEndAssets()

```
int Lgsx_ReaderEndAssets (
            LgsxReaderPtr reader)
```

Ends the current Asset enumeration.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.5.2.24 Lgsx_ReaderEnumAssets()

```
int Lgsx_ReaderEnumAssets (
            LgsxReaderPtr reader,
            int * pNumAsset,
            bool bGetAll)
```

Collects assets in the project and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumAsset*	Number of collected Assets.
in	*bGetAll*	If TRUE, retrieve all assets in the project. If FALSE, only return Assets that are owned directly by the project (that is, exclude Assets that are owned by GeoTags, Sitemaps, or Models).

Returns

0 for success.

6.1.5.2.25 Lgsx_ReaderGetAsset()

```
int Lgsx_ReaderGetAsset (
            LgsxReaderPtr reader,
            uint64_t assetId,
            CYASSET * pAsset,
            LgsxMetadataPtr * pMetadata)
```

Get Info and metadata for current asset as per Lgsx_ReaderMoveNextAsset() context.

Parameters

in	*reader*	Reader handle.
in	*assetId*	Asset ID
out	*pAsset*	Asset info.
out	*pMetadata*	Asset metadata. Metadata must be freed using Lgsx_FreeHandle().

Returns

Returns 0 for success.

6.1.5.2.26 Lgsx_ReaderGetAssetHandle()

```
int Lgsx_ReaderGetAssetHandle (
            LgsxReaderPtr reader,
            uint64_t assetId,
            AssetHandle * pHandle)
```

Acquire an Asset handle for the Asset with the given ID.

Note

Extracts the asset payload to a temporary file owned by the reader. Only one extracted file is kept at a time: calling this again (for this or any other asset) deletes the previous handle's temp file, so do not hold two AssetHandles concurrently. The file is also removed by Lgsx_ReaderMoveNextAsset() and Lgsx_ReaderEndAssets().

Returns non-zero if the asset payload cannot be extracted to the SDK temp folder (I/O error, including running out of space). Use Lgsx_GetLastError() / Lgsx_GetLastErrorNo() to retrieve a descriptive message; when both can be measured, the asset size and current temp-folder free space are included as a hint that insufficient space may be the cause.

Parameters

in	*reader*	Reader handle.
in	*assetId*	Asset ID.
out	*pHandle*	Pointer to store the Asset handle. Buffer must be freed using Lgsx_AssetRelease().

Returns

Returns 0 for success.

6.1.5.2.27 Lgsx_ReaderMoveNextAsset()

```
int Lgsx_ReaderMoveNextAsset (
            LgsxReaderPtr reader,
            uint64_t * assetId)
```

Advance to the next Asset in the active collection and return its ID.

Note

Deletes the temporary file backing the previous asset, if any. Any AssetHandle previously obtained from Lgsx_ReaderGetAssetHandle() must not be used after this call - its filename is no longer on disk.

Parameters

in	*reader*	Reader handle.
out	*asset↩ Id*	Asset ID.

Returns

Returns 0 for success. Returns -1 when end of collection is reached.

6.1.6 Categories Functions

Categories.

Data Structures

- struct CategoryHandle

  Descriptor for a Category object.
- struct CategoryValueHandle

  Descriptor for a Category Value object.
- struct CategoryEntryHandle

  Descriptor for a Category Entry object.

Functions

- int Lgsx_ReaderEnumCategories (LgsxReaderPtr reader, int ∗pNumCategories)

  Collects Categories in the project and returns the count.
- int Lgsx_ReaderGetCategory (LgsxReaderPtr reader, int pos, CategoryHandle ∗pHandle)

  Read Category data at the given position in the active collection.
- int Lgsx_CategoryRelease (CategoryHandle ∗pHandle)

  Release the Category data pointer returned by Lgsx_ReaderGetCategory().
- int Lgsx_CategoryGetGuid (const CategoryHandle handle, const char ∗∗pId)

  Get the GUID of the Category.
- int Lgsx_CategoryGetName (const CategoryHandle handle, const wchar_t ∗∗pName)

  Get the name of the Category.
- int Lgsx_CategoryGetPriority (const CategoryHandle handle, int ∗pPriority)

  Get the priority of the Category.
- int Lgsx_CategoryGetCreationTimestamp (const CategoryHandle handle, uint64_t ∗pTimestamp)

  Get the creation timestamp of the Category.
- int Lgsx_CategoryGetModificationTimestamp (const CategoryHandle handle, uint64_t ∗pTimestamp)

  Get the modification timestamp of the Category.
- int Lgsx_CategoryEnumValues (const CategoryHandle handle, int ∗pNumValues)

  Enumerate the values within a Category and return the count.
- int Lgsx_CategoryGetValue (const CategoryHandle handle, int pos, CategoryValueHandle ∗pValueHandle)

  Get a Category Value at the specified position.
- int Lgsx_CategoryValueGetGuid (const CategoryValueHandle handle, const char ∗∗pId)

  Get the GUID of a Category Value.
- int Lgsx_CategoryValueGetValue (const CategoryValueHandle handle, const wchar_t ∗∗pValue)

Get the value of a Category Value.

- int Lgsx_CategoryValueGetPriority (const CategoryValueHandle handle, int *pPriority)

  Get the priority of a Category Value.
- int Lgsx_CategoryValueGetCreationTimestamp (const CategoryValueHandle handle, uint64_t *pTimestamp)

  Get the creation timestamp of a Category Value.
- int Lgsx_CategoryValueGetModificationTimestamp (const CategoryValueHandle handle, uint64_t *pTimestamp)

  Get the modification timestamp of a Category Value.
- int Lgsx_CategoryValueHasColor (const CategoryValueHandle handle, bool *pStatus)

  Check if a Category Value has a color associated with it.
- int Lgsx_CategoryValueGetColor (const CategoryValueHandle handle, CYRGBA *pColor)

  Get the color of a Category Value.
- int Lgsx_CategoryEntryGetCategory (const CategoryEntryHandle handle, CategoryHandle *pCategoryHandle)

  Get the Category from a Category Entry.
- int Lgsx_CategoryEntryHasValue (const CategoryEntryHandle handle, bool *pStatus)

  Check if a Category Entry has a value associated with it.
- int Lgsx_CategoryEntryGetValue (const CategoryEntryHandle handle, CategoryValueHandle *pValueHandle)

  Get the Category Value from a Category Entry.

6.1.6.1 Detailed Description

Categories.

6.1.6.2 Function Documentation

6.1.6.2.1 Lgsx_CategoryEntryGetCategory()

```
int Lgsx_CategoryEntryGetCategory (
            const CategoryEntryHandle handle,
            CategoryHandle * pCategoryHandle)
```

Get the Category from a Category Entry.

Parameters

in	*handle*	Category Entry handle.
out	*pCategoryHandle*	Pointer to store the Category handle. Returns read-only pointer bound to lifetime of CategoryEntryHandle, shall not be released manually.

Returns

Returns 0 for success.

6.1.6.2.2 Lgsx_CategoryEntryGetValue()

```
int Lgsx_CategoryEntryGetValue (
            const CategoryEntryHandle handle,
            CategoryValueHandle * pValueHandle)
```

Get the Category Value from a Category Entry.

Parameters

in	*handle*	Category Entry handle.
out	*pValueHandle*	Pointer to store the Category Value handle. Returns read-only pointer bound to lifetime of CategoryEntryHandle, shall not be released manually. Might return null pointer if the Category Entry does not have a value associated with it.

Returns

Returns 0 for success.

6.1.6.2.3 Lgsx_CategoryEntryHasValue()

```
int Lgsx_CategoryEntryHasValue (
            const CategoryEntryHandle handle,
            bool * pStatus)
```

Check if a Category Entry has a value associated with it.

Parameters

in	*handle*	Category Entry handle.
out	*pStatus*	Pointer to store the status.

Returns

Returns 0 for success.

6.1.6.2.4 Lgsx_CategoryEnumValues()

```
int Lgsx_CategoryEnumValues (
            const CategoryHandle handle,
            int * pNumValues)
```

Enumerate the values within a Category and return the count.

Parameters

in	*handle*	Category handle.
out	*pNumValues*	Pointer to store the number of values in the Category.

Returns

Returns 0 for success.

6.1.6.2.5 Lgsx_CategoryGetCreationTimestamp()

```
int Lgsx_CategoryGetCreationTimestamp (
            const CategoryHandle handle,
            uint64_t * pTimestamp)
```

Get the creation timestamp of the Category.

Parameters

in	*handle*	Category handle.
out	*pTimestamp*	Pointer to store the creation timestamp of the Category.

Returns

Returns 0 for success.

6.1.6.2.6 Lgsx_CategoryGetGuid()

```
int Lgsx_CategoryGetGuid (
            const CategoryHandle handle,
            const char ** pId)
```

Get the GUID of the Category.

Parameters

in	*handle*	Category handle.
out	*pId*	Pointer to store the GUID string of the Category.

Returns

Returns 0 for success.

6.1.6.2.7 Lgsx_CategoryGetModificationTimestamp()

```
int Lgsx_CategoryGetModificationTimestamp (
            const CategoryHandle handle,
            uint64_t * pTimestamp)
```

Get the modification timestamp of the Category.

Parameters

in	*handle*	Category handle.
out	*pTimestamp*	Pointer to store the modification timestamp of the Category.

Returns

Returns 0 for success.

6.1.6.2.8 Lgsx_CategoryGetName()

```
int Lgsx_CategoryGetName (
            const CategoryHandle handle,
            const wchar_t ** pName)
```

Get the name of the Category.

Parameters

in	*handle*	Category handle.
out	*pName*	Pointer to store the name of the Category.

Returns

Returns 0 for success.

6.1.6.2.9 Lgsx_CategoryGetPriority()

```
int Lgsx_CategoryGetPriority (
            const CategoryHandle handle,
            int * pPriority)
```

Get the priority of the Category.

Parameters

in	*handle*	Category handle.
out	*pPriority*	Pointer to store the priority value of the Category.

Returns

Returns 0 for success.

6.1.6.2.10 Lgsx_CategoryGetValue()

```
int Lgsx_CategoryGetValue (
            const CategoryHandle handle,
            int pos,
            CategoryValueHandle * pValueHandle)
```

Get a Category Value at the specified position.

Parameters

in	*handle*	Category handle.
in	*pos*	Position in the Category's value collection.
out	*pValueHandle*	Pointer to store the Category Value handle. Returns read-only pointer bound to lifetime of CategoryHandle, shall not be released manually.

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.6.2.11 Lgsx_CategoryRelease()

```
int Lgsx_CategoryRelease (
            CategoryHandle * pHandle)
```

Release the Category data pointer returned by Lgsx_ReaderGetCategory().

Parameters

in	*pHandle*	Pointer to Category data

Returns

Returns 0 for success.

6.1.6.2.12 Lgsx_CategoryValueGetColor()

```
int Lgsx_CategoryValueGetColor (
            const CategoryValueHandle handle,
            CYRGBA * pColor)
```

Get the color of a Category Value.

Parameters

in	*handle*	Category Value handle.
out	*pColor*	Pointer to store the RGBA color of the Category Value.

Returns

Returns 0 for success.

6.1.6.2.13 Lgsx_CategoryValueGetCreationTimestamp()

```
int Lgsx_CategoryValueGetCreationTimestamp (
            const CategoryValueHandle handle,
            uint64_t * pTimestamp)
```

Get the creation timestamp of a Category Value.

Parameters

in	*handle*	Category Value handle.
out	*pTimestamp*	Pointer to store the creation timestamp of the Category Value.

Returns

Returns 0 for success.

6.1.6.2.14 Lgsx_CategoryValueGetGuid()

```
int Lgsx_CategoryValueGetGuid (
            const CategoryValueHandle handle,
            const char ** pId)
```

Get the GUID of a Category Value.

Parameters

in	*handle*	Category Value handle.
out	*pId*	Pointer to store the ID string of the Category Value.

Returns

Returns 0 for success.

6.1.6.2.15 Lgsx_CategoryValueGetModificationTimestamp()

```
int Lgsx_CategoryValueGetModificationTimestamp (
            const CategoryValueHandle handle,
            uint64_t * pTimestamp)
```

Get the modification timestamp of a Category Value.

Parameters

in	*handle*	Category Value handle.
out	*pTimestamp*	Pointer to store the modification timestamp of the Category Value.

Returns

Returns 0 for success.

6.1.6.2.16 Lgsx_CategoryValueGetPriority()

```
int Lgsx_CategoryValueGetPriority (
            const CategoryValueHandle handle,
            int * pPriority)
```

Get the priority of a Category Value.

Parameters

in	*handle*	Category Value handle.
out	*pPriority*	Pointer to store the priority value of the Category Value.

Returns

Returns 0 for success.

6.1.6.2.17 Lgsx_CategoryValueGetValue()

```
int Lgsx_CategoryValueGetValue (
            const CategoryValueHandle handle,
            const wchar_t ** pValue)
```

Get the value of a Category Value.

Parameters

in	*handle*	Category Value handle.
out	*pValue*	Pointer to store the value string of the Category Value.

Returns

Returns 0 for success.

6.1.6.2.18 Lgsx_CategoryValueHasColor()

```
int Lgsx_CategoryValueHasColor (
            const CategoryValueHandle handle,
            bool * pStatus)
```

Check if a Category Value has a color associated with it.

Parameters

in	*handle*	Category Value handle.
out	*pStatus*	Pointer to store the status. Set to true if the Category Value has a color associated with it.

Returns

Returns 0 for success.

6.1.6.2.19 Lgsx_ReaderEnumCategories()

```
int Lgsx_ReaderEnumCategories (
            LgsxReaderPtr reader,
            int * pNumCategories)
```

Collects Categories in the project and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumCategories*	Number of collected Categories.

Returns

Returns 0 for success.

6.1.6.2.20 Lgsx_ReaderGetCategory()

```
int Lgsx_ReaderGetCategory (
            LgsxReaderPtr reader,
            int pos,
            CategoryHandle * pHandle)
```

Read Category data at the given position in the active collection.

Active collection is set by Lgsx_ReaderEnumCategories()

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in the active collection.
out	*pHandle*	Pointer to store pointer to Category data. The pointer has to be released using Lgsx_CategoryRelease()

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.7 Fields Functions

Fields.

Data Structures

- struct FieldHandle

  Descriptor for a Field object.
- struct FieldEntryHandle

  Descriptor for a Field Entry object.

Functions

- int Lgsx_ReaderEnumFields (LgsxReaderPtr reader, int *pNumFields)

  Collects Fields in the project and returns the count.
- int Lgsx_ReaderGetField (LgsxReaderPtr reader, int pos, FieldHandle *pHandle)

  Read Field data at the given position in the active collection.
- int Lgsx_FieldRelease (FieldHandle *pHandle)

  Release the Field data pointer returned by Lgsx_ReaderGetField().
- int Lgsx_FieldGetGuid (const FieldHandle handle, const char **pId)

  Get the GUID of the Field.
- int Lgsx_FieldGetName (const FieldHandle handle, const wchar_t **pName)

  Get the name of the Field.
- int Lgsx_FieldGetPriority (const FieldHandle handle, int *pPriority)

  Get the priority of the Field.
- int Lgsx_FieldGetCreationTimestamp (const FieldHandle handle, uint64_t *pTimestamp)

  Get the creation timestamp of the Field.
- int Lgsx_FieldGetModificationTimestamp (const FieldHandle handle, uint64_t *pTimestamp)

  Get the modification timestamp of the Field.
- int Lgsx_FieldEntryGetField (const FieldEntryHandle handle, FieldHandle *pFieldHandle)

  Get the Field from a Field Entry.
- int Lgsx_FieldEntryGetValue (const FieldEntryHandle handle, const wchar_t **pValue)

  Get the value from a Field Entry.

6.1.7.1 Detailed Description

Fields.

6.1.7.2 Function Documentation

6.1.7.2.1 Lgsx_FieldEntryGetField()

```
int Lgsx_FieldEntryGetField (
            const FieldEntryHandle handle,
            FieldHandle * pFieldHandle)
```

Get the Field from a Field Entry.

Parameters

in	*handle*	Field Entry handle.
out	*pFieldHandle*	Pointer to store the Field handle. Returns read-only pointer bound to lifetime of FieldEntryHandle, shall not be released manually.

Returns

Returns 0 for success.

6.1.7.2.2 Lgsx_FieldEntryGetValue()

```
int Lgsx_FieldEntryGetValue (
            const FieldEntryHandle handle,
            const wchar_t ** pValue)
```

Get the value from a Field Entry.

Parameters

in	*handle*	Field Entry handle.
out	*pValue*	Pointer to store the value string of the Field Entry.

Returns

Returns 0 for success.

6.1.7.2.3 Lgsx_FieldGetCreationTimestamp()

```
int Lgsx_FieldGetCreationTimestamp (
            const FieldHandle handle,
            uint64_t * pTimestamp)
```

Get the creation timestamp of the Field.

Parameters

in	*handle*	Field handle.
out	*pTimestamp*	Pointer to store the creation timestamp of the Field.

Returns

Returns 0 for success.

6.1.7.2.4 Lgsx_FieldGetGuid()

```
int Lgsx_FieldGetGuid (
            const FieldHandle handle,
            const char ** pId)
```

Get the GUID of the Field.

Parameters

in	*handle*	Field handle.
out	*pId*	Pointer to store the GUID string of the Field.

Returns

Returns 0 for success.

6.1.7.2.5 Lgsx_FieldGetModificationTimestamp()

```
int Lgsx_FieldGetModificationTimestamp (
            const FieldHandle handle,
            uint64_t * pTimestamp)
```

Get the modification timestamp of the Field.

Parameters

in	*handle*	Field handle.
out	*pTimestamp*	Pointer to store the modification timestamp of the Field.

Returns

Returns 0 for success.

6.1.7.2.6 Lgsx_FieldGetName()

```
int Lgsx_FieldGetName (
            const FieldHandle handle,
            const wchar_t ** pName)
```

Get the name of the Field.

Parameters

in	*handle*	Field handle.
out	*pName*	Pointer to store the name of the Field.

Returns

Returns 0 for success.

6.1.7.2.7 Lgsx_FieldGetPriority()

```
int Lgsx_FieldGetPriority (
            const FieldHandle handle,
            int * pPriority)
```

Get the priority of the Field.

Parameters

in	*handle*	Field handle.
out	*pPriority*	Pointer to store the priority value of the Field.

Returns

Returns 0 for success.

6.1.7.2.8 Lgsx_FieldRelease()

```
int Lgsx_FieldRelease (
            FieldHandle * pHandle)
```

Release the Field data pointer returned by Lgsx_ReaderGetField().

Parameters

in	*pHandle*	Pointer to Field data

Returns

Returns 0 for success.

6.1.7.2.9 Lgsx_ReaderEnumFields()

```
int Lgsx_ReaderEnumFields (
            LgsxReaderPtr reader,
            int * pNumFields)
```

Collects Fields in the project and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumFields*	Number of collected Fields.

Returns

Returns 0 for success.

6.1.7.2.10 Lgsx_ReaderGetField()

```
int Lgsx_ReaderGetField (
            LgsxReaderPtr reader,
            int pos,
            FieldHandle * pHandle)
```

Read Field data at the given position in the active collection.

Active collection is set by Lgsx_ReaderEnumFields()

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in the active collection.
out	*pHandle*	Pointer to store pointer to Field data. The pointer has to be released using Lgsx_FieldRelease()

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.8 Migrated Fields

Old Geotag's metadata "Category", "Label" and "TagIndex" are now automatically migrated to new Fields.

Enumerations

- enum LgsxMigratedMetaKeyField {
  LgsxMigratedMetaKeyField_Category = 0 ,
  LgsxMigratedMetaKeyField_Label ,
  LgsxMigratedMetaKeyField_TagIndex }
  Enum for the migrated metadata.

Functions

- int Lgsx_GetMigratedFieldGuid (LgsxMigratedMetaKeyField field, const char **pId)
  Get the GUID of the migrated Field specified by "field" argument.

6.1.8.1 Detailed Description

Old Geotag's metadata "Category", "Label" and "TagIndex" are now automatically migrated to new Fields.

Those new fields have predefined GUIDs that can be retrieved using the function below.

6.1.8.2 Enumeration Type Documentation

6.1.8.2.1 LgsxMigratedMetaKeyField

`enum LgsxMigratedMetaKeyField`

Enum for the migrated metadata.

Each value corresponds to specific migrated field.

Enumerator

LgsxMigratedMetaKeyField_Category	Field "Category".
LgsxMigratedMetaKeyField_Label	Field "Label".
LgsxMigratedMetaKeyField_TagIndex	Field "TagIndex".

6.1.8.3 Function Documentation

6.1.8.3.1 Lgsx_GetMigratedFieldGuid()

```
int Lgsx_GetMigratedFieldGuid (
            LgsxMigratedMetaKeyField field,
            const char ** pId)
```

Get the GUID of the migrated Field specified by "field" argument.

Parameters

in	*field*	Field specifier.
out	*pId*	Pointer to store the read-only GUID string.

Returns

Returns 0 for success.

6.1.9 Setup Functions

Functions

- int Lgsx_ReaderEnumSetups (LgsxReaderPtr reader, int *pNumSetup)

  Collects Setups in the project and returns the count.
- int Lgsx_ReaderEnumSetupsEx (LgsxReaderPtr reader, PNT3D *pLocation, double range, int *pNumSetup)

  Collects Setups in the project that are within the given range from the given position.
- int Lgsx_ReaderMoveNextSetup (LgsxReaderPtr reader, uint64_t *setupId, CYSETUPINFO *pSetup, LgsxMetadataPtr *pMetadata)

  Advances to the next setup in the active collection and returns ID, info, and metadata.
- int Lgsx_ReaderGetSetupGuid (LgsxReaderPtr reader, uint64_t setupId, char **pGuidPtr)

  Get GUID of the setup for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_ReaderGetImageLayerNames (LgsxReaderPtr reader, wchar_t *pLayers, int maxLen)

  Retrieve the Image Layer names in the active Setup.
- int Lgsx_ReaderGetImageLayerNames2 (LgsxReaderPtr reader, wchar_t **pLayersPtr)

  Retrieve the Image Layer names in the active Setup.
- int Lgsx_ReaderGetPanoImage (LgsxReaderPtr reader, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char **panoImage)

  Get the pano image for the active setup.
- int Lgsx_ReaderGetCubemapImageType (LgsxReaderPtr reader, const wchar_t *pLayer, wchar_t **p↩ ImageType)

  Retrieves the image type string for a cubemap image layer in the active setup.
- int Lgsx_ReaderGetCubemapMaxLevel (LgsxReaderPtr reader, const wchar_t *pLayer, int *pLevel)

  Retrieves the maximum mipmap level available for a cubemap image layer in the active setup.
- int Lgsx_ReaderGetCubemapFaceSize (LgsxReaderPtr reader, const wchar_t *pLayer, int level, int *pWidth, int *pHeight)

  Retrieves the size of a cubemap face for a given layer and mipmap level in the active setup.
- int Lgsx_ReaderGetCubemapMetadata (LgsxReaderPtr reader, const wchar_t *pLayer, LgsxMetadataPtr *pMetadata)

  Retrieves the metadata for a cubemap in the active setup.

- int Lgsx_ReaderGetCubemapFaceMetadata (LgsxReaderPtr reader, const wchar_t ∗pLayer, int face, LgsxMetadataPtr ∗pMetadata)

  Retrieves the metadata for a specific face of a cubemap in the active setup.
- int Lgsx_ReaderGetCubemapImageBytes (LgsxReaderPtr reader, const wchar_t ∗pLayer, int level, int p↩ BufferSizes[6], unsigned char ∗images[6])

  Retrieves the raw image bytes for each face of a cubemap image layer in the active setup.
- int Lgsx_ReaderGetCubeImage (LgsxReaderPtr reader, const wchar_t ∗pLayer, SCyRgdBdyTxf ∗txfFrom↩ Setup, int ∗width, int ∗height, int ∗widthInBytes, ImagePixelType ∗pixelType, unsigned char ∗cubeImages[6])

  Get the cubemap corresponding to the given layer name for the active setup.
- int Lgsx_GetCubemapFaceOrientation (CubemapFaceIndex faceIndex, QUA4D ∗pOrientation)

  Returns the default orientation quaternion for a specific cubemap face.

6.1.9.1 Detailed Description

6.1.9.2 Function Documentation

6.1.9.2.1 Lgsx_GetCubemapFaceOrientation()

```
int Lgsx_GetCubemapFaceOrientation (
            CubemapFaceIndex faceIndex,
            QUA4D * pOrientation)
```

Returns the default orientation quaternion for a specific cubemap face.

Parameters

in	*faceIndex*	Cubemap face index to query.
out	*pOrientation*	Caller-allocated quaternion output for the requested face. Must not be `nullptr`.

Returns

Returns 0 on success.

6.1.9.2.2 Lgsx_ReaderEnumSetups()

```
int Lgsx_ReaderEnumSetups (
            LgsxReaderPtr reader,
            int * pNumSetup)
```

Collects Setups in the project and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumSetup*	Number of collected setups.

Returns

0 for success.

6.1.9.2.3 Lgsx_ReaderEnumSetupsEx()

```
int Lgsx_ReaderEnumSetupsEx (
            LgsxReaderPtr reader,
            PNT3D * pLocation,
            double range,
            int * pNumSetup)
```

Collects Setups in the project that are within the given range from the given position.

Parameters

in	*reader*	Reader handle.
in	*pLocation*	Position of range filter.
in	*range*	Distance from position for range filter.
out	*pNumSetup*	Number of collected setups.

Returns

0 for success.

6.1.9.2.4 Lgsx_ReaderGetCubeImage()

```
int Lgsx_ReaderGetCubeImage (
            LgsxReaderPtr reader,
            const wchar_t * pLayer,
            SCyRgdBdyTxf * txfFromSetup,
            int * width,
            int * height,
            int * widthInBytes,
            ImagePixelType * pixelType,
            unsigned char * cubeImages[6])
```

Get the cubemap corresponding to the given layer name for the active setup.

Parameters

in	*reader*	Reader handle.
in	*pLayer*	Cubemap layer to get, see Lgsx_ReaderGetImageLayerNames().
out	*txfFromSetup*	Offset from the origin of the scan data (typically at the apparent center of laser beam origin) to the location of the cameras that are used to create the panoramic images.
out	*width*	Cubemap image width.
out	*height*	Cubemap image height.
out	*widthInBytes*	Cubemap image byte-width.
out	*pixelType*	Cubemap image pixel type.
out	*cubeImages*	Array of cubemap images, one for each cube face.

Returns

Returns 0 for success.

6.1.9.2.5 Lgsx_ReaderGetCubemapFaceMetadata()

```
int Lgsx_ReaderGetCubemapFaceMetadata (
            LgsxReaderPtr reader,
            const wchar_t * pLayer,
            int face,
            LgsxMetadataPtr * pMetadata)
```

Retrieves the metadata for a specific face of a cubemap in the active setup.

This function returns a pointer to the metadata object associated with the specified face of the cubemap. The metadata contains additional information about the cubemap face, such as properties, tags, or other project-specific data.

Parameters

in	*reader*	Reader handle.
in	*pLayer*	Name of the cubemap layer (e.g. "camera" etc.).
in	*face*	Index of the cubemap face (POS_Y = 0, NEG_Y = 1, NEG_X = 2, POS_X = 3, POS_Z = 4, NEG_Z = 5).
out	*pMetadata*	Pointer to store the metadata object for the cubemap face. The metadata object must be freed using Lgsx_FreeHandle().

Returns

Returns 0 for success.

6.1.9.2.6 Lgsx_ReaderGetCubemapFaceSize()

```
int Lgsx_ReaderGetCubemapFaceSize (
            LgsxReaderPtr reader,
            const wchar_t * pLayer,
            int level,
            int * pWidth,
            int * pHeight)
```

Retrieves the size of a cubemap face for a given layer and mipmap level in the active setup.

This function returns the width and height of a single face of the cubemap image for the specified layer and mipmap level.

Parameters

in	*reader*	Reader handle.
in	*pLayer*	Name of the cubemap layer (e.g. "camera" etc.).
in	*level*	Mipmap level to query (0 is the highest resolution).
out	*pWidth*	Pointer to an integer that will receive the face width.
out	*pHeight*	Pointer to an integer that will receive the face height.

Returns

Returns 0 for success.

6.1.9.2.7 Lgsx_ReaderGetCubemapImageBytes()

```
int Lgsx_ReaderGetCubemapImageBytes (
            LgsxReaderPtr reader,
            const wchar_t * pLayer,
            int level,
            int pBufferSizes[6],
            unsigned char * images[6])
```

Retrieves the raw image bytes for each face of a cubemap image layer in the active setup.

This function fills the provided buffers with the image data for each of the six cubemap faces for the specified layer and mipmap level.

Parameters

in	*reader*	Reader handle.
in	*pLayer*	Name of the cubemap layer (e.g. "camera" etc.).
in	*level*	Mipmap level to query (0 is the highest resolution).
out	*pBufferSizes*	Array of 6 integers specifying the size of each buffer in the images array.
out	*images*	Array of 6 pointers to buffers that will receive the image bytes for each cubemap face.

Returns

Returns 0 for success.

6.1.9.2.8 Lgsx_ReaderGetCubemapImageType()

```
int Lgsx_ReaderGetCubemapImageType (
            LgsxReaderPtr reader,
            const wchar_t * pLayer,
            wchar_t ** pImageType)
```

Retrieves the image type string for a cubemap image layer in the active setup.

This function returns the type of the cubemap image for the specified layer name. The returned string is allocated by the SDK and must be freed using LgsxUtil_FreeMem().

Parameters

in	*reader*	Reader handle.
in	*pLayer*	Name of the cubemap layer (e.g. "camera" etc.).
out	*pImageType*	Pointer to store pointer to the image type string. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success.

6.1.9.2.9 Lgsx_ReaderGetCubemapMaxLevel()

```
int Lgsx_ReaderGetCubemapMaxLevel (
            LgsxReaderPtr reader,
            const wchar_t * pLayer,
            int * pLevel)
```

Retrieves the maximum mipmap level available for a cubemap image layer in the active setup.

This function returns the highest mipmap level for the specified cubemap image layer. The mipmap level determines the resolution of the cubemap images, with level 0 being the highest resolution.

Parameters

in	*reader*	Reader handle.
in	*pLayer*	Name of the cubemap layer (e.g. "camera", etc.).
out	*pLevel*	Pointer to an integer that will receive the maximum mipmap level.

Returns

Returns 0 for success.

6.1.9.2.10 Lgsx_ReaderGetCubemapMetadata()

```
int Lgsx_ReaderGetCubemapMetadata (
            LgsxReaderPtr reader,
            const wchar_t * pLayer,
            LgsxMetadataPtr * pMetadata)
```

Retrieves the metadata for a cubemap in the active setup.

This function returns a pointer to the metadata object associated with the specified cubemap. The metadata contains additional information about the cubemap layer, such as properties, tags, or other project-specific data.

Parameters

in	*reader*	Reader handle.
in	*pLayer*	Name of the cubemap layer (e.g. "camera", etc.).
out	*pMetadata*	Pointer to store the metadata object for the cubemap layer. The metadata object must be freed using Lgsx_FreeHandle().

Returns

Returns 0 for success.

6.1.9.2.11 Lgsx_ReaderGetImageLayerNames()

```
int Lgsx_ReaderGetImageLayerNames (
            LgsxReaderPtr reader,
            wchar_t * pLayers,
            int maxLen)
```

Retrieve the Image Layer names in the active Setup.

See also

Lgsx_ReaderGetImageLayerNames2() is a version that always returns complete data without truncation.

A Setup may contain one or more "layers" of panoramic raster data. This function returns the list of layer names that are present in the active Setup. Layer names are separated by commas.

Layer names include:

- Camera : Color camera imagery.
- HDR : High Dynamic Range imagery.
- IR : Non-visual Infrared temperature data.
- DisplayIR : Infrared temperature imagery. This is a processed version of the IR data that makes it easier to see the temperature differences.
- ScanIntensity : Grayscale intensity data.
- ScanDepth : Non-visual depth data. Each "pixel" of data represents the return distance for that location.

Parameters

in	*reader*	Reader handle.
out	*pLayers*	Comma-separated list of layer names present in Setup object.
in	*maxLen*	Length of the given layer string buffer.

Returns

Returns number of layers (>=0) on success.

6.1.9.2.12 Lgsx_ReaderGetImageLayerNames2()

```
int Lgsx_ReaderGetImageLayerNames2 (
            LgsxReaderPtr reader,
            wchar_t ** pLayersPtr)
```

Retrieve the Image Layer names in the active Setup.

A Setup may contain one or more "layers" of panoramic raster data. This function returns the list of layer names that are present in the active Setup. Layer names are separated by commas.

Layer names include:

- Camera : Color camera imagery.
- HDR : High Dynamic Range imagery.
- IR : Non-visual Infrared temperature data.
- DisplayIR : Infrared temperature imagery. This is a processed version of the IR data that makes it easier to see the temperature differences.
- ScanIntensity : Grayscale intensity data.
- ScanDepth : Non-visual depth data. Each "pixel" of data represents the return distance for that location.

Parameters

in	*reader*	Reader handle.
out	*pLayersPtr*	Pointer to store pointer to comma-separated list of layer names present in Setup object. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns number of layers (>=0) on success.

6.1.9.2.13 Lgsx_ReaderGetPanoImage()

```
int Lgsx_ReaderGetPanoImage (
            LgsxReaderPtr reader,
            int * width,
            int * height,
            int * widthInBytes,
            ImagePixelType * pixelType,
            unsigned char ** panoImage)
```

Get the pano image for the active setup.

Remarks

Lgsx_ReaderGetPanoImage() is provided to support projects created prior to JetStream version 1.5.0. The vast majority of LGSx files will use cubemaps, which can be retrieved via Lgsx_ReaderGetCubeImage().

Parameters

in	*reader*	Reader handle.
out	*width*	Pano image width.
out	*height*	Pano image height.
out	*widthInBytes*	Pano image byte-width.
out	*pixelType*	Pano image pixel type.
out	*panoImage*	Pano image.

Returns

Returns 0 for success.

6.1.9.2.14 Lgsx_ReaderGetSetupGuid()

```
int Lgsx_ReaderGetSetupGuid (
            LgsxReaderPtr reader,
            uint64_t setupId,
            char ** pGuidPtr)
```

Get GUID of the setup for current setup as per Lgsx_ReaderMoveNextSetup() context.

Parameters

in	*reader*	Reader handle.
in	*setupId*	Setup ID to match last setup returned by Lgsx_ReaderMoveNextSetup().
out	*pGuidPtr*	Pointer to store pointer to GUID string. Buffer must be freed using LgsxUtil_FreeMem(). May be an empty string if the source object does not have a GUID.

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.9.2.15 Lgsx_ReaderMoveNextSetup()

```
int Lgsx_ReaderMoveNextSetup (
            LgsxReaderPtr reader,
            uint64_t * setupId,
            CYSETUPINFO * pSetup,
            LgsxMetadataPtr * pMetadata)
```

Advances to the next setup in the active collection and returns ID, info, and metadata.

Parameters

in	*reader*	Reader handle.
out	*setupId*	Setup ID.
out	*pSetup*	Setup info.
out	*pMetadata*	Setup metadata. Metadata must be freed using Lgsx_FreeHandle(). If NULL is given as input, metadata is not returned.

Returns

Returns 0 for success. Returns -1 when end of collection is reached.

6.1.10 SensorInfo Functions

SensorInfo.

Data Structures

- struct SensorInfoHandle

  SensorInfo data handle for accessing the SensorInfo data.
- struct ProcessingInfoHandle

  ProcessingInfo data handle for accessing setup ProcessingInfo data.
- struct ScanHandle

  Scan data handle for accessing the Scan data.

Functions

- int Lgsx_ReaderHasSetupSensorInfo (LgsxReaderPtr reader, bool *pHasInfo)

  Check if setup has SensorInfo for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_ReaderGetSetupSensorInfo (LgsxReaderPtr reader, SensorInfoHandle *pHandle)

  Get SensorInfo of the setup for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_SensorInfoGetGuid (const SensorInfoHandle handle, const char **pGuidPtr)

  Get the GUID of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetDeviceInfoCaptureTime (const SensorInfoHandle handle, uint64_t *pTimestamp)

  Get the capture time of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetManufacturer (const SensorInfoHandle handle, const wchar_t **pManufacturer)

  Get the manufacturer of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetScannerType (const SensorInfoHandle handle, const wchar_t **pScannerType)

  Get the scanner type of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetSerialNumber (const SensorInfoHandle handle, const wchar_t **pSerialNumber)

  Get the serial number of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetArticleNumber (const SensorInfoHandle handle, const wchar_t **pArticleNumber)

  Get the article number of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetHardwareVersion (const SensorInfoHandle handle, const wchar_t **pHardware↩ Version)

  Get the hardware version of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetFirmwareVersion (const SensorInfoHandle handle, const wchar_t **pFirmware↩ Version)

  Get the firmware version of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetDeviceInfoSnapshot (const SensorInfoHandle handle, LgsxMetadataPtr *pSnapshot)

  Get the device info snapshot of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoRelease (SensorInfoHandle *pHandle)

  Release the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_ReaderEnumSetupProcessingInfos (LgsxReaderPtr reader, int *pNumProcessingInfo)

  Enumerate ProcessingInfo entries associated with current setup as per Lgsx_ReaderMoveNextSetup() context and return the count.
- int Lgsx_ReaderGetSetupProcessingInfo (LgsxReaderPtr reader, int pos, ProcessingInfoHandle *pHandle)

  Get ProcessingInfo entry at the specified position for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_ProcessingInfoGetGuid (const ProcessingInfoHandle handle, const char **pGuidPtr)

  Get GUID of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetProcessingTimestamp (const ProcessingInfoHandle handle, uint64_t *p↩ Timestamp)

  Get processing timestamp of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetAppName (const ProcessingInfoHandle handle, const wchar_t **pAppName)

  Get processing app name of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetAppVersion (const ProcessingInfoHandle handle, const wchar_t **pAppVersion)

  Get processing app version of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoHasLibraryVersion (const ProcessingInfoHandle handle, bool *pHasLibraryVersion)

  Check whether ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo() has library version.
- int Lgsx_ProcessingInfoGetLibraryVersion (const ProcessingInfoHandle handle, const wchar_t **pLibrary↩ Version)

  Get processing library version of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetOrderIndex (const ProcessingInfoHandle handle, int *pOrderIndex)

  Get processing order index of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetPipelineInfo (const ProcessingInfoHandle handle, LgsxMetadataPtr *pPipeline↩ Info)

  Get pipeline metadata of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

- int Lgsx_ProcessingInfoRelease (ProcessingInfoHandle *pHandle)

  Release ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ReaderEnumSetupScans (LgsxReaderPtr reader, int *pNumScans)

  Collects Scans for the current Setup and returns the count.
- int Lgsx_ReaderGetSetupScan (LgsxReaderPtr reader, int pos, ScanHandle *pHandle)

  Read Scan data at the given position in the active collection.
- int Lgsx_ScanRelease (ScanHandle *pHandle)

  Release the Scan data pointer returned by Lgsx_ReaderGetSetupScan().
- int Lgsx_ScanGetGuid (const ScanHandle handle, const char **pGuid)

  Get GUID of the Scan.
- int Lgsx_ScanGetScanType (const ScanHandle handle, int *pScanType)

  Get scan type of the Scan.
- int Lgsx_ScanGetScanRole (const ScanHandle handle, int *pScanRole)

  Get scan role of the Scan.
- int Lgsx_ScanGetPointCount (const ScanHandle handle, uint64_t *pPointCount)

  Get point count of the Scan.
- int Lgsx_ScanGetCreationTimestamp (const ScanHandle handle, uint64_t *pTimestamp)

  Get creation timestamp of the Scan.
- int Lgsx_ScanGetModificationTimestamp (const ScanHandle handle, uint64_t *pTimestamp)

  Get modification timestamp of the Scan.
- int Lgsx_ScanHasWindow (const ScanHandle handle, bool *pHasWindow)

  Check whether the Scan has a scan window.
- int Lgsx_ScanGetWindowMinAzimuth (const ScanHandle handle, double *pMinAzimuth)

  Get minimum azimuth angle of the scan window (radians).
- int Lgsx_ScanGetWindowMinElevation (const ScanHandle handle, double *pMinElevation)

  Get minimum elevation angle of the scan window (radians).
- int Lgsx_ScanGetWindowMaxAzimuth (const ScanHandle handle, double *pMaxAzimuth)

  Get maximum azimuth angle of the scan window (radians).
- int Lgsx_ScanGetWindowMaxElevation (const ScanHandle handle, double *pMaxElevation)

  Get maximum elevation angle of the scan window (radians).
- int Lgsx_ScanGetCaptureTimestamp (const ScanHandle handle, uint64_t *pTimestamp)

  Get capture timestamp of the Scan.
- int Lgsx_ScanGetScanDensity (const ScanHandle handle, double *pDensity)

  Get scan density of the Scan.
- int Lgsx_ScanGetCaptureInfoSnapshot (const ScanHandle handle, LgsxMetadataPtr *pSnapshot)

  Get capture info snapshot metadata of the Scan.
- int Lgsx_ReaderGetCurrentSetupName (LgsxReaderPtr reader, wchar_t **pNamePtr)

  Get the full (untruncated) name of the most recently returned Setup.
- int Lgsx_ReaderEndSetups (LgsxReaderPtr reader)

  Frees the active collection.
- int Lgsx_ReaderHasRunSensorInfo (LgsxReaderPtr reader, bool *pHasInfo)

  Check if current run has SensorInfo as per Lgsx_ReaderMoveNextRun() context.
- int Lgsx_ReaderGetRunSensorInfo (LgsxReaderPtr reader, SensorInfoHandle *pHandle)

  Get SensorInfo of the current run as per Lgsx_ReaderMoveNextRun() context.
- int Lgsx_ReaderEnumRunProcessingInfos (LgsxReaderPtr reader, int *pNumProcessingInfo)

  Enumerate ProcessingInfo entries associated with current run as per Lgsx_ReaderMoveNextRun() context and return the count.
- int Lgsx_ReaderGetRunProcessingInfo (LgsxReaderPtr reader, int pos, ProcessingInfoHandle *pHandle)

  Get ProcessingInfo entry at the specified position for current run as per Lgsx_ReaderMoveNextRun() context.
- int Lgsx_ReaderEnumRunScans (LgsxReaderPtr reader, int *pNumScans)

  Collects Scans for the current Run's Setup and returns the count.
- int Lgsx_ReaderGetRunScan (LgsxReaderPtr reader, int pos, ScanHandle *pHandle)

  Read Scan data at the given position in the active run-scoped collection.

6.1.10.1 Detailed Description

SensorInfo.

6.1.10.2 Function Documentation

6.1.10.2.1 Lgsx_ProcessingInfoGetAppName()

```
int Lgsx_ProcessingInfoGetAppName (
            const ProcessingInfoHandle handle,
            const wchar_t ** pAppName)
```

Get processing app name of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pAppName*	Pointer to store app name buffer. Buffer is read-only, bound to the lifetime of ProcessingInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.2 Lgsx_ProcessingInfoGetAppVersion()

```
int Lgsx_ProcessingInfoGetAppVersion (
            const ProcessingInfoHandle handle,
            const wchar_t ** pAppVersion)
```

Get processing app version of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pAppVersion*	Pointer to store app version buffer. Buffer is read-only, bound to the lifetime of ProcessingInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.3 Lgsx_ProcessingInfoGetGuid()

```
int Lgsx_ProcessingInfoGetGuid (
            const ProcessingInfoHandle handle,
            const char ** pGuidPtr)
```

Get GUID of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pGuidPtr*	Pointer to store GUID buffer of the ProcessingInfo. Buffer is read-only, bound to the lifetime of ProcessingInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.4 Lgsx_ProcessingInfoGetLibraryVersion()

```
int Lgsx_ProcessingInfoGetLibraryVersion (
            const ProcessingInfoHandle handle,
            const wchar_t ** pLibraryVersion)
```

Get processing library version of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pLibraryVersion*	Pointer to store library version buffer. Buffer is read-only, bound to the lifetime of ProcessingInfo data. Must not be freed.

Returns

Returns 0 for success. Returns -1 when library version is not available.

6.1.10.2.5 Lgsx_ProcessingInfoGetOrderIndex()

```
int Lgsx_ProcessingInfoGetOrderIndex (
            const ProcessingInfoHandle handle,
            int * pOrderIndex)
```

Get processing order index of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pOrderIndex*	Pointer to store order index.

Returns

Returns 0 for success.

6.1.10.2.6 Lgsx_ProcessingInfoGetPipelineInfo()

```
int Lgsx_ProcessingInfoGetPipelineInfo (
            const ProcessingInfoHandle handle,
            LgsxMetadataPtr * pPipelineInfo)
```

Get pipeline metadata of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pPipelineInfo*	Pointer to store pipeline metadata. pPipelineInfo is an unstructured key-value container holding application-defined processing pipeline settings. Metadata is bound to the lifetime of ProcessingInfoHandle data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.7 Lgsx_ProcessingInfoGetProcessingTimestamp()

```
int Lgsx_ProcessingInfoGetProcessingTimestamp (
            const ProcessingInfoHandle handle,
            uint64_t * pTimestamp)
```

Get processing timestamp of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pTimestamp*	Pointer to store processing timestamp.

Returns

Returns 0 for success.

6.1.10.2.8 Lgsx_ProcessingInfoHasLibraryVersion()

```
int Lgsx_ProcessingInfoHasLibraryVersion (
            const ProcessingInfoHandle handle,
            bool * pHasLibraryVersion)
```

Check whether ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo() has library version.

Parameters

in	*handle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().
out	*pHasLibraryVersion*	True when library version is available.

Returns

Returns 0 for success.

6.1.10.2.9 Lgsx_ProcessingInfoRelease()

```
int Lgsx_ProcessingInfoRelease (
            ProcessingInfoHandle * pHandle)
```

Release ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().

Parameters

in	*pHandle*	Pointer to ProcessingInfo data returned by Lgsx_ReaderGetSetupProcessingInfo().

Returns

Returns 0 for success.

6.1.10.2.10 Lgsx_ReaderEndSetups()

```
int Lgsx_ReaderEndSetups (
            LgsxReaderPtr reader)
```

Frees the active collection.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.10.2.11 Lgsx_ReaderEnumRunProcessingInfos()

```
int Lgsx_ReaderEnumRunProcessingInfos (
            LgsxReaderPtr reader,
            int * pNumProcessingInfo)
```

Enumerate ProcessingInfo entries associated with current run as per Lgsx_ReaderMoveNextRun() context and return the count.

Parameters

in	*reader*	Reader handle.
out	*pNumProcessingInfo*	Pointer to store number of ProcessingInfo entries.

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.12 Lgsx_ReaderEnumRunScans()

```
int Lgsx_ReaderEnumRunScans (
            LgsxReaderPtr reader,
            int * pNumScans)
```

Collects Scans for the current Run's Setup and returns the count.

Must be called inside a Lgsx_ReaderMoveNextRun() loop.

Parameters

in	*reader*	Reader handle.
out	*pNumScans*	Pointer to store the number of Scans.

Returns

Returns 0 for success.

6.1.10.2.13 Lgsx_ReaderEnumSetupProcessingInfos()

```
int Lgsx_ReaderEnumSetupProcessingInfos (
            LgsxReaderPtr reader,
            int * pNumProcessingInfo)
```

Enumerate ProcessingInfo entries associated with current setup as per Lgsx_ReaderMoveNextSetup() context and return the count.

Parameters

in	*reader*	Reader handle.
out	*pNumProcessingInfo*	Pointer to store number of ProcessingInfo entries.

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.14 Lgsx_ReaderEnumSetupScans()

```
int Lgsx_ReaderEnumSetupScans (
            LgsxReaderPtr reader,
            int * pNumScans)
```

Collects Scans for the current Setup and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumScans*	Pointer to store the number of Scans.

Returns

Returns 0 for success.

6.1.10.2.15 Lgsx_ReaderGetCurrentSetupName()

```
int Lgsx_ReaderGetCurrentSetupName (
            LgsxReaderPtr reader,
            wchar_t ** pNamePtr)
```

Get the full (untruncated) name of the most recently returned Setup.

See also

Lgsx_ReaderMoveNextSetup() for the enumeration context.

The `name` field in CYSETUPINFO may be truncated. Use this function to get the complete name.

Parameters

in	*reader*	Reader handle.
out	*pNamePtr*	Pointer to store pointer to the name string. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success. Returns -2 if called outside an active Lgsx_ReaderMoveNextSetup() loop.

6.1.10.2.16 Lgsx_ReaderGetRunProcessingInfo()

```
int Lgsx_ReaderGetRunProcessingInfo (
            LgsxReaderPtr reader,
            int pos,
            ProcessingInfoHandle * pHandle)
```

Get ProcessingInfo entry at the specified position for current run as per Lgsx_ReaderMoveNextRun() context.

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in run ProcessingInfo collection.
out	*pHandle*	Pointer to store ProcessingInfo handle. Buffer must be freed using Lgsx_ProcessingInfoRelease().

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.17 Lgsx_ReaderGetRunScan()

```
int Lgsx_ReaderGetRunScan (
            LgsxReaderPtr reader,
            int pos,
            ScanHandle * pHandle)
```

Read Scan data at the given position in the active run-scoped collection.

Active collection is set by Lgsx_ReaderEnumRunScans().

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in the active collection.
out	*pHandle*	Pointer to store pointer to Scan data. The pointer has to be released using Lgsx_ScanRelease().

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.10.2.18 Lgsx_ReaderGetRunSensorInfo()

```
int Lgsx_ReaderGetRunSensorInfo (
            LgsxReaderPtr reader,
            SensorInfoHandle * pHandle)
```

Get SensorInfo of the current run as per Lgsx_ReaderMoveNextRun() context.

Parameters

in	*reader*	Reader handle.
out	*pHandle*	Pointer to store pointer to SensorInfo data. Buffer must be freed using Lgsx_SensorInfoRelease().

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.19 Lgsx_ReaderGetSetupProcessingInfo()

```
int Lgsx_ReaderGetSetupProcessingInfo (
            LgsxReaderPtr reader,
            int pos,
            ProcessingInfoHandle * pHandle)
```

Get ProcessingInfo entry at the specified position for current setup as per Lgsx_ReaderMoveNextSetup() context.

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in setup ProcessingInfo collection.
out	*pHandle*	Pointer to store ProcessingInfo handle. Buffer must be freed using Lgsx_ProcessingInfoRelease().

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.20 Lgsx_ReaderGetSetupScan()

```
int Lgsx_ReaderGetSetupScan (
            LgsxReaderPtr reader,
            int pos,
            ScanHandle * pHandle)
```

Read Scan data at the given position in the active collection.

Active collection is set by Lgsx_ReaderEnumSetupScans().

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in the active collection.
out	*pHandle*	Pointer to store pointer to Scan data. The pointer has to be released using Lgsx_ScanRelease().

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.10.2.21 Lgsx_ReaderGetSetupSensorInfo()

```
int Lgsx_ReaderGetSetupSensorInfo (
            LgsxReaderPtr reader,
            SensorInfoHandle * pHandle)
```

Get SensorInfo of the setup for current setup as per Lgsx_ReaderMoveNextSetup() context.

Parameters

in	*reader*	Reader handle.
out	*pHandle*	Pointer to store pointer to SensorInfo data. Buffer must be freed using Lgsx_SensorInfoRelease().

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.22 Lgsx_ReaderHasRunSensorInfo()

```
int Lgsx_ReaderHasRunSensorInfo (
            LgsxReaderPtr reader,
            bool * pHasInfo)
```

Check if current run has SensorInfo as per Lgsx_ReaderMoveNextRun() context.

Parameters

in	*reader*	Reader handle.
out	*pHasInfo*	True if SensorInfo is available for the run, false otherwise.

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.23 Lgsx_ReaderHasSetupSensorInfo()

```
int Lgsx_ReaderHasSetupSensorInfo (
            LgsxReaderPtr reader,
            bool * pHasInfo)
```

Check if setup has SensorInfo for current setup as per Lgsx_ReaderMoveNextSetup() context.

Parameters

in	*reader*	Reader handle.
out	*pHasInfo*	True if SensorInfo is available for the setup, false otherwise.

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.10.2.24 Lgsx_ScanGetCaptureInfoSnapshot()

```
int Lgsx_ScanGetCaptureInfoSnapshot (
            const ScanHandle handle,
            LgsxMetadataPtr * pSnapshot)
```

Get capture info snapshot metadata of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pSnapshot*	Pointer to store the metadata pointer. Metadata is bound to the lifetime of ScanHandle data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.25 Lgsx_ScanGetCaptureTimestamp()

```
int Lgsx_ScanGetCaptureTimestamp (
            const ScanHandle handle,
            uint64_t * pTimestamp)
```

Get capture timestamp of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pTimestamp*	Pointer to store the capture timestamp.

Returns

Returns 0 for success.

6.1.10.2.26 Lgsx_ScanGetCreationTimestamp()

```
int Lgsx_ScanGetCreationTimestamp (
            const ScanHandle handle,
            uint64_t * pTimestamp)
```

Get creation timestamp of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pTimestamp*	Pointer to store the creation timestamp.

Returns

Returns 0 for success.

6.1.10.2.27 Lgsx_ScanGetGuid()

```
int Lgsx_ScanGetGuid (
            const ScanHandle handle,
            const char ** pGuid)
```

Get GUID of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pGuid*	Pointer to store the GUID string of the Scan.

Returns

Returns 0 for success.

6.1.10.2.28 Lgsx_ScanGetModificationTimestamp()

```
int Lgsx_ScanGetModificationTimestamp (
            const ScanHandle handle,
            uint64_t * pTimestamp)
```

Get modification timestamp of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pTimestamp*	Pointer to store the modification timestamp.

Returns

Returns 0 for success.

6.1.10.2.29 Lgsx_ScanGetPointCount()

```
int Lgsx_ScanGetPointCount (
            const ScanHandle handle,
            uint64_t * pPointCount)
```

Get point count of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pPointCount*	Pointer to store the point count.

Returns

Returns 0 for success.

6.1.10.2.30 Lgsx_ScanGetScanDensity()

```
int Lgsx_ScanGetScanDensity (
            const ScanHandle handle,
            double * pDensity)
```

Get scan density of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pDensity*	Pointer to store the scan density.

Returns

Returns 0 for success.

6.1.10.2.31 Lgsx_ScanGetScanRole()

```
int Lgsx_ScanGetScanRole (
            const ScanHandle handle,
            int * pScanRole)
```

Get scan role of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pScanRole*	Pointer to store the scan role (0=PRIMARY, 1=SUPPLEMENTARY).

Returns

Returns 0 for success.

6.1.10.2.32 Lgsx_ScanGetScanType()

```
int Lgsx_ScanGetScanType (
            const ScanHandle handle,
            int * pScanType)
```

Get scan type of the Scan.

Parameters

in	*handle*	Scan handle.
out	*pScanType*	Pointer to store the scan type (0=FULL_DOME, 1=DETAIL, 2=TARGET).

Returns

Returns 0 for success.

6.1.10.2.33 Lgsx_ScanGetWindowMaxAzimuth()

```
int Lgsx_ScanGetWindowMaxAzimuth (
            const ScanHandle handle,
            double * pMaxAzimuth)
```

Get maximum azimuth angle of the scan window (radians).

Parameters

in	*handle*	Scan handle.
out	*pMaxAzimuth*	Pointer to store the maximum azimuth.

Returns

Returns 0 for success. Returns -4 when no scan window is present.

6.1.10.2.34 Lgsx_ScanGetWindowMaxElevation()

```
int Lgsx_ScanGetWindowMaxElevation (
            const ScanHandle handle,
            double * pMaxElevation)
```

Get maximum elevation angle of the scan window (radians).

Parameters

in	*handle*	Scan handle.
out	*pMaxElevation*	Pointer to store the maximum elevation.

Returns

Returns 0 for success. Returns -4 when no scan window is present.

6.1.10.2.35 Lgsx_ScanGetWindowMinAzimuth()

```
int Lgsx_ScanGetWindowMinAzimuth (
            const ScanHandle handle,
            double * pMinAzimuth)
```

Get minimum azimuth angle of the scan window (radians).

Parameters

in	*handle*	Scan handle.
out	*pMinAzimuth*	Pointer to store the minimum azimuth.

Returns

Returns 0 for success. Returns -4 when no scan window is present.

6.1.10.2.36 Lgsx_ScanGetWindowMinElevation()

```
int Lgsx_ScanGetWindowMinElevation (
            const ScanHandle handle,
            double * pMinElevation)
```

Get minimum elevation angle of the scan window (radians).

Parameters

in	*handle*	Scan handle.
out	*pMinElevation*	Pointer to store the minimum elevation.

Returns

Returns 0 for success. Returns -4 when no scan window is present.

6.1.10.2.37 Lgsx_ScanHasWindow()

```
int Lgsx_ScanHasWindow (
            const ScanHandle handle,
            bool * pHasWindow)
```

Check whether the Scan has a scan window.

Parameters

in	*handle*	Scan handle.
out	*pHasWindow*	True when a scan window is present.

Returns

Returns 0 for success.

6.1.10.2.38 Lgsx_ScanRelease()

```
int Lgsx_ScanRelease (
            ScanHandle * pHandle)
```

Release the Scan data pointer returned by Lgsx_ReaderGetSetupScan().

Parameters

in	*pHandle*	Pointer to Scan data.

Returns

Returns 0 for success.

6.1.10.2.39 Lgsx_SensorInfoGetArticleNumber()

```
int Lgsx_SensorInfoGetArticleNumber (
            const SensorInfoHandle handle,
            const wchar_t ** pArticleNumber)
```

Get the article number of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pArticleNumber*	Pointer to store article number buffer of the SensorInfo. Buffer is read-only, bound to the lifetime of the SensorInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.40 Lgsx_SensorInfoGetDeviceInfoCaptureTime()

```
int Lgsx_SensorInfoGetDeviceInfoCaptureTime (
            const SensorInfoHandle handle,
            uint64_t * pTimestamp)
```

Get the capture time of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pTimestamp*	Pointer to store capture time buffer of the SensorInfo.

Returns

Returns 0 for success.

6.1.10.2.41 Lgsx_SensorInfoGetDeviceInfoSnapshot()

```
int Lgsx_SensorInfoGetDeviceInfoSnapshot (
            const SensorInfoHandle handle,
            LgsxMetadataPtr * pSnapshot)
```

Get the device info snapshot of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

The device info snapshot is an unstructured key-value metadata container holding additional device-specific properties captured at the time the sensor info was recorded (e.g. calibration parameters, device settings). Its contents are device-dependent.

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pSnapshot*	Pointer to store snapshot buffer of the SensorInfo. Snapshot is bound to the lifetime of the SensorInfoHandle data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.42 Lgsx_SensorInfoGetFirmwareVersion()

```
int Lgsx_SensorInfoGetFirmwareVersion (
            const SensorInfoHandle handle,
            const wchar_t ** pFirmwareVersion)
```

Get the firmware version of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pFirmwareVersion*	Pointer to store firmware version buffer of the SensorInfo. Buffer is read-only, bound to the lifetime of the SensorInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.43 Lgsx_SensorInfoGetGuid()

```
int Lgsx_SensorInfoGetGuid (
            const SensorInfoHandle handle,
            const char ** pGuidPtr)
```

Get the GUID of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pGuidPtr*	Pointer to store GUID buffer of the SensorInfo. Buffer is read-only, bound to the lifetime of the SensorInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.44 Lgsx_SensorInfoGetHardwareVersion()

```
int Lgsx_SensorInfoGetHardwareVersion (
            const SensorInfoHandle handle,
            const wchar_t ** pHardwareVersion)
```

Get the hardware version of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pHardwareVersion*	Pointer to store hardware version buffer of the SensorInfo. Buffer is read-only, bound to the lifetime of the SensorInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.45 Lgsx_SensorInfoGetManufacturer()

```
int Lgsx_SensorInfoGetManufacturer (
            const SensorInfoHandle handle,
            const wchar_t ** pManufacturer)
```

Get the manufacturer of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pManufacturer*	Pointer to store manufacturer buffer of the SensorInfo. Buffer is read-only, bound to the lifetime of the SensorInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.46 Lgsx_SensorInfoGetScannerType()

```
int Lgsx_SensorInfoGetScannerType (
            const SensorInfoHandle handle,
            const wchar_t ** pScannerType)
```

Get the scanner type of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pScannerType*	Pointer to store scanner type buffer of the SensorInfo. Buffer is read-only, bound to the lifetime of the SensorInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.47 Lgsx_SensorInfoGetSerialNumber()

```
int Lgsx_SensorInfoGetSerialNumber (
            const SensorInfoHandle handle,
            const wchar_t ** pSerialNumber)
```

Get the serial number of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*handle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().
out	*pSerialNumber*	Pointer to store serial number buffer of the SensorInfo. Buffer is read-only, bound to the lifetime of the SensorInfo data. Must not be freed.

Returns

Returns 0 for success.

6.1.10.2.48 Lgsx_SensorInfoRelease()

```
int Lgsx_SensorInfoRelease (
            SensorInfoHandle * pHandle)
```

Release the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().

Parameters

in	*pHandle*	Pointer to SensorInfo data returned by Lgsx_ReaderGetSetupSensorInfo().

Returns

Returns 0 for success.

6.1.11 Target Functions

Functions

- int Lgsx_ReaderEnumTarget (LgsxReaderPtr reader, int ∗pNumTarget)

  Collects Targets in the project and returns the count.
- int Lgsx_ReaderEnumTargetOfSetup (LgsxReaderPtr reader, uint64_t setupId, int ∗pNumTarget)

  Collects Targets in the given Setup and returns the count.
- int Lgsx_ReaderMoveNextTarget (LgsxReaderPtr reader, uint64_t ∗targetId, CYTARGETINFO ∗pTarget, LgsxMetadataPtr ∗pMetadata)

  Advance to the next target in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderGetCurrentTargetLabel (LgsxReaderPtr reader, wchar_t ∗∗pLabelPtr)

  Get the full (untruncated) label of the most recently returned Target.
- int Lgsx_ReaderEndTargets (LgsxReaderPtr reader)

  Frees the active Targets collection.

6.1.11.1 Detailed Description

6.1.11.2 Function Documentation

6.1.11.2.1 Lgsx_ReaderEndTargets()

```
int Lgsx_ReaderEndTargets (
            LgsxReaderPtr reader)
```

Frees the active Targets collection.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.11.2.2 Lgsx_ReaderEnumTarget()

```
int Lgsx_ReaderEnumTarget (
            LgsxReaderPtr reader,
            int * pNumTarget)
```

Collects Targets in the project and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumTarget*	Number of collected Targets.

Returns

0 for success.

6.1.11.2.3 Lgsx_ReaderEnumTargetOfSetup()

```
int Lgsx_ReaderEnumTargetOfSetup (
            LgsxReaderPtr reader,
            uint64_t setupId,
            int * pNumTarget)
```

Collects Targets in the given Setup and returns the count.

Parameters

in	*reader*	Reader handle.
in	*setupId*	Parent Setup ID.
out	*pNumTarget*	Number of collected targets.

Returns

0 for success.

6.1.11.2.4 Lgsx_ReaderGetCurrentTargetLabel()

```
int Lgsx_ReaderGetCurrentTargetLabel (
            LgsxReaderPtr reader,
            wchar_t ** pLabelPtr)
```

Get the full (untruncated) label of the most recently returned Target.

See also

Lgsx_ReaderMoveNextTarget() for the enumeration context.

The `label` field in CYTARGETINFO may be truncated. Use this function to get the complete label.

Parameters

in	*reader*	Reader handle.
out	*pLabelPtr*	Pointer to store pointer to the label string. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success. Returns -2 if called outside an active Lgsx_ReaderMoveNextTarget() loop.

6.1.11.2.5 Lgsx_ReaderMoveNextTarget()

```
int Lgsx_ReaderMoveNextTarget (
            LgsxReaderPtr reader,
            uint64_t * targetId,
            CYTARGETINFO * pTarget,
            LgsxMetadataPtr * pMetadata)
```

Advance to the next target in the active collection and return the ID, info, and metadata.

Parameters

in	*reader*	Reader handle.
out	*targetId*	Target ID.
out	*pTarget*	Target info.
out	*pMetadata*	Target metadata. Metadata must be freed using Lgsx_FreeHandle().

Returns

Returns 0 for success. Returns -1 when end of collection is reached.

6.1.12 Run Functions

Functions

- int Lgsx_ReaderEnumRuns (LgsxReaderPtr reader, int *pNumRun)

  Collects Runs in the project and returns the count.
- int Lgsx_ReaderMoveNextRun (LgsxReaderPtr reader, uint64_t *runId, CYTRAJECTORYINFO *info, int *nVertex, CYTRAJECTORYVERTEX **ppPath, LgsxMetadataPtr *pMetadata)

  Advance to the next Run in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderMoveNextRun2 (LgsxReaderPtr reader, uint64_t *runId, CYTRAJECTORYINFO *info, int *nVertex, CYTRAJECTORYVERTEX **ppPath, LgsxMetadataPtr *pMetadata, wchar_t **pJobNamePtr, wchar_t **pRunNamePtr)

  Advance to the next Run and return full (untruncated) job and run names alongside the complete Run info.
- int Lgsx_ReaderEndRuns (LgsxReaderPtr reader)

  Frees the active Run collection.
- int Lgsx_ReaderRunGetGuid (LgsxReaderPtr reader, char **pGuidPtr)

  Gets the Run GUID for the current Run in the enumeration (Lgsx_ReaderEnumRuns).
- int Lgsx_ReaderGetLUTImage (LgsxReaderPtr reader, wchar_t *typeName, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the current run context.
- int Lgsx_ReaderGetLUTImage2 (LgsxReaderPtr reader, wchar_t **pTypeNamePtr, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the desired type, without buffer truncation.
- int Lgsx_ReaderGetLUTImageOfSetup (LgsxReaderPtr reader, uint64_t setupId, wchar_t *typeName, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the given Setup.
- int Lgsx_ReaderGetLUTImageOfSetup2 (LgsxReaderPtr reader, uint64_t setupId, wchar_t **pTypeNamePtr, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the given Setup, without buffer truncation.
- int Lgsx_ReaderEnumSetupCamera (LgsxReaderPtr reader, int *pNumSetupCamera)

  Collects Setup Cameras for the current setup and returns the count.

- int Lgsx_ReaderEnumSetupCameraOfSetup (LgsxReaderPtr reader, uint64_t setupId, int *pNumSetupCamera)

  Collects Setup Cameras for the given Setup returns the count.
- int Lgsx_ReaderEndSetupCameras (LgsxReaderPtr reader)

  Frees the active Setup Cameras collection.
- int Lgsx_ReaderMoveNextSetupCamera (LgsxReaderPtr reader, uint64_t *pSetupCameraId, CYSETUPCAMERAINFO *pSetupCamera, LgsxMetadataPtr *pMetadata)

  Advance to the next Setup Camera in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderGetCurrentSetupCameraName (LgsxReaderPtr reader, wchar_t **pNamePtr)

  Get the full (untruncated) name of the most recently returned Setup Camera.
- int Lgsx_ReaderGetSetupCameraImage (LgsxReaderPtr reader, wchar_t *typeName, int *nBytes, void **blob, int *nThumbBytes, void **thumbBlob)

  Get the Setup Camera image and thumbnail image corresponding to the given layer name for the active Setup Camera.
- int Lgsx_ReaderGetSetupCameraImage2 (LgsxReaderPtr reader, wchar_t **pTypeNamePtr, int *nBytes, void **blob, int *nThumbBytes, void **thumbBlob)

  Get the Setup Camera image and thumbnail image for the active Setup Camera, without buffer truncation.

6.1.12.1 Detailed Description

6.1.12.2 Function Documentation

6.1.12.2.1 Lgsx_ReaderEndRuns()

```
int Lgsx_ReaderEndRuns (
            LgsxReaderPtr reader)
```

Frees the active Run collection.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.12.2.2 Lgsx_ReaderEndSetupCameras()

```
int Lgsx_ReaderEndSetupCameras (
            LgsxReaderPtr reader)
```

Frees the active Setup Cameras collection.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.12.2.3 Lgsx_ReaderEnumRuns()

```
int Lgsx_ReaderEnumRuns (
            LgsxReaderPtr reader,
            int * pNumRun)
```

Collects Runs in the project and returns the count.

Remarks

Runs are also known as "Trajectories", "Tracks" or "Walks" to some integrators.

Runs are generated by kinematic scanners.

Parameters

in	*reader*	Reader handle.
out	*pNumRun*	Number of collected Runs.

Returns

0 for success.

6.1.12.2.4 Lgsx_ReaderEnumSetupCamera()

```
int Lgsx_ReaderEnumSetupCamera (
            LgsxReaderPtr reader,
            int * pNumSetupCamera)
```

Collects Setup Cameras for the current setup and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumSetupCamera*	Number of collected Setup Cameras.

Returns

0 for success.

6.1.12.2.5 Lgsx_ReaderEnumSetupCameraOfSetup()

```
int Lgsx_ReaderEnumSetupCameraOfSetup (
            LgsxReaderPtr reader,
            uint64_t setupId,
            int * pNumSetupCamera)
```

Collects Setup Cameras for the given Setup returns the count.

Parameters

in	*reader*	Reader handle.
in	*setupId*	Parent Setup ID.
out	*pNumSetupCamera*	Number of collected Setup Cameras.

Returns

0 for success.

6.1.12.2.6 Lgsx_ReaderGetCurrentSetupCameraName()

```
int Lgsx_ReaderGetCurrentSetupCameraName (
            LgsxReaderPtr reader,
            wchar_t ** pNamePtr)
```

Get the full (untruncated) name of the most recently returned Setup Camera.

See also

Lgsx_ReaderMoveNextSetupCamera() for the enumeration context.

The `name` field in CYSETUPCAMERAINFO may be truncated. Use this function to get the complete name.

Parameters

in	*reader*	Reader handle.
out	*pNamePtr*	Pointer to store pointer to the name string. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success. Returns -2 if called outside an active Lgsx_ReaderMoveNextSetupCamera() loop.

6.1.12.2.7 Lgsx_ReaderGetLUTImage()

```
int Lgsx_ReaderGetLUTImage (
            LgsxReaderPtr reader,
            wchar_t * typeName,
            int * nBytes,
            void ** lut)
```

Gets the Setup Camera LUT (Lookup Table) for the current run context.

See also

Lgsx_ReaderGetLUTImage2() for a version that returns the type name without truncation.

Parameters

in	*reader*	Reader handle.
out	*typeName*	Returned lookup table type name. Pre-allocate at least 40 characters. May be truncated; use Lgsx_ReaderGetLUTImage2() to avoid truncation.
out	*nBytes*	Number of bytes in lookup table.
out	*lut*	Lookup table.

Returns

0 for success.

6.1.12.2.8 Lgsx_ReaderGetLUTImage2()

```
int Lgsx_ReaderGetLUTImage2 (
            LgsxReaderPtr reader,
            wchar_t ** pTypeNamePtr,
            int * nBytes,
            void ** lut)
```

Gets the Setup Camera LUT (Lookup Table) for the desired type, without buffer truncation.

See also

Lgsx_ReaderGetLUTImage() for the truncating alternative.

Parameters

in	*reader*	Reader handle.
out	*pTypeNamePtr*	Pointer to store pointer to the type name string. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.
out	*nBytes*	Number of bytes in lookup table.
out	*lut*	Lookup table.

Returns

0 for success.

6.1.12.2.9 Lgsx_ReaderGetLUTImageOfSetup()

```
int Lgsx_ReaderGetLUTImageOfSetup (
            LgsxReaderPtr reader,
            uint64_t setupId,
            wchar_t * typeName,
            int * nBytes,
            void ** lut)
```

Gets the Setup Camera LUT (Lookup Table) for the given Setup.

See also

Lgsx_ReaderGetLUTImageOfSetup2() for a version that returns the type name without truncation.

Parameters

in	*reader*	Reader handle.
in	*setupId*	Setup that owns the lookup table.
out	*typeName*	Returned lookup table type name. Pre-allocate at least 40 characters. May be truncated; use Lgsx_ReaderGetLUTImageOfSetup2() to avoid truncation.
out	*nBytes*	Number of bytes in lookup table.
out	*lut*	Lookup table.

Returns

0 for success.

6.1.12.2.10 Lgsx_ReaderGetLUTImageOfSetup2()

```
int Lgsx_ReaderGetLUTImageOfSetup2 (
            LgsxReaderPtr reader,
            uint64_t setupId,
            wchar_t ** pTypeNamePtr,
            int * nBytes,
            void ** lut)
```

Gets the Setup Camera LUT (Lookup Table) for the given Setup, without buffer truncation.

See also

Lgsx_ReaderGetLUTImageOfSetup() for the truncating alternative.

Parameters

in	*reader*	Reader handle.
in	*setupId*	Setup that owns the lookup table.
out	*pTypeNamePtr*	Pointer to store pointer to the type name string. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.
out	*nBytes*	Number of bytes in lookup table.
out	*lut*	Lookup table.

Returns

0 for success.

6.1.12.2.11 Lgsx_ReaderGetSetupCameraImage()

```
int Lgsx_ReaderGetSetupCameraImage (
            LgsxReaderPtr reader,
            wchar_t * typeName,
            int * nBytes,
            void ** blob,
            int * nThumbBytes,
            void ** thumbBlob)
```

Get the Setup Camera image and thumbnail image corresponding to the given layer name for the active Setup Camera.

Parameters

in	*reader*	Reader handle.
in	*typeName*	Lookup table type.
out	*nBytes*	Number of bytes in Setup Camera image.
out	*blob*	Setup Camera image.
out	*nThumbBytes*	Number of bytes in Setup Camera thumbnail image.
out	*thumbBlob*	Setup Camera thumbnail image.

Returns

Returns 0 for success.

6.1.12.2.12 Lgsx_ReaderGetSetupCameraImage2()

```
int Lgsx_ReaderGetSetupCameraImage2 (
            LgsxReaderPtr reader,
            wchar_t ** pTypeNamePtr,
            int * nBytes,
            void ** blob,
            int * nThumbBytes,
            void ** thumbBlob)
```

Get the Setup Camera image and thumbnail image for the active Setup Camera, without buffer truncation.

See also

Lgsx_ReaderGetSetupCameraImage() for the truncating alternative.

Parameters

in	*reader*	Reader handle.
out	*pTypeNamePtr*	Pointer to store pointer to the image type name string. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.
out	*nBytes*	Number of bytes in Setup Camera image.
out	*blob*	Setup Camera image.
out	*nThumbBytes*	Number of bytes in Setup Camera thumbnail image.
out	*thumbBlob*	Setup Camera thumbnail image.

Returns

Returns 0 for success.

6.1.12.2.13 Lgsx_ReaderMoveNextRun()

```
int Lgsx_ReaderMoveNextRun (
            LgsxReaderPtr reader,
            uint64_t * runId,
            CYTRAJECTORYINFO * info,
            int * nVertex,
            CYTRAJECTORYVERTEX ** ppPath,
            LgsxMetadataPtr * pMetadata)
```

Advance to the next Run in the active collection and return the ID, info, and metadata.

Parameters

in	*reader*	Reader handle.
out	*runId*	Run ID.
out	*info*	Target info.
out	*nVertex*	Number of vertices in the Run's trajectory.
out	*ppPath*	Array of Trajectory Vertex info structs.
out	*pMetadata*	Run metadata. Metadata must be freed using Lgsx_FreeHandle().

Returns

Returns 0 for success. Returns -1 when end of collection is reached.

6.1.12.2.14 Lgsx_ReaderMoveNextRun2()

```
int Lgsx_ReaderMoveNextRun2 (
            LgsxReaderPtr reader,
            uint64_t * runId,
            CYTRAJECTORYINFO * info,
            int * nVertex,
            CYTRAJECTORYVERTEX ** ppPath,
            LgsxMetadataPtr * pMetadata,
            wchar_t ** pJobNamePtr,
            wchar_t ** pRunNamePtr)
```

Advance to the next Run and return full (untruncated) job and run names alongside the complete Run info.

See also

Lgsx_ReaderMoveNextRun() for a version without uncapped name output.

The `jobName` and `runName` fields in CYTRAJECTORYINFO may be truncated. Use this function to get the complete names.

Parameters

in	*reader*	Reader handle.
out	*runId*	Run ID.
out	*info*	Run/Trajectory info (name fields may be truncated; use pJobNamePtr/pRunNamePtr instead).
out	*nVertex*	Number of vertices in the Run's trajectory.
out	*ppPath*	Array of Trajectory Vertex info structs.
out	*pMetadata*	Run metadata. Metadata must be freed using Lgsx_FreeHandle(). May be NULL.
out	*pJobNamePtr*	Pointer to store pointer to the full job name. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.
out	*pRunNamePtr*	Pointer to store pointer to the full run name. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.

Returns

Returns 0 for success. Returns -1 when end of collection is reached.

6.1.12.2.15 Lgsx_ReaderMoveNextSetupCamera()

```
int Lgsx_ReaderMoveNextSetupCamera (
            LgsxReaderPtr reader,
            uint64_t * pSetupCameraId,
            CYSETUPCAMERAINFO * pSetupCamera,
            LgsxMetadataPtr * pMetadata)
```

Advance to the next Setup Camera in the active collection and return the ID, info, and metadata.

Parameters

in	*reader*	Reader handle.
out	*pSetup↩ CameraId*	Run ID.
out	*pSetupCamera*	Setup Camera info.
out	*pMetadata*	Setup Camera metadata. Metadata must be freed using Lgsx_FreeHandle(). If NULL is given as input, metadata is not returned.

Returns

Returns 0 for success. Returns -1 when end of collection is reached.

6.1.12.2.16 Lgsx_ReaderRunGetGuid()

```
int Lgsx_ReaderRunGetGuid (
            LgsxReaderPtr reader,
            char ** pGuidPtr)
```

Gets the Run GUID for the current Run in the enumeration (Lgsx_ReaderEnumRuns).

Parameters

in	*reader*	Reader handle.
out	*pGuidPtr*	Pointer to store pointer to GUID string. Buffer must be freed using LgsxUtil_FreeMem(). May be an empty string if the source object does not have a GUID.

Returns

Returns 0 for success. Returns -1 if an error occurs.

6.1.13 GeoTag Functions

Data Structures

- struct GeoTagHandle

  GeoTag data handle for accessing the GeoTag data.

Functions

- int Lgsx_ReaderEnumGeoTags (LgsxReaderPtr reader, int *pNumGeoTag)

  Collects GeoTags in the project and returns the count.
- int Lgsx_ReaderEnumGeoTagsOfSetup (LgsxReaderPtr reader, uint64_t setupId, int *pNumGeoTag)

  Enumerate all geotags that are visible or related to the given setup.
- int Lgsx_ReaderMoveNextGeoTag (LgsxReaderPtr reader, uint64_t *geoTagId, CYGEOTAG *pGeoTag, LgsxMetadataPtr *pMetadata)

  Advance to the next GeoTag in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderEndGeoTags (LgsxReaderPtr reader)

  Frees the active GeoTags collection.
- int Lgsx_ReaderGetGeoTagName (LgsxReaderPtr reader, uint64_t geoTagId, wchar_t **pNamePtr)

  Get name attribute of the GeoTag with the given ID.
- int Lgsx_ReaderHasGeoTagDescription (LgsxReaderPtr reader, uint64_t geoTagId, bool *pStatus)

  Check if the GeoTag with the given ID has a description attribute.
- int Lgsx_ReaderGetGeoTagDescription (LgsxReaderPtr reader, uint64_t geoTagId, wchar_t **pDescPtr)

  Get description attribute of the GeoTag with the given ID.
- int Lgsx_ReaderGetGeoTag (LgsxReaderPtr reader, int pos, GeoTagHandle *pHandle)

  Read GeoTag data at the given position in the active collection.
- int Lgsx_GeoTagRelease (GeoTagHandle *pHandle)

  Release the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetId (const GeoTagHandle handle, uint64_t *pGeoTagId)

  Get the numeric ID of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetGuid (const GeoTagHandle handle, const char **pGuidPtr)

  Get the GUID of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetName (const GeoTagHandle handle, const wchar_t **pNamePtr)

  Get name attribute of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagHasDescription (const GeoTagHandle handle, bool *pStatus)

  Check if the GeoTag has a description attribute.
- int Lgsx_GeoTagGetDescription (const GeoTagHandle handle, const wchar_t **pDescPtr)

  Get description attribute of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetPosition (const GeoTagHandle handle, PNT3D *pGeoTagPosition)

  Get the position of the GeoTag.
- int Lgsx_GeoTagGetRelativePosition (const GeoTagHandle handle, PNT3D *pGeoTagPosition)

  Get the relative position of the GeoTag.
- int Lgsx_GeoTagHasCameraPosition (const GeoTagHandle handle, bool *pStatus)

  Check if the GeoTag has a camera position.
- int Lgsx_GeoTagGetCameraPosition (const GeoTagHandle handle, PNT3D *pCameraPosition)

  Get the camera position of the GeoTag.
- int Lgsx_GeoTagGetCameraRelativePosition (const GeoTagHandle handle, PNT3D *pCameraPosition)

  Get the relative camera position of the GeoTag.
- int Lgsx_GeoTagGetAnchor (const GeoTagHandle handle, GeotagAnchorType *pAnchorType, uint64_t *pAnchorId)

  Get the anchor of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetCreationTimestamp (const GeoTagHandle handle, uint64_t *pCreationTimestamp)

  Get the creation timestamp of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetModificationTimestamp (const GeoTagHandle handle, uint64_t *pModificationTimestamp)

  Get the modification timestamp of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetMetadata (const GeoTagHandle handle, LgsxMetadataPtr *pMetadata)

  Get the metadata of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

- int Lgsx_GeoTagEnumCategoryEntries (const GeoTagHandle handle, int ∗pNumEntries)

  Enumerate the Category Entries associated with the GeoTag and return the count.
- int Lgsx_GeoTagGetCategoryEntry (const GeoTagHandle handle, int pos, CategoryEntryHandle ∗pEntry↩ Handle)

  Get a Category Entry at the specified position for the GeoTag.
- int Lgsx_GeoTagEnumFieldEntries (const GeoTagHandle handle, int ∗pNumEntries)

  Enumerate the Field Entries associated with the GeoTag and return the count.
- int Lgsx_GeoTagGetFieldEntry (const GeoTagHandle handle, int pos, FieldEntryHandle ∗pEntryHandle)

  Get a Field Entry at the specified position for the GeoTag.
- int Lgsx_GeoTagEnumAssets (const GeoTagHandle handle, int ∗pNumAssets)

  Collects Assets associated with the given GeoTag and returns the count.
- int Lgsx_ReaderGetGeoTagAsset (LgsxReaderPtr reader, const GeoTagHandle handle, int pos, CYASSET ∗pAsset, LgsxMetadataPtr ∗pMetadata)

  Get Info and metadata for the GeoTag Asset at the given position.
- int Lgsx_ReaderGetGeoTagAssetThumbnail (LgsxReaderPtr reader, const GeoTagHandle handle, int pos, int ∗pWidth, int ∗pHeight, int ∗pWidthInBytes, ImagePixelType ∗pPixelType, unsigned char ∗∗pImage)

  Get the thumbnail image for GeoTag Asset at the given position.
- int Lgsx_ReaderGetGeoTagAssetHandle (LgsxReaderPtr reader, const GeoTagHandle handle, int pos, AssetHandle ∗pHandle)

  Acquire an Asset handle for the GeoTag Asset at the given position.
- int Lgsx_GeoTagHasThumbnail (const GeoTagHandle handle, bool ∗pHasThumbnail)

  Checks if GeoTag has a thumbnail.
- int Lgsx_GeoTagGetThumbnail (const GeoTagHandle handle, int ∗pWidth, int ∗pHeight, int ∗pWidthInBytes, ImagePixelType ∗pPixelType, unsigned char ∗∗pImage)

  Get the thumbnail image for the GeoTag itself.

6.1.13.1 Detailed Description

6.1.13.2 Function Documentation

6.1.13.2.1 Lgsx_GeoTagEnumAssets()

```
int Lgsx_GeoTagEnumAssets (
            const GeoTagHandle handle,
            int * pNumAssets)
```

Collects Assets associated with the given GeoTag and returns the count.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pNumAssets*	Number of collected Assets.

Returns

Returns 0 for success.

6.1.13.2.2 Lgsx_GeoTagEnumCategoryEntries()

```
int Lgsx_GeoTagEnumCategoryEntries (
            const GeoTagHandle handle,
            int * pNumEntries)
```

Enumerate the Category Entries associated with the GeoTag and return the count.

Parameters

in	*handle*	GeoTag handle.
out	*pNumEntries*	Pointer to store the number of Category Entries in the GeoTag.

Returns

Returns 0 for success.

6.1.13.2.3 Lgsx_GeoTagEnumFieldEntries()

```
int Lgsx_GeoTagEnumFieldEntries (
            const GeoTagHandle handle,
            int * pNumEntries)
```

Enumerate the Field Entries associated with the GeoTag and return the count.

Parameters

in	*handle*	GeoTag handle.
out	*pNumEntries*	Pointer to store the number of Field Entries in the GeoTag.

Returns

Returns 0 for success.

6.1.13.2.4 Lgsx_GeoTagGetAnchor()

```
int Lgsx_GeoTagGetAnchor (
            const GeoTagHandle handle,
            GeotagAnchorType * pAnchorType,
            uint64_t * pAnchorId)
```

Get the anchor of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pAnchorType*	Pointer to store anchor object type of the GeoTag.
out	*pAnchorId*	Pointer to store anchor object ID of the GeoTag.

Returns

Returns 0 for success.

6.1.13.2.5 Lgsx_GeoTagGetCameraPosition()

```
int Lgsx_GeoTagGetCameraPosition (
            const GeoTagHandle handle,
            PNT3D * pCameraPosition)
```

Get the camera position of the GeoTag.

Camera position is specified in the project coordinates.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pCameraPosition*	Pointer to a structure where the function will store the camera position.

Returns

Returns 0 for success.

6.1.13.2.6 Lgsx_GeoTagGetCameraRelativePosition()

```
int Lgsx_GeoTagGetCameraRelativePosition (
            const GeoTagHandle handle,
            PNT3D * pCameraPosition)
```

Get the relative camera position of the GeoTag.

Camera position is specified in the coordinates relative to setup.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pCameraPosition*	Pointer to a structure where the function will store the relative camera position.

Returns

Returns 0 for success.

6.1.13.2.7 Lgsx_GeoTagGetCategoryEntry()

```
int Lgsx_GeoTagGetCategoryEntry (
            const GeoTagHandle handle,
            int pos,
            CategoryEntryHandle * pEntryHandle)
```

Get a Category Entry at the specified position for the GeoTag.

Parameters

in	*handle*	GeoTag handle.
in	*pos*	Position in the GeoTag's Category Entry collection.
out	*pEntryHandle*	Pointer to store the Category Entry handle. Returns read-only pointer bound to lifetime of GeoTagHandle, shall not be released manually.

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.13.2.8 Lgsx_GeoTagGetCreationTimestamp()

```
int Lgsx_GeoTagGetCreationTimestamp (
            const GeoTagHandle handle,
            uint64_t * pCreationTimestamp)
```

Get the creation timestamp of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pCreationTimestamp*	Pointer to store creation timestamp of the GeoTag.

Returns

Returns 0 for success.

6.1.13.2.9 Lgsx_GeoTagGetDescription()

```
int Lgsx_GeoTagGetDescription (
            const GeoTagHandle handle,
            const wchar_t ** pDescPtr)
```

Get description attribute of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pDescPtr*	Pointer to store description buffer. Buffer is read-only, bound to the lifetime of the GeoTag data. Must not be freed.

Returns

Returns 0 for success.

6.1.13.2.10 Lgsx_GeoTagGetFieldEntry()

```
int Lgsx_GeoTagGetFieldEntry (
            const GeoTagHandle handle,
            int pos,
            FieldEntryHandle * pEntryHandle)
```

Get a Field Entry at the specified position for the GeoTag.

Parameters

in	*handle*	GeoTag handle.
in	*pos*	Position in the GeoTag's Field Entry collection.
out	*pEntryHandle*	Pointer to store the Field Entry handle. Returns read-only pointer bound to lifetime of GeoTagHandle, shall not be released manually.

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.13.2.11 Lgsx_GeoTagGetGuid()

```
int Lgsx_GeoTagGetGuid (
            const GeoTagHandle handle,
            const char ** pGuidPtr)
```

Get the GUID of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pGuidPtr*	Pointer to store GUID buffer of the GeoTag. Buffer is read-only, bound to the lifetime of the GeoTag data. Must not be freed. May be an empty string if the source object does not have a GUID.

Returns

Returns 0 for success.

6.1.13.2.12 Lgsx_GeoTagGetId()

```
int Lgsx_GeoTagGetId (
            const GeoTagHandle handle,
            uint64_t * pGeoTagId)
```

Get the numeric ID of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pGeoTagId*	Pointer to store GeoTag ID.

Returns

Returns 0 for success.

6.1.13.2.13 Lgsx_GeoTagGetMetadata()

```
int Lgsx_GeoTagGetMetadata (
            const GeoTagHandle handle,
            LgsxMetadataPtr * pMetadata)
```

Get the metadata of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pMetadata*	GeoTag metadata. Metada is bound to the lifetime of the GeoTagHandle data. Must not be freed.

Returns

Returns 0 for success.

6.1.13.2.14 Lgsx_GeoTagGetModificationTimestamp()

```
int Lgsx_GeoTagGetModificationTimestamp (
            const GeoTagHandle handle,
            uint64_t * pModificationTimestamp)
```

Get the modification timestamp of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pModificationTimestamp*	Pointer to store modification timestamp of the GeoTag.

Returns

Returns 0 for success.

6.1.13.2.15 Lgsx_GeoTagGetName()

```
int Lgsx_GeoTagGetName (
            const GeoTagHandle handle,
            const wchar_t ** pNamePtr)
```

Get name attribute of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pNamePtr*	Pointer to store name buffer. Buffer is read-only, bound to the lifetime of the GeoTag data. Must not be freed.

Returns

Returns 0 for success.

6.1.13.2.16 Lgsx_GeoTagGetPosition()

```
int Lgsx_GeoTagGetPosition (
            const GeoTagHandle handle,
            PNT3D * pGeoTagPosition)
```

Get the position of the GeoTag.

Position is specified in the project coordinates.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pGeoTagPosition*	Pointer to a structure where the function will store the position of the GeoTag.

Returns

Returns 0 for success.

6.1.13.2.17 Lgsx_GeoTagGetRelativePosition()

```
int Lgsx_GeoTagGetRelativePosition (
            const GeoTagHandle handle,
            PNT3D * pGeoTagPosition)
```

Get the relative position of the GeoTag.

Position is specified in the the coordinates relative to setup.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pGeoTagPosition*	Pointer to a structure where the function will store the relative position of the GeoTag.

Returns

Returns 0 for success.

6.1.13.2.18 Lgsx_GeoTagGetThumbnail()

```
int Lgsx_GeoTagGetThumbnail (
            const GeoTagHandle handle,
            int * pWidth,
            int * pHeight,
            int * pWidthInBytes,
            ImagePixelType * pPixelType,
            unsigned char ** pImage)
```

Get the thumbnail image for the GeoTag itself.

Note

If the thumbnail does not exist, an error is returned; please check that GeoTag has thumbnail using Lgsx_GeoTagHasThumbnail().

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pWidth*	Image width.
out	*pHeight*	Image height.
out	*pWidthInBytes*	Image byte-width.
out	*pPixelType*	Pixel type.
out	*pImage*	GeoTag thumbnail image.

Returns

Returns 0 for success.

6.1.13.2.19 Lgsx_GeoTagHasCameraPosition()

```
int Lgsx_GeoTagHasCameraPosition (
            const GeoTagHandle handle,
            bool * pStatus)
```

Check if the GeoTag has a camera position.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pStatus*	Pointer to flag where the function will store wether geotag has camera position.

Returns

Returns 0 for success.

6.1.13.2.20 Lgsx_GeoTagHasDescription()

```
int Lgsx_GeoTagHasDescription (
            const GeoTagHandle handle,
            bool * pStatus)
```

Check if the GeoTag has a description attribute.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pStatus*	Pointer to store the status.

Returns

Returns 0 for success.

6.1.13.2.21 Lgsx_GeoTagHasThumbnail()

```
int Lgsx_GeoTagHasThumbnail (
            const GeoTagHandle handle,
            bool * pHasThumbnail)
```

Checks if GeoTag has a thumbnail.

Parameters

in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
out	*pHasThumbnail*	Set to true if the GeoTag has thumbnail.

Returns

Returns 0 for success.

6.1.13.2.22 Lgsx_GeoTagRelease()

```
int Lgsx_GeoTagRelease (
            GeoTagHandle * pHandle)
```

Release the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().

Parameters

in	*pHandle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().

Returns

Returns 0 for success.

6.1.13.2.23 Lgsx_ReaderEndGeoTags()

```
int Lgsx_ReaderEndGeoTags (
            LgsxReaderPtr reader)
```

Frees the active GeoTags collection.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.13.2.24 Lgsx_ReaderEnumGeoTags()

```
int Lgsx_ReaderEnumGeoTags (
            LgsxReaderPtr reader,
            int * pNumGeoTag)
```

Collects GeoTags in the project and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumGeoTag*	Number of collected GeoTags.

Returns

0 for success.

6.1.13.2.25 Lgsx_ReaderEnumGeoTagsOfSetup()

```
int Lgsx_ReaderEnumGeoTagsOfSetup (
            LgsxReaderPtr reader,
            uint64_t setupId,
            int * pNumGeoTag)
```

Enumerate all geotags that are visible or related to the given setup.

Note

Being visible or related to setup does not necessarily mean that the geotag is attached to the setup, e.g. the setup index assigned to such GeoTag might be equal to zero.

Parameters

in	*reader*	Reader handle.
in	*setupId*	Setup associated with GeoTags.
out	*pNumGeoTag*	Number of collected GeoTags.

Returns

0 for success.

6.1.13.2.26 Lgsx_ReaderGetGeoTag()

```
int Lgsx_ReaderGetGeoTag (
            LgsxReaderPtr reader,
            int pos,
            GeoTagHandle * pHandle)
```

Read GeoTag data at the given position in the active collection.

Active collection is set by Lgsx_ReaderEnumGeoTags() or Lgsx_ReaderEnumGeoTagsOfSetup().

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in the active collection.
out	*pHandle*	Pointer to store pointer to GeoTag data. Buffer must be freed using Lgsx_GeoTagRelease().

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.13.2.27 Lgsx_ReaderGetGeoTagAsset()

```
int Lgsx_ReaderGetGeoTagAsset (
            LgsxReaderPtr reader,
            const GeoTagHandle handle,
            int pos,
            CYASSET * pAsset,
            LgsxMetadataPtr * pMetadata)
```

Get Info and metadata for the GeoTag Asset at the given position.

Parameters

in	*reader*	Reader handle.
in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
in	*pos*	Position in the GeoTag's Asset collection.
out	*pAsset*	Asset info.
out	*pMetadata*	Asset metadata. Metadata must be freed using Lgsx_FreeHandle().

Returns

Returns 0 for success.

6.1.13.2.28 Lgsx_ReaderGetGeoTagAssetHandle()

```
int Lgsx_ReaderGetGeoTagAssetHandle (
            LgsxReaderPtr reader,
            const GeoTagHandle handle,
            int pos,
            AssetHandle * pHandle)
```

Acquire an Asset handle for the GeoTag Asset at the given position.

See also

Lgsx_ReaderGetGeoTagAsset() for the truncating alternative using CYASSET.

Parameters

in	*reader*	Reader handle.
in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
in	*pos*	Position in the GeoTag's Asset collection.
out	*pHandle*	Pointer to store the Asset handle. Buffer must be freed using Lgsx_AssetRelease().

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.13.2.29 Lgsx_ReaderGetGeoTagAssetThumbnail()

```
int Lgsx_ReaderGetGeoTagAssetThumbnail (
            LgsxReaderPtr reader,
            const GeoTagHandle handle,
            int pos,
            int * pWidth,
            int * pHeight,
            int * pWidthInBytes,
            ImagePixelType * pPixelType,
            unsigned char ** pImage)
```

Get the thumbnail image for GeoTag Asset at the given position.

Parameters

in	*reader*	Reader handle.
in	*handle*	Pointer to GeoTag data returned by Lgsx_ReaderGetGeoTag().
in	*pos*	Position in the GeoTag's Asset collection.
out	*pWidth*	Image width.
out	*pHeight*	Image height.
out	*pWidthInBytes*	Thumbnail image byte-width.
out	*pPixelType*	Pixel type.
out	*pImage*	Asset thumbnail image.

Returns

Returns 0 for success.

6.1.13.2.30 Lgsx_ReaderGetGeoTagDescription()

```
int Lgsx_ReaderGetGeoTagDescription (
            LgsxReaderPtr reader,
            uint64_t geoTagId,
            wchar_t ** pDescPtr)
```

Get description attribute of the GeoTag with the given ID.

Parameters

in	*reader*	Reader handle.
in	*geo↩ TagId*	GeoTag ID.
out	*pDescPtr*	Pointer to store description buffer. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success.

Deprecated 26.06.2025 - New GeoTag API available via GeoTagHandle

6.1.13.2.31 Lgsx_ReaderGetGeoTagName()

```
int Lgsx_ReaderGetGeoTagName (
            LgsxReaderPtr reader,
            uint64_t geoTagId,
            wchar_t ** pNamePtr)
```

Get name attribute of the GeoTag with the given ID.

Parameters

in	*reader*	Reader handle.
in	*geoTagId*	GeoTag ID.
out	*pNamePtr*	Pointer to store name buffer. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success.

Deprecated 26.06.2025 - New GeoTag API available via GeoTagHandle

6.1.13.2.32 Lgsx_ReaderHasGeoTagDescription()

```
int Lgsx_ReaderHasGeoTagDescription (
            LgsxReaderPtr reader,
            uint64_t geoTagId,
            bool * pStatus)
```

Check if the GeoTag with the given ID has a description attribute.

Parameters

in	*reader*	Reader handle.
in	*geo↩ TagId*	GeoTag ID.
out	*pStatus*	Pointer to store the status.

Returns

Returns 0 for success.

Deprecated 26.06.2025 - New GeoTag API available via GeoTagHandle

6.1.13.2.33 Lgsx_ReaderMoveNextGeoTag()

```
int Lgsx_ReaderMoveNextGeoTag (
            LgsxReaderPtr reader,
            uint64_t * geoTagId,
            CYGEOTAG * pGeoTag,
            LgsxMetadataPtr * pMetadata)
```

Advance to the next GeoTag in the active collection and return the ID, info, and metadata.

Parameters

in	*reader*	Reader handle.
out	*geoTagId*	GeoTag ID.
out	*pGeoTag*	GeoTag info.
out	*pMetadata*	GeoTag metadata. Metadata must be freed using Lgsx_FreeHandle().

Returns

Returns 0 for success. Returns -1 when end of collection is reached.

Deprecated 26.06.2025 - New GeoTag API available via GeoTagHandle

6.1.14 Sitemap Functions

Data Structures

- struct SitemapHandle

  Sitemap data handle for accessing Sitemap data without reader-level cursor state.
- struct SitemapItemHandle

  Sitemap item handle bound to the lifetime of its parent SitemapHandle.

Functions

- int Lgsx_ReaderEnumSitemaps (LgsxReaderPtr reader, int *pNumSitemap)

  Collects Sitemaps in the project and returns the count.
- int Lgsx_ReaderGetSitemapHandle (LgsxReaderPtr reader, int pos, SitemapHandle *pHandle)

  Get a SitemapHandle at the given position in the active collection.
- int Lgsx_SitemapRelease (SitemapHandle *pHandle)

  Release a SitemapHandle returned by Lgsx_ReaderGetSitemapHandle().
- int Lgsx_SitemapGetId (const SitemapHandle handle, uint64_t *pId)

  Get the numeric ID of the Sitemap.
- int Lgsx_SitemapGetGuid (const SitemapHandle handle, const char **pGuid)

  Get the GUID of the Sitemap.
- int Lgsx_SitemapGetName (const SitemapHandle handle, const wchar_t **pName)

  Get the display name of the Sitemap.
- int Lgsx_SitemapGetOrderingIndex (const SitemapHandle handle, int *pIndex)

  Get the display ordering index of the Sitemap.
- int Lgsx_SitemapGetIsMaster (const SitemapHandle handle, bool *pIsMaster)

  Returns true if this is a master sitemap.
- int Lgsx_SitemapGetMetadata (const SitemapHandle handle, LgsxMetadataPtr *pMetadata)

  Get the metadata of the Sitemap.
- int Lgsx_SitemapHasImage (const SitemapHandle handle, SitemapImageType imageType, bool *pHas↩ Image)

  Check whether a sitemap image of the given type exists.
- int Lgsx_SitemapImageGetHandle (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, AssetHandle *pAssetHandle)

  Acquire an AssetHandle for one of the sitemap images.
- int Lgsx_SitemapImageGetMetadata (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, LgsxMetadataPtr *pMetadata)

  Get metadata for one of the sitemap images from image file metadata.
- int Lgsx_SitemapImageGetIsBlank (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, bool *pIsBlank)

  Returns true if the given sitemap image is a blank placeholder.
- int Lgsx_SitemapImageGetCorners (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, CYSITEMAPIMAGECORNERS2D *pCorners, int *pFlags)

  Get the world-space image footprint for one of the sitemap images.
- int Lgsx_SitemapEnumItems (const SitemapHandle handle, int *pCount)

  Collect all items belonging to the sitemap and return the count.
- int Lgsx_SitemapGetItem (const SitemapHandle handle, int pos, SitemapItemHandle *pItemHandle)

  Get the sitemap item at the given position.
- int Lgsx_SitemapItemGetId (const SitemapItemHandle handle, uint64_t *pId)

  Get the numeric ID of a sitemap item.
- int Lgsx_SitemapItemGetType (const SitemapItemHandle handle, SitemapItemType *pType)

  Get the type of a sitemap item.
- int Lgsx_SitemapItemGetGuid (const SitemapItemHandle handle, const char **pGuid)

  Get the GUID of a sitemap item.
- int Lgsx_SitemapItemGetPose (const SitemapItemHandle handle, SCyRgdBdyTxf *pPose)

  Get the resolved project-space pose of a sitemap item.
- int Lgsx_SitemapGetActiveCoordSystemId (const SitemapHandle handle, uint64_t *pCoordSysId)

  Get the numeric ID of the active UCS associated with the sitemap.
- int Lgsx_SitemapImageRead (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **p↩ Image)

Read a selected sitemap image through the decoded-image path.

- int Lgsx_SitemapImageReadBinary (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, wchar_t **pImageType, int *pWidth, int *pHeight, unsigned char **pData, int *pSize)

  Retrieves the raw encoded image bytes for a sitemap image.
- int Lgsx_ReaderEndSitemaps (LgsxReaderPtr reader)

  Frees the active Sitemaps collection.

6.1.14.1 Detailed Description

6.1.14.2 Function Documentation

6.1.14.2.1 Lgsx_ReaderEndSitemaps()

```
int Lgsx_ReaderEndSitemaps (
            LgsxReaderPtr reader)
```

Frees the active Sitemaps collection.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.14.2.2 Lgsx_ReaderEnumSitemaps()

```
int Lgsx_ReaderEnumSitemaps (
            LgsxReaderPtr reader,
            int * pNumSitemap)
```

Collects Sitemaps in the project and returns the count.

Parameters

in	*reader*	Reader handle.
out	*pNumSitemap*	Number of collected Sitemaps.

Returns

0 for success.

6.1.14.2.3 Lgsx_ReaderGetSitemapHandle()

```
int Lgsx_ReaderGetSitemapHandle (
            LgsxReaderPtr reader,
            int pos,
            SitemapHandle * pHandle)
```

Get a SitemapHandle at the given position in the active collection.

Active collection is set by Lgsx_ReaderEnumSitemaps().

Parameters

in	*reader*	Reader handle.
in	*pos*	Position in the active collection (0-based).
out	*pHandle*	Pointer to store the Sitemap handle. Must be freed using Lgsx_SitemapRelease().

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.14.2.4 Lgsx_SitemapEnumItems()

```
int Lgsx_SitemapEnumItems (
            const SitemapHandle handle,
            int * pCount)
```

Collect all items belonging to the sitemap and return the count.

Parameters

in	*handle*	Sitemap handle.
out	*pCount*	Pointer to store the number of items.

Returns

Returns 0 for success.

6.1.14.2.5 Lgsx_SitemapGetActiveCoordSystemId()

```
int Lgsx_SitemapGetActiveCoordSystemId (
            const SitemapHandle handle,
            uint64_t * pCoordSysId)
```

Get the numeric ID of the active UCS associated with the sitemap.

Parameters

in	*handle*	Sitemap handle.
out	*pCoord↩ SysId*	Pointer to store the UCS numeric ID. Set to 0 if no UCS is active.

Returns

Returns 0 for success.

6.1.14.2.6 Lgsx_SitemapGetGuid()

```
int Lgsx_SitemapGetGuid (
            const SitemapHandle handle,
            const char ** pGuid)
```

Get the GUID of the Sitemap.

Parameters

in	*handle*	Sitemap handle.
out	*pGuid*	Pointer to store the GUID string. Buffer is read-only, bound to the lifetime of the SitemapHandle. Must not be freed.

Returns

Returns 0 for success.

6.1.14.2.7 Lgsx_SitemapGetId()

```
int Lgsx_SitemapGetId (
            const SitemapHandle handle,
            uint64_t * pId)
```

Get the numeric ID of the Sitemap.

Parameters

in	*handle*	Sitemap handle.
out	*pId*	Pointer to store the numeric ID.

Returns

Returns 0 for success.

6.1.14.2.8 Lgsx_SitemapGetIsMaster()

```
int Lgsx_SitemapGetIsMaster (
            const SitemapHandle handle,
            bool * pIsMaster)
```

Returns true if this is a master sitemap.

Parameters

in	*handle*	Sitemap handle.
out	*pIsMaster*	Pointer to store the master flag.

Returns

Returns 0 for success.

6.1.14.2.9 Lgsx_SitemapGetItem()

```
int Lgsx_SitemapGetItem (
            const SitemapHandle handle,
            int pos,
            SitemapItemHandle * pItemHandle)
```

Get the sitemap item at the given position.

Parameters

in	*handle*	Sitemap handle.
in	*pos*	Position in the item collection (0-based).
out	*pItemHandle*	Pointer to store the item handle. Read-only, bound to the lifetime of the SitemapHandle. Must not be freed.

Returns

Returns 0 for success. Returns -1 when position is out of range.

6.1.14.2.10 Lgsx_SitemapGetMetadata()

```
int Lgsx_SitemapGetMetadata (
            const SitemapHandle handle,
            LgsxMetadataPtr * pMetadata)
```

Get the metadata of the Sitemap.

Parameters

in	*handle*	Sitemap handle.
out	*pMetadata*	Sitemap metadata. Metadata is returned as an independent handle and must be freed using Lgsx_FreeHandle(). If NULL is given as input, metadata is not returned.

Returns

Returns 0 for success.

6.1.14.2.11 Lgsx_SitemapGetName()

```
int Lgsx_SitemapGetName (
            const SitemapHandle handle,
            const wchar_t ** pName)
```

Get the display name of the Sitemap.

Parameters

in	*handle*	Sitemap handle.
out	*pName*	Pointer to store the name string. Buffer is read-only, bound to the lifetime of the SitemapHandle. Must not be freed.

Returns

Returns 0 for success.

6.1.14.2.12 Lgsx_SitemapGetOrderingIndex()

```
int Lgsx_SitemapGetOrderingIndex (
            const SitemapHandle handle,
            int * pIndex)
```

Get the display ordering index of the Sitemap.

Parameters

in	*handle*	Sitemap handle.
out	*pIndex*	Pointer to store the ordering index.

Returns

Returns 0 for success.

6.1.14.2.13 Lgsx_SitemapHasImage()

```
int Lgsx_SitemapHasImage (
            const SitemapHandle handle,
            SitemapImageType imageType,
            bool * pHasImage)
```

Check whether a sitemap image of the given type exists.

Parameters

in	*handle*	Sitemap handle.
in	*imageType*	Type of image (Background, Overview, or Acceptance).
out	*pHasImage*	Pointer to store the result.

Returns

Returns 0 for success.

6.1.14.2.14 Lgsx_SitemapImageGetCorners()

```
int Lgsx_SitemapImageGetCorners (
            LgsxReaderPtr reader,
            const SitemapHandle handle,
            SitemapImageType imageType,
            CYSITEMAPIMAGECORNERS2D * pCorners,
            int * pFlags)
```

Get the world-space image footprint for one of the sitemap images.

The four corners define the image rectangle in project coordinates.

Automatic corner correction is controlled by `pFlags`. Three fixes are available:

- LGSX_SITEMAP_CORNERS_CALCULATE_MISSING: reconstruct corners from setup/track pixel-position correspondences when corner data is absent.
- LGSX_SITEMAP_CORNERS_FIX_INCONSISTENT: detect corners that are inconsistent with the recorded setup pixel positions (e.g. stored in a different coordinate system) and reconstruct them from the correspondences.
- LGSX_SITEMAP_CORNERS_REJECT_DEGENERATE: treat geometrically degenerate stored corners as missing. Combined with LGSX_SITEMAP_CORNERS_CALCULATE_MISSING, reconstruction is attempted; without it, the function returns an error.

The reconstruction fixes require a readable image file (to obtain pixel dimensions) and at least two setup-to-pixel correspondences. If either prerequisite is unavailable the fix is silently skipped and the original corners (or error) are returned.

Parameters

in	*reader*	Reader handle.
in	*handle*	Sitemap handle.
in	*imageType*	Type of image (Background, Overview, or Acceptance).
out	*pCorners*	Pointer to store the normalized image corners.
in,out	*pFlags*	Optional bitmask pointer. On entry: bitwise OR of the fix flags to enable. Pass NULL to enable all fixes (LGSX_SITEMAP_CORNERS_CALCULATE_MISSING | LGSX_SITEMAP_CORNERS_FIX_INCONSISTENT | LGSX_SITEMAP_CORNERS_REJECT_DEGENERATE). On exit: set to the flags of the fix(es) that were actually applied (0 if corners were returned as-is from metadata).

Returns

Returns 0 for success. Returns non-zero if corners are not available and could not be reconstructed.

Note

This is a breaking change from the previous four-argument signature. Existing call sites must add NULL as the fifth argument to restore the previous behaviour.

6.1.14.2.15 Lgsx_SitemapImageGetHandle()

```
int Lgsx_SitemapImageGetHandle (
            LgsxReaderPtr reader,
            const SitemapHandle handle,
            SitemapImageType imageType,
            AssetHandle * pAssetHandle)
```

Acquire an AssetHandle for one of the sitemap images.

Parameters

in	*reader*	Reader handle.
in	*handle*	Sitemap handle.
in	*imageType*	Type of image (Background, Overview, or Acceptance).
out	*pAssetHandle*	Pointer to store the Asset handle. Must be freed using Lgsx_AssetRelease().

Returns

Returns 0 for success. Returns non-zero if the image type is not available.

6.1.14.2.16 Lgsx_SitemapImageGetIsBlank()

```
int Lgsx_SitemapImageGetIsBlank (
            LgsxReaderPtr reader,
            const SitemapHandle handle,
            SitemapImageType imageType,
            bool * pIsBlank)
```

Returns true if the given sitemap image is a blank placeholder.

Parameters

in	*reader*	Reader handle.
in	*handle*	Sitemap handle.
in	*imageType*	Type of image (Background, Overview, or Acceptance).
out	*pIsBlank*	Pointer to store the blank flag. Set to false if IsBlank metadata is not present.

Returns

Returns 0 for success. Returns non-zero if the image type is not available.

6.1.14.2.17 Lgsx_SitemapImageGetMetadata()

```
int Lgsx_SitemapImageGetMetadata (
            LgsxReaderPtr reader,
            const SitemapHandle handle,
            SitemapImageType imageType,
            LgsxMetadataPtr * pMetadata)
```

Get metadata for one of the sitemap images from image file metadata.

Parameters

in	*reader*	Reader handle.
in	*handle*	Sitemap handle.
in	*imageType*	Type of image (Background, Overview, or Acceptance).
out	*pMetadata*	Image metadata. Metadata is returned as an independent handle and must be freed using Lgsx_FreeHandle(). If NULL is given as input, metadata is not returned.

Returns

Returns 0 for success. Returns non-zero if the image type is not available.

6.1.14.2.18 Lgsx_SitemapImageRead()

```
int Lgsx_SitemapImageRead (
            LgsxReaderPtr reader,
            const SitemapHandle handle,
            SitemapImageType imageType,
            int * pWidth,
            int * pHeight,
            int * pWidthInBytes,
            ImagePixelType * pPixelType,
            unsigned char ** pImage)
```

Read a selected sitemap image through the decoded-image path.

Parameters

in	*reader*	Reader handle.

Parameters

in	*handle*	Sitemap handle.
in	*imageType*	Type of image to retrieve (Background, Overview, or Acceptance).
out	*pWidth*	Image width in pixels.
out	*pHeight*	Image height in pixels.
out	*pWidthInBytes*	Image row width in bytes.
out	*pPixelType*	Pixel format.
out	*pImage*	Pointer to the decoded image data.

Returns

Returns 0 for success.

6.1.14.2.19 Lgsx_SitemapImageReadBinary()

```
int Lgsx_SitemapImageReadBinary (
            LgsxReaderPtr reader,
            const SitemapHandle handle,
            SitemapImageType imageType,
            wchar_t ** pImageType,
            int * pWidth,
            int * pHeight,
            unsigned char ** pData,
            int * pSize)
```

Retrieves the raw encoded image bytes for a sitemap image.

Returns the original image data without decoding. The caller receives the encoded bytes, the image type (mime), and pixel dimensions.

Note

This function may fail if the image format stored in the sitemap is not recognized by the implementation. Callers should fall back to the decoded-image path (Lgsx_SitemapImageRead()) when this function returns a non-zero error code.

Parameters

in	*reader*	Reader handle.
in	*handle*	Sitemap handle obtained via Lgsx_ReaderGetSitemapHandle().
in	*imageType*	Type of image to retrieve (Background, Overview, or Acceptance).
out	*pImageType*	Image type string (e.g. L"jpg", L"png"). Free with LgsxUtil_FreeMem().
out	*pWidth*	Image width in pixels.
out	*pHeight*	Image height in pixels.
out	*pData*	Pointer to receive the raw image bytes. Free with LgsxUtil_FreeMem().
out	*pSize*	Number of bytes in the pData buffer.

Returns

Returns 0 for success.

6.1.14.2.20 Lgsx_SitemapItemGetGuid()

```
int Lgsx_SitemapItemGetGuid (
            const SitemapItemHandle handle,
            const char ** pGuid)
```

Get the GUID of a sitemap item.

Parameters

in	*handle*	Sitemap item handle.
out	*pGuid*	Pointer to store the GUID string. Buffer is read-only, bound to the lifetime of the parent SitemapHandle. Must not be freed. May be an empty string if the source object does not have a GUID.

Returns

Returns 0 for success.

6.1.14.2.21 Lgsx_SitemapItemGetId()

```
int Lgsx_SitemapItemGetId (
            const SitemapItemHandle handle,
            uint64_t * pId)
```

Get the numeric ID of a sitemap item.

Parameters

in	*handle*	Sitemap item handle.
out	*pId*	Pointer to store the numeric ID.

Returns

Returns 0 for success.

6.1.14.2.22 Lgsx_SitemapItemGetPose()

```
int Lgsx_SitemapItemGetPose (
            const SitemapItemHandle handle,
            SCyRgdBdyTxf * pPose)
```

Get the resolved project-space pose of a sitemap item.

Parameters

in	*handle*	Sitemap item handle.
out	*pPose*	Pointer to store the rigid body transform (position + rotation).

Returns

Returns 0 for success.

6.1.14.2.23 Lgsx_SitemapItemGetType()

```
int Lgsx_SitemapItemGetType (
            const SitemapItemHandle handle,
            SitemapItemType * pType)
```

Get the type of a sitemap item.

Parameters

in	*handle*	Sitemap item handle.
out	*pType*	Pointer to store the item type (e.g. TlsSetup, MlsRun).

Returns

Returns 0 for success.

6.1.14.2.24 Lgsx_SitemapRelease()

```
int Lgsx_SitemapRelease (
            SitemapHandle * pHandle)
```

Release a SitemapHandle returned by Lgsx_ReaderGetSitemapHandle().

Parameters

in,out	*pHandle*	Pointer to the Sitemap handle to release.

Returns

Returns 0 for success.

6.1.15 Model and Model Node Functions

Functions

- int Lgsx_ReaderGetModelNodes (LgsxReaderPtr reader, int *pNumModelNode, uint64_t **ppModelNodeIds)

  Retrieve the Model Nodes for this project.
- int Lgsx_ReaderGetModelNodeInfo (LgsxReaderPtr reader, uint64_t nodeId, CYMODELNODEINFO *pNodeInfo, uint64_t **ppChildIds)

  Retrieve the info for a given Model Node.
- int Lgsx_ReaderGetModelNodeInfo2 (LgsxReaderPtr reader, uint64_t nodeId, CYMODELNODEINFO *pNodeInfo, uint64_t **ppChildIds, wchar_t **pNamePtr)

  Retrieve the info for a given Model Node, with the full (untruncated) node name.
- int Lgsx_ReaderGetModelInfo (LgsxReaderPtr reader, uint64_t modelId, CYMODELINFO *pModelInfo, LgsxMetadataPtr *pMetadata, uint64_t **ppRefAssetIds)

  Retrieve the info for a given Model.
- int Lgsx_ReaderGetModelInfo2 (LgsxReaderPtr reader, uint64_t modelId, CYMODELINFO *pModelInfo, LgsxMetadataPtr *pMetadata, uint64_t **ppRefAssetIds, wchar_t **pNamePtr)

  Retrieve the info for a given Model, with the full (untruncated) model name.

6.1.15.1 Detailed Description

6.1.15.2 Function Documentation

6.1.15.2.1 Lgsx_ReaderGetModelInfo()

```
int Lgsx_ReaderGetModelInfo (
            LgsxReaderPtr reader,
            uint64_t modelId,
            CYMODELINFO * pModelInfo,
            LgsxMetadataPtr * pMetadata,
            uint64_t ** ppRefAssetIds)
```

Retrieve the info for a given Model.

Each model object references two Assets: Source Rep and Scene Rep. The Source Rep represents the actual source model file (e.g., an IFC file). The Scene Rep Asset represents the model file converted into a form that is optimized for rendering (it is OpenInventor format).

Parameters

in	*reader*	Reader handle.
in	*modelId*	ID of the model to query.
out	*pModelInfo*	Info for the requested Model.
out	*pMetadata*	Model Metadata. Metadata must be freed using Lgsx_FreeHandle().
out	*ppRefAssetIds*	Model Asset IDs. Number of Assets is contained in the CYMODELINFO. These are the Source and Scene Assets mentioned above.

Returns

Returns 0 for success.

6.1.15.2.2 Lgsx_ReaderGetModelInfo2()

```
int Lgsx_ReaderGetModelInfo2 (
            LgsxReaderPtr reader,
            uint64_t modelId,
            CYMODELINFO * pModelInfo,
            LgsxMetadataPtr * pMetadata,
            uint64_t ** ppRefAssetIds,
            wchar_t ** pNamePtr)
```

Retrieve the info for a given Model, with the full (untruncated) model name.

See also

Lgsx_ReaderGetModelInfo() for the version without uncapped name output.

The `name` field in CYMODELINFO may be truncated. Use `pNamePtr` to get the complete name.

Parameters

in	*reader*	Reader handle.
in	*modelId*	ID of the model to query.
out	*pModelInfo*	Info for the requested Model (name field may be truncated).
out	*pMetadata*	Model Metadata. Metadata must be freed using Lgsx_FreeHandle(). May be NULL.
out	*ppRefAssetIds*	Model Asset IDs, or NULL to skip.
out	*pNamePtr*	Pointer to store pointer to the full model name. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.

Returns

Returns 0 for success.

6.1.15.2.3 Lgsx_ReaderGetModelNodeInfo()

```
int Lgsx_ReaderGetModelNodeInfo (
            LgsxReaderPtr reader,
            uint64_t nodeId,
            CYMODELNODEINFO * pNodeInfo,
            uint64_t ** ppChildIds)
```

Retrieve the info for a given Model Node.

A ModelNode represents an instance of a Model placed somewhere in the scene. Model Node holds a reference to the the Model object, as well as the transformation matrix which positions and orients the model within the project.

Parameters

in	*reader*	Reader handle.
in	*nodeId*	ID of the model node to query.
out	*pNodeInfo*	Info for the requested Model Node.
out	*ppChildIds*	Array of child Model Nodes. Note that the number of children in this array is contained within the Model Node info.

Returns

Returns 0 for success.

6.1.15.2.4 Lgsx_ReaderGetModelNodeInfo2()

```
int Lgsx_ReaderGetModelNodeInfo2 (
            LgsxReaderPtr reader,
            uint64_t nodeId,
            CYMODELNODEINFO * pNodeInfo,
            uint64_t ** ppChildIds,
            wchar_t ** pNamePtr)
```

Retrieve the info for a given Model Node, with the full (untruncated) node name.

See also

Lgsx_ReaderGetModelNodeInfo() for the version without uncapped name output.

The `name` field in CYMODELNODEINFO may be truncated. Use `pNamePtr` to get the complete name.

Parameters

in	*reader*	Reader handle.
in	*nodeId*	ID of the model node to query.
out	*pNodeInfo*	Info for the requested Model Node (name field may be truncated).
out	*ppChildIds*	Array of child Model Nodes, or NULL to skip.
out	*pNamePtr*	Pointer to store pointer to the full model node name. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.

Returns

Returns 0 for success.

6.1.15.2.5 Lgsx_ReaderGetModelNodes()

```
int Lgsx_ReaderGetModelNodes (
            LgsxReaderPtr reader,
            int * pNumModelNode,
            uint64_t ** ppModelNodeIds)
```

Retrieve the Model Nodes for this project.

Model nodes can form a hierarchical scene graph to combine multiple model "parts" into a single model.

Parameters

in	*reader*	Reader handle.
out	*pNumModelNode*	Number of Model Nodes returned.
out	*ppModelNodeIds*	Array of Model Node IDs.

Returns

Returns 0 for success.

6.1.16 Point Cloud Functions

Data Structures

- struct EnumPointsConfigHandle

  Configuration handle for Lgsx_ReaderEnumPointsEx2().
- struct PointCloudFileHandle

  Point Cloud handle for accessing the raw point cloud data.

Functions

- int Lgsx_ReaderQueryPropertyTypes (LgsxReaderPtr reader)

  Returns the Property Types available for points in the project's Point Cloud.
- int Lgsx_ReaderGetPropertyTypeInfo (LgsxReaderPtr reader, int typeMask, const wchar_t *name, int *pnBytes)

  Retrieve Property Type information from the Point Cloud.
- int Lgsx_ReaderEnumPoints (LgsxReaderPtr reader, int desiredTypes, int subSample, bool visible)

  Collects points in the Point Cloud.
- int Lgsx_ReaderEnumPointsEx (LgsxReaderPtr reader, int desiredTypes, int subSample, bool visible, bool scrictOrder)

  Collects points in the Point Cloud.
- int Lgsx_InitEnumPointsConfig (EnumPointsConfigHandle *pHandle)

  Allocate and initialise an EnumPointsConfigHandle with default values.
- int Lgsx_ReleaseEnumPointsConfig (EnumPointsConfigHandle *pHandle)

  Release an EnumPointsConfigHandle previously allocated by Lgsx_InitEnumPointsConfig().
- int Lgsx_EnumPointsConfigSetDesiredTypes (const EnumPointsConfigHandle handle, int desiredTypes)

  Set the desired property-type mask on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetSubSample (const EnumPointsConfigHandle handle, int subSample)

  Set the sub-sample rate on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetVisible (const EnumPointsConfigHandle handle, bool visible)

  Set the visibility filter on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetStrictOrder (const EnumPointsConfigHandle handle, bool strictOrder)

  Set the strict-order flag on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetMaxMemoryHintBytes (const EnumPointsConfigHandle handle, uint64_t maxMemoryHintBytes)

  Set the maximum memory hint on an EnumPointsConfigHandle.
- int Lgsx_ReaderEnumPointsEx2 (LgsxReaderPtr reader, const EnumPointsConfigHandle config)

  Collects points in the Point Cloud using the given configuration.
- int Lgsx_ReaderMoveNextPoints (LgsxReaderPtr reader, int chunkSize, int bmpFormat, double *points, float *intens, uchar *colors, float *normals)

  Advances to the next group of points in the Point Cloud and returns the basic properties.
- int Lgsx_ReaderMoveNextFields (LgsxReaderPtr reader, int chunkSize, int bmpFormat, void *ppData[ ])

  Advances to the next group of points in the Point Cloud and returns the desired properties.
- int Lgsx_ReaderExtractPointCloud (LgsxReaderPtr reader, char *inOutPathname)

  Extract the entire point cloud from the LGSx file and store it to the specified path/file.
- int Lgsx_ReaderPointCloudFileOpen (LgsxReaderPtr reader, PointCloudFileHandle *pHandle)

  Opens a virtual file handle to access the raw point cloud data.
- int Lgsx_PointCloudFileClose (PointCloudFileHandle *pHandle)

  Closes the Point Cloud file returned by Lgsx_ReaderPointCloudFileOpen().
- int Lgsx_PointCloudFileGetSize (const PointCloudFileHandle handle, uint64_t *pBytes)

  Get the size in bytes of the Point Cloud file.
- int Lgsx_PointCloudFileRead (const PointCloudFileHandle handle, uint64_t offset, uint64_t numBytes, void *pBuffer, uint64_t *pBytesRead)

  Read a chunk of raw Point Cloud file into the provided buffer.
- int Lgsx_ReaderGetClassModels (LgsxReaderPtr reader, wchar_t **ppClassModels)

  Retrieve the classification model names for this Point Cloud.
- int Lgsx_ReaderGetClassVisibles (LgsxReaderPtr reader, int modelIdx, wchar_t **ppClassNames, int **ppClassVisibles, int **ppClassColors)

  Retrieve the classification properties for the given classification model index.

6.1.16.1 Detailed Description

6.1.16.2 Function Documentation

6.1.16.2.1 Lgsx_EnumPointsConfigSetDesiredTypes()

```
int Lgsx_EnumPointsConfigSetDesiredTypes (
            const EnumPointsConfigHandle handle,
            int desiredTypes)
```

Set the desired property-type mask on an EnumPointsConfigHandle.

Parameters

in	*handle*	Configuration handle.
in	*desiredTypes*	Mask containing the properties to be retrieved during iteration.

Returns

Returns 0 for success.

6.1.16.2.2 Lgsx_EnumPointsConfigSetMaxMemoryHintBytes()

```
int Lgsx_EnumPointsConfigSetMaxMemoryHintBytes (
            const EnumPointsConfigHandle handle,
            uint64_t maxMemoryHintBytes)
```

Set the maximum memory hint on an EnumPointsConfigHandle.

Use this to cap the memory consumed by point enumeration, for example when other memory-intensive operations run in the same process.

Parameters

in	*handle*	Configuration handle.
in	*maxMemoryHintBytes*	Soft cap (bytes) on the memory used during point enumeration. Pass 0 to keep the default behaviour.

Returns

Returns 0 for success.

6.1.16.2.3 Lgsx_EnumPointsConfigSetStrictOrder()

```
int Lgsx_EnumPointsConfigSetStrictOrder (
            const EnumPointsConfigHandle handle,
            bool strictOrder)
```

Set the strict-order flag on an EnumPointsConfigHandle.

Parameters

in	*handle*	Configuration handle.
in	*strictOrder*	If true, the points are returned in stable order (if possible).

Returns

Returns 0 for success.

6.1.16.2.4 Lgsx_EnumPointsConfigSetSubSample()

```
int Lgsx_EnumPointsConfigSetSubSample (
            const EnumPointsConfigHandle handle,
            int subSample)
```

Set the sub-sample rate on an EnumPointsConfigHandle.

Parameters

in	*handle*	Configuration handle.
in	*subSample*	Not currently supported and should be set to 1.

Returns

Returns 0 for success.

6.1.16.2.5 Lgsx_EnumPointsConfigSetVisible()

```
int Lgsx_EnumPointsConfigSetVisible (
            const EnumPointsConfigHandle handle,
            bool visible)
```

Set the visibility filter on an EnumPointsConfigHandle.

Parameters

in	*handle*	Configuration handle.
in	*visible*	Limits collection to "visible" points when true. Otherwise collection includes clipped points.

Returns

Returns 0 for success.

6.1.16.2.6 Lgsx_InitEnumPointsConfig()

```
int Lgsx_InitEnumPointsConfig (
            EnumPointsConfigHandle * pHandle)
```

Allocate and initialise an EnumPointsConfigHandle with default values.

Default values: desiredTypes = 0, subSample = 1, visible = false, strictOrder = false, maxMemoryHintBytes = 0.

Parameters

out	*pHandle*	Pointer to the handle to initialise. Must be released with Lgsx_ReleaseEnumPointsConfig().

Returns

Returns 0 for success.

6.1.16.2.7 Lgsx_PointCloudFileClose()

```
int Lgsx_PointCloudFileClose (
            PointCloudFileHandle * pHandle)
```

Closes the Point Cloud file returned by Lgsx_ReaderPointCloudFileOpen().

Parameters

in	*pHandle*	Pointer to Point Cloud file returned by Lgsx_ReaderPointCloudFileOpen().

Returns

Returns 0 for success.

6.1.16.2.8 Lgsx_PointCloudFileGetSize()

```
int Lgsx_PointCloudFileGetSize (
            const PointCloudFileHandle handle,
            uint64_t * pBytes)
```

Get the size in bytes of the Point Cloud file.

Parameters

in	*handle*	Pointer to Point Cloud file returned by Lgsx_ReaderPointCloudFileOpen().
out	*pBytes*	Pointer to store the size of the Point Cloud file in bytes

Returns

Returns 0 for success.

6.1.16.2.9 Lgsx_PointCloudFileRead()

```
int Lgsx_PointCloudFileRead (
            const PointCloudFileHandle handle,
            uint64_t offset,
            uint64_t numBytes,
            void * pBuffer,
            uint64_t * pBytesRead)
```

Read a chunk of raw Point Cloud file into the provided buffer.

Parameters

in	*handle*	Pointer to Point Cloud file returned by Lgsx_ReaderPointCloudFileOpen().
in	*offset*	Offset in bytes from the beginning of the Point Cloud file.
in	*numBytes*	Number of bytes to read.
out	*pBuffer*	Buffer to store the read data.
out	*pBytesRead*	Pointer to store the number of bytes actually read.

Returns

Returns 0 for success.

6.1.16.2.10 Lgsx_ReaderEnumPoints()

```
int Lgsx_ReaderEnumPoints (
            LgsxReaderPtr reader,
            int desiredTypes,
            int subSample,
            bool visible)
```

Collects points in the Point Cloud.

Parameters

in	*reader*	Reader handle.
in	*desiredTypes*	Mask containing the properties to be retrieved during iteration.
in	*subSample*	Not surrently supported and should be set to 1.
in	*visible*	Limits collection to "visible" points when true. Otherwise collection includes clipped points.

Returns

Returns 0 for success.

6.1.16.2.11 Lgsx_ReaderEnumPointsEx()

```
int Lgsx_ReaderEnumPointsEx (
            LgsxReaderPtr reader,
            int desiredTypes,
            int subSample,
            bool visible,
            bool scrictOrder)
```

Collects points in the Point Cloud.

Parameters

in	*reader*	Reader handle.
in	*desiredTypes*	Mask containing the properties to be retrieved during iteration.
in	*subSample*	Not surrently supported and should be set to 1.
in	*visible*	Limits collection to "visible" points when true. Otherwise collection includes clipped points.
in	*scrictOrder*	If true, the points are returned in the stable order (if possible).

Returns

Returns 0 for success.

6.1.16.2.12 Lgsx_ReaderEnumPointsEx2()

```
int Lgsx_ReaderEnumPointsEx2 (
            LgsxReaderPtr reader,
            const EnumPointsConfigHandle config)
```

Collects points in the Point Cloud using the given configuration.

Parameters

in	*reader*	Reader handle.
in	*config*	Configuration handle initialised via Lgsx_InitEnumPointsConfig() and populated with the desired setter functions.

Returns

Returns 0 for success.

6.1.16.2.13 Lgsx_ReaderExtractPointCloud()

```
int Lgsx_ReaderExtractPointCloud (
            LgsxReaderPtr reader,
            char * inOutPathname)
```

Extract the entire point cloud from the LGSx file and store it to the specified path/file.

File format will be packed HSPC.

Warning

When inOutPathname is empty it will be used to store auto-generated name. The buffer must be at least 280 characters long.

Parameters

in	*reader*	Reader handle.
in,out	*inOutPathname*	Path at which to store the extracted file if not empty. Buffer to store path (if empty on function call).

Returns

Returns 0 for success.

6.1.16.2.14 Lgsx_ReaderGetClassModels()

```
int Lgsx_ReaderGetClassModels (
            LgsxReaderPtr reader,
            wchar_t ** ppClassModels)
```

Retrieve the classification model names for this Point Cloud.

Returns the number of model names.

Model name buffer is created by the SDK. Caller must free this buffer via LgsxUtil_FreeMem().

Parameters

in	*reader*	Reader handle.
out	*ppClassModels*	Classification model names. Names are null-terminated strings packed into a single buffer.

Returns

Returns the number of model names.

6.1.16.2.15 Lgsx_ReaderGetClassVisibles()

```
int Lgsx_ReaderGetClassVisibles (
            LgsxReaderPtr reader,
            int modelIdx,
            wchar_t ** ppClassNames,
            int ** ppClassVisibles,
            int ** ppClassColors)
```

Retrieve the classification properties for the given classification model index.

Classification properties are returned as arrays in buffers created by the SDK. Caller must free these buffers via LgsxUtil_FreeMem().

Parameters

in	*reader*	Reader handle.
in	*modelIdx*	Index of the model to retrieve.
out	*ppClassNames*	Array of classification name strings. Strings are null-terminated and packed into the returned buffer.
out	*ppClassVisibles*	Array of flags indicating whether the classification is visible by default.
out	*ppClassColors*	Array of colors for each classification. Each color is 4-byte RGBA format.

Returns

Returns the number of classifications in the requested model.

6.1.16.2.16 Lgsx_ReaderGetPropertyTypeInfo()

```
int Lgsx_ReaderGetPropertyTypeInfo (
            LgsxReaderPtr reader,
            int typeMask,
            const wchar_t * name,
            int * pnBytes)
```

Retrieve Property Type information from the Point Cloud.

Specifically, it returns the number of bytes used for the property being queried. This function fails if the requested property is not present in the Point Cloud. Only one property can be queried at a time.

Each Point Cloud contains a set of Data Properties. Basic properties include position, color, intensity, and point normal, etc. Point Clouds can also contain optional types: SetupIndex and Classification. Finally, a point cloud can have user-defined properties. This function allows the application to query for which optional and user-defined properties are present in the Point Cloud.

Parameters

in	*reader*	Reader handle.
in	*typeMask*	Property type to query (see Property type masks).
in	*name*	Property name being queried. Only used for user-defined types.
out	*pnBytes*	Number of bytes used for values of this property type.

Returns

Returns 0 for success.

6.1.16.2.17 Lgsx_ReaderMoveNextFields()

```
int Lgsx_ReaderMoveNextFields (
            LgsxReaderPtr reader,
            int chunkSize,
            int bmpFormat,
            void * ppData[])
```

Advances to the next group of points in the Point Cloud and returns the desired properties.

The desired properties are specified in the call to Lgsx_ReaderEnumPoints(). Caller passes in an array of buffer pointers. Each array index corresponds to one of the property masks. Individual property types can be skipped by setting the corresponding buffer pointer to null. Buffers need to be sized to the number of values specified by **chunkSize**.

Parameters

in	*reader*	Reader handle.
in	*chunkSize*	Number of points to read per call.
in	*bmpFormat*	Bitmap format to use for returned colors (BitmapFormat__RGB or BitmapFormat__RGBA)
out	*ppData*	Array of buffers to be filled with point data.

Returns

Returns number of points retrieved. Note that this can be less than the number of points requested. Return value of 0 indicates there are no more points to read.

6.1.16.2.18 Lgsx_ReaderMoveNextPoints()

```
int Lgsx_ReaderMoveNextPoints (
            LgsxReaderPtr reader,
            int chunkSize,
            int bmpFormat,
            double * points,
            float * intens,
            uchar * colors,
            float * normals)
```

Advances to the next group of points in the Point Cloud and returns the basic properties.

Parameters

in	*reader*	Reader handle.
in	*chunkSize*	Number of points to read per call.
in	*bmpFormat*	Bitmap format to use for returned colors (BitmapFormat__RGB or BitmapFormat__RGBA)
out	*points*	XYZ data.
out	*intens*	Scan Intensity data.
out	*colors*	Color data.
out	*normals*	Normal vectors.

Returns

Returns number of points retrieved. Note that this can be less than the number of points requested. Return value of 0 indicates there are no more points to read.

6.1.16.2.19 Lgsx_ReaderPointCloudFileOpen()

```
int Lgsx_ReaderPointCloudFileOpen (
            LgsxReaderPtr reader,
            PointCloudFileHandle * pHandle)
```

Opens a virtual file handle to access the raw point cloud data.

NOTE: The handle will only be returned if the point cloud exists and the type is packed HSPC.

Parameters

in	*reader*	Reader handle.
out	*pHandle*	Pointer to store pointer to Point Cloud file handle. Must be freed using Lgsx_PointCloudFileClose().

Returns

Returns 0 for success.

6.1.16.2.20 Lgsx_ReaderQueryPropertyTypes()

```
int Lgsx_ReaderQueryPropertyTypes (
            LgsxReaderPtr reader)
```

Returns the Property Types available for points in the project's Point Cloud.

Property types are defined in LgsxClient.h, and are things like XYZ, Intensity, Color, Normal, etc.

Parameters

in	*reader*	Reader handle.

Returns

Property type mask for this project.

6.1.16.2.21 Lgsx_ReleaseEnumPointsConfig()

```
int Lgsx_ReleaseEnumPointsConfig (
            EnumPointsConfigHandle * pHandle)
```

Release an EnumPointsConfigHandle previously allocated by Lgsx_InitEnumPointsConfig().

Parameters

in,out	*pHandle*	Pointer to the handle to release.

Returns

Returns 0 for success.

6.1.17 User Coordinate System Functions

Functions

- int Lgsx_ReaderEnumCoordSystems (LgsxReaderPtr reader, int ∗pNumCs)

  Collects User Coordinate Systems.
- int Lgsx_ReaderMoveNextCoordSystems (LgsxReaderPtr reader, bool ∗pActive, wchar_t ∗pName, int max↩ Name, wchar_t ∗pWkt, int maxWkt, SCyRgdBdyTxf ∗pTxf)

  Advances to the next User Coordinate System (UCS) object and returns its properties.
- int Lgsx_ReaderEndCoordSystems (LgsxReaderPtr reader)

  Frees the active User Coordinate Systems collection.
- int Lgsx_ReaderMoveNextCoordSystems2 (LgsxReaderPtr reader, bool ∗pActive, wchar_t ∗∗pNamePtr, wchar_t ∗∗pWktPtr, SCyRgdBdyTxf ∗pTxf)

  Advances to the next User Coordinate System (UCS) object and returns its properties.
- int Lgsx_ReaderGetCurrentCoordSystemId (LgsxReaderPtr reader, uint64_t ∗pId)

  Get the numeric ID of the User Coordinate System most recently returned by Lgsx_ReaderMoveNextCoordSystems2().
- int Lgsx_ReaderGetCurrentCoordSystemGuid (LgsxReaderPtr reader, char ∗∗pGuidPtr)

  Get the GUID of the User Coordinate System most recently returned by Lgsx_ReaderMoveNextCoordSystems2().

6.1.17.1 Detailed Description

6.1.17.2 Function Documentation

6.1.17.2.1 Lgsx_ReaderEndCoordSystems()

```
int Lgsx_ReaderEndCoordSystems (
            LgsxReaderPtr reader)
```

Frees the active User Coordinate Systems collection.

Parameters

in	*reader*	Reader handle.

Returns

Returns 0 for success.

6.1.17.2.2 Lgsx_ReaderEnumCoordSystems()

```
int Lgsx_ReaderEnumCoordSystems (
            LgsxReaderPtr reader,
            int * pNumCs)
```

Collects User Coordinate Systems.

Parameters

in	*reader*	Reader handle.
out	*pNumCs*	Number of User Coordinate Systems.

Returns

Returns 0 for success.

6.1.17.2.3 Lgsx_ReaderGetCurrentCoordSystemGuid()

```
int Lgsx_ReaderGetCurrentCoordSystemGuid (
            LgsxReaderPtr reader,
            char ** pGuidPtr)
```

Get the GUID of the User Coordinate System most recently returned by Lgsx_ReaderMoveNextCoordSystems2().

Parameters

in	*reader*	Reader handle.
out	*pGuidPtr*	Pointer to store pointer to the GUID string. Buffer must be freed using LgsxUtil_FreeMem().

Returns

Returns 0 for success.

6.1.17.2.4 Lgsx_ReaderGetCurrentCoordSystemId()

```
int Lgsx_ReaderGetCurrentCoordSystemId (
            LgsxReaderPtr reader,
            uint64_t * pId)
```

Get the numeric ID of the User Coordinate System most recently returned by Lgsx_ReaderMoveNextCoordSystems2().

Parameters

in	*reader*	Reader handle.
out	*pId*	Pointer to store the UCS numeric ID.

Returns

Returns 0 for success.

6.1.17.2.5 Lgsx_ReaderMoveNextCoordSystems()

```
int Lgsx_ReaderMoveNextCoordSystems (
            LgsxReaderPtr reader,
            bool * pActive,
            wchar_t * pName,
            int maxName,
            wchar_t * pWkt,
            int maxWkt,
            SCyRgdBdyTxf * pTxf)
```

Advances to the next User Coordinate System (UCS) object and returns its properties.

Well Known Text (WKT) is a standard format for specifying Coordinate Reference Systems (e.g, state planes, UTM, etc). You would include a WKT string in a UCS to give a project real-world position information. If the WKT string is empty, then the coordinates are localized Cartesian coordinates.

See also

Lgsx_ReaderMoveNextCoordSystems2() is a version that always returns complete data without truncation.

Parameters

in	*reader*	Reader handle.
out	*pActive*	Active flag, set to true if this UCS is active.
out	*pName*	UCS name.
in	*maxName*	Max size of name buffer.
out	*pWkt*	WKT string for this UCS.
in	*maxWkt*	Max size of WKT buffer.
out	*pTxf*	Transform/rotation for this UCS.

Returns

Returns 0 for success.

6.1.17.2.6 Lgsx_ReaderMoveNextCoordSystems2()

```
int Lgsx_ReaderMoveNextCoordSystems2 (
            LgsxReaderPtr reader,
            bool * pActive,
            wchar_t ** pNamePtr,
            wchar_t ** pWktPtr,
            SCyRgdBdyTxf * pTxf)
```

Advances to the next User Coordinate System (UCS) object and returns its properties.

Well Known Text (WKT) is a standard format for specifying Coordinate Reference Systems (e.g, state planes, UTM, etc). You would include a WKT string in a UCS to give a project real-world position information. If the WKT string is empty, then the coordinates are localized Cartesian coordinates.

Parameters

in	*reader*	Reader handle.
out	*pActive*	Active flag, set to true if this UCS is active.
out	*pNamePtr*	Pointer to store pointer to UCS name. Buffer must be freed using LgsxUtil_FreeMem().
out	*pWktPtr*	Pointer to store pointer to WKT string for this UCS. Buffer must be freed using LgsxUtil_FreeMem().
out	*pTxf*	Transform/rotation for this UCS.

Returns

Returns 0 for success.

6.2 General Functions

General convenience functions.

Functions

- const wchar_t * LgsxUtil_GetCurrentTimeString ()

  Convenience function: Return current local time formatted as "HHMMSS".
- const wchar_t * LgsxUtil_GetCurrentDateString ()

  Convenience function: Return local date formatted as "YYMMDD".
- void * LgsxUtil_TimeStart ()

  Create a new timer and starts it.
- double LgsxUtil_TimeGetLapse (void *pTime)

  Returns elapsed time for the given timer in seconds.
- const wchar_t * LgsxUtil_TimeGetLapseStr (void *pTime, int format)

  Returns elapsed time for the given timer in seconds as a string.
- int LgsxUtil_TimeEnd (void **pTime)

  Frees the timer object.
- int LgsxUtil_Sleep (int millisec)

  Suspends execution for the given number of milliseconds.
- void * LgsxUtil_AllocMem (int size)

  Allocate a block of memory with the given size and return a pointer to it.
- void LgsxUtil_FreeMem (void **ptr)

  Free a block of memory previously created by the SDK.
- int LgsxUtil_StrDupA (const char *value, char **ptrDupValue)

  Duplicate string.
- int LgsxUtil_StrDupW (const wchar_t *value, wchar_t **ptrDupValue)

  Duplicate string.
- int LgsxUtil_A2W (const char *value, wchar_t *tValue)

  Convert UTF8 encoded text to UTF16.
- int LgsxUtil_A2WEx (const char *value, wchar_t *tValue, int maxLen)

  Convert UTF8 encoded text to UTF16 into a bounded output buffer.
- int LgsxUtil_AllocA2W (const char *value, wchar_t **pTValuePtr)

  Convert UTF8 encoded text to UTF16 and allocate output buffer for the whole converted string.
- int LgsxUtil_W2A (const wchar_t *value, char *sValue)

Convert UTF16 encoded text to UTF8.

- int LgsxUtil_W2AEx (const wchar_t *value, char *sValue, int maxLen)

  Converts a wide string to a UTF-8 encoded string with buffer size limit.
- int LgsxUtil_AllocW2A (const wchar_t *value, char **pSValuePtr)

  Convert UTF16 encoded text to UTF8 and allocate output buffer for the whole converted string.
- void Lgsx_FreeHandle (void *handle)

  Frees the SDK object referenced by the given handle.
- const wchar_t * Lgsx_GetLastError ()

  Returns error string for most recent SDK error.
- int Lgsx_GetLastErrorNo ()

  Returns error number for most recent SDK error.
- int Lgsx_GetProductVersion (wchar_t *version, long *buildNum)

  Get LGSx SDK application's version and build number.
- int Lgsx_GetProductVersion2 (wchar_t **pVersionPtr, long *buildNum)

  Get LGSx SDK application's version and build number, without buffer truncation.
- int Lgsx_InitProductVersion (const wchar_t *customVersion)

  Initialize LGSx SDK application's version string.
- int Lgsx_InitProductVersionEx (const wchar_t *customVersion, const wchar_t *fileVersion)

  Initialize LGSx SDK application's version string and file version.
- int Lgsx_InitProductName (const wchar_t *prodName)

  Define the name of the product you want to display in About dialog.
- int Lgsx_InitProductBuildNumber (int build)

  Initialize LGSx SDK application's build number.

6.2.1 Detailed Description

General convenience functions.

Methods in this class are actually global C APIs.

6.2.2 Function Documentation

6.2.2.1 Lgsx_FreeHandle()

```
void Lgsx_FreeHandle (
            void * handle)
```

Frees the SDK object referenced by the given handle.

Parameters

in	*handle*	Handle to free.

6.2.2.2 Lgsx_GetLastError()

```
const wchar_t * Lgsx_GetLastError ()
```

Returns error string for most recent SDK error.

Returns

String containing error message. Caller does **not** need to free string memory.

6.2.2.3 Lgsx_GetLastErrorNo()

```
int Lgsx_GetLastErrorNo ()
```

Returns error number for most recent SDK error.

Returns

Error number.

6.2.2.4 Lgsx_GetProductVersion()

```
int Lgsx_GetProductVersion (
            wchar_t * version,
            long * buildNum)
```

Get LGSx SDK application's version and build number.

The version string and build number are displayed in About dialog.

Parameters

out	*version*	Returned version string. Need to pre-allocate at least 40 characters.
out	*buildNum*	Returned build number.

Returns

0 for success.

6.2.2.5 Lgsx_GetProductVersion2()

```
int Lgsx_GetProductVersion2 (
            wchar_t ** pVersionPtr,
            long * buildNum)
```

Get LGSx SDK application's version and build number, without buffer truncation.

See also

Lgsx_GetProductVersion() for the truncating alternative that uses a fixed-size buffer.

Parameters

out	*pVersionPtr*	Pointer to store pointer to the version string. Buffer must be freed using LgsxUtil_FreeMem(). May be NULL.
out	*buildNum*	Returned build number. May be NULL.

Returns

0 for success.

6.2.2.6 Lgsx_InitProductBuildNumber()

```
int Lgsx_InitProductBuildNumber (
            int build)
```

Initialize LGSx SDK application's build number.

The version string and build number are displayed in About dialog.

Parameters

in	*build*	Build number.

Returns

0 for success.

6.2.2.7 Lgsx_InitProductName()

```
int Lgsx_InitProductName (
            const wchar_t * prodName)
```

Define the name of the product you want to display in About dialog.

Parameters

in	*prodName*	Product name

Returns

0 for success.

6.2.2.8 Lgsx_InitProductVersion()

```
int Lgsx_InitProductVersion (
            const wchar_t * customVersion)
```

Initialize LGSx SDK application's version string.

The version string and build number are displayed in About dialog.

Parameters

in	*customVersion*	Custom version string, such as "2.1A".

Returns

0 for success.

6.2.2.9 Lgsx_InitProductVersionEx()

```
int Lgsx_InitProductVersionEx (
            const wchar_t * customVersion,
            const wchar_t * fileVersion)
```

Initialize LGSx SDK application's version string and file version.

The version string and build number are displayed in About dialog.

Parameters

in	*customVersion*	Custom version string, such as "2.1A".
in	*fileVersion*	Version string to be displayed in module/dll.

Returns

0 for success.

6.2.2.10 LgsxUtil_A2W()

```
int LgsxUtil_A2W (
            const char * value,
            wchar_t * tValue)
```

Convert UTF8 encoded text to UTF16.

Deprecated Legacy function without buffer size limit. Use LgsxUtil_A2WEx() instead.

Parameters

in	*value*	Source string (UTF-8).
out	*tValue*	Output buffer for converted wide string.

Returns

0 for success, negative on error (e.g. NULL input).

6.2.2.11 LgsxUtil_A2WEx()

```
int LgsxUtil_A2WEx (
            const char * value,
            wchar_t * tValue,
            int maxLen)
```

Convert UTF8 encoded text to UTF16 into a bounded output buffer.

For UTF-8 to wide conversion the output length is always <= the input byte length, so passing strlen(value)+1 as maxLen is always sufficient.

Parameters

in	*value*	Source string (UTF-8).
out	*tValue*	Output buffer for converted wide string.
in	*maxLen*	Capacity of tValue in wchar_t elements.

Returns

0 for success, 1 when output was truncated, negative values on invalid input.

6.2.2.12 LgsxUtil_AllocA2W()

```
int LgsxUtil_AllocA2W (
            const char * value,
            wchar_t ** pTValuePtr)
```

Convert UTF8 encoded text to UTF16 and allocate output buffer for the whole converted string.

Parameters

in	*value*	Source string (UTF-8).
out	*pTValuePtr*	Pointer to store output buffer pointer. Buffer must be freed using LgsxUtil_FreeMem().

Returns

0 for success, negative on error.

6.2.2.13 LgsxUtil_AllocMem()

```
void * LgsxUtil_AllocMem (
            int size)
```

Allocate a block of memory with the given size and return a pointer to it.

Parameters

in	*size*	Size of memory to create, in bytes.

Returns

Pointer to allocated memory.

6.2.2.14 LgsxUtil_AllocW2A()

```
int LgsxUtil_AllocW2A (
            const wchar_t * value,
            char ** pSValuePtr)
```

Convert UTF16 encoded text to UTF8 and allocate output buffer for the whole converted string.

Parameters

in	*value*	Source wide string.
out	*pSValuePtr*	Pointer to store output buffer pointer. Buffer must be freed using LgsxUtil_FreeMem().

Returns

0 for success, negative on error.

6.2.2.15 LgsxUtil_FreeMem()

```
void LgsxUtil_FreeMem (
            void ** ptr)
```

Free a block of memory previously created by the SDK.

Parameters

in	*ptr*	Pointer to memory to be freed.

6.2.2.16 LgsxUtil_GetCurrentDateString()

```
const wchar_t * LgsxUtil_GetCurrentDateString ()
```

Convenience function: Return local date formatted as "YYMMDD".

Range of years is 0 to 99. Range of months is 1 to 12. Range of days is 1 to 31.

Returns

String pointer to formatted date. String memory is managed by the SDK.

6.2.2.17 LgsxUtil_GetCurrentTimeString()

```
const wchar_t * LgsxUtil_GetCurrentTimeString ()
```

Convenience function: Return current local time formatted as "HHMMSS".

Hours are in 24 hour format (0 to 23).

Returns

String pointer to formatted local time. String memory is managed by the SDK.

6.2.2.18 LgsxUtil_Sleep()

```
int LgsxUtil_Sleep (
            int millisec)
```

Suspends execution for the given number of milliseconds.

Parameters

in	*millisec*	Number of milliseconds to sleep.

Returns

0 for success.

6.2.2.19 LgsxUtil_StrDupA()

```
int LgsxUtil_StrDupA (
            const char * value,
            char ** ptrDupValue)
```

Duplicate string.

The duplicated string must be freed using LgsxUtil_FreeMem().

Parameters

in	*value*	Source string. If NULL, the function will return error.
out	*ptrDupValue*	Pointer to store pointer to duplicated string.

Returns

0 for success.

6.2.2.20 LgsxUtil_StrDupW()

```
int LgsxUtil_StrDupW (
            const wchar_t * value,
            wchar_t ** ptrDupValue)
```

Duplicate string.

The duplicated string must be freed using LgsxUtil_FreeMem().

Parameters

in	*value*	Source string. If NULL, the function will return error.
out	*ptrDupValue*	Pointer to store pointer to duplicated string.

Returns

0 for success.

6.2.2.21 LgsxUtil_TimeEnd()

```
int LgsxUtil_TimeEnd (
            void ** pTime)
```

Frees the timer object.

Parameters

in	*pTime*	Timer object handle.

Returns

0 for success.

6.2.2.22 LgsxUtil_TimeGetLapse()

```
double LgsxUtil_TimeGetLapse (
            void * pTime)
```

Returns elapsed time for the given timer in seconds.

Parameters

in	*pTime*	Timer object handle.

Returns

Elapsed time in seconds.

6.2.2.23 LgsxUtil_TimeGetLapseStr()

```
const wchar_t * LgsxUtil_TimeGetLapseStr (
            void * pTime,
            int format)
```

Returns elapsed time for the given timer in seconds as a string.

Parameters

in	*pTime*	Timer object handle.
in	*format*	String format to return. 0 returns time string in seconds. 1 returns time as **hh:mm:ss**, where **hh** is two-digit hours, **mm** is two-digit minutes, and **ss** is seconds.

Returns

Buffer containing time string. Buffer must be freed by the caller.

6.2.2.24 LgsxUtil_TimeStart()

```
void * LgsxUtil_TimeStart ()
```

Create a new timer and starts it.

Returns

Handle for the new Timer object.

6.2.2.25 LgsxUtil_W2A()

```
int LgsxUtil_W2A (
            const wchar_t * value,
            char * sValue)
```

Convert UTF16 encoded text to UTF8.

Deprecated Legacy function without buffer size limit. Use LgsxUtil_W2AEx() instead.

Parameters

in	*value*	Source wide string.
out	*sValue*	Converted UTF-8 string.

Returns

0 for success, negative on error (e.g. NULL input).

6.2.2.26 LgsxUtil_W2AEx()

```
int LgsxUtil_W2AEx (
            const wchar_t * value,
            char * sValue,
            int maxLen)
```

Converts a wide string to a UTF-8 encoded string with buffer size limit.

Parameters

in	*value*	Source wide string.
out	*sValue*	Converted UTF-8 string.
in	*maxLen*	Maximum buffer size including null terminator.

Returns

0 for success, 1 if truncated, negative on error.

6.3 Application Functions

Functions that all SDK Applications will need to use.

Functions

- int Lgsx_Initialize (const wchar_t *pTempPath, bool enableLog)
  Initialize the LGSx SDK APIs.
- int Lgsx_Uninitialize ()
  Unintialize LGSx SDK APIs.

6.3.1 Detailed Description

Functions that all SDK Applications will need to use.

6.3.2 Function Documentation

6.3.2.1 Lgsx_Initialize()

```
int Lgsx_Initialize (
            const wchar_t * pTempPath,
            bool enableLog)
```

Initialize the LGSx SDK APIs.

This should be the first call before checking license or loading a project.

Parameters

in	*pTempPath*	Path to folder to be used for temporary files used by the SDK.
in	*enableLog*	Set to TRUE to enable the SDK's logging/log file. Logging is disabled if set to FALSE.

Returns

0 for success.

6.3.2.2 Lgsx_Uninitialize()

```
int Lgsx_Uninitialize ()
```

Unintialize LGSx SDK APIs.

This should always be called before exiting from your application. It is not recommended to call the pair of Lgsx_↩ Initialize and Lgsx_Uninitialize multiple times during a session.

Returns

0 for success.

6.4 Licensing Functions

Functions

- int Lgsx_InitLicense (const wchar_t *lic, const wchar_t *ver)
  Initialize the license to be loaded.
- int Lgsx_InitLicenseEx (const wchar_t *lic, const wchar_t *ver, const wchar_t *product)
  Initialize the license to be loaded.
- int Lgsx_InitLicenseEx2 (const wchar_t *lic, const wchar_t *ult, const wchar_t *ver, const wchar_t *product)
  Initialize the license to be loaded.
- int Lgsx_IsLicensed ()
  Check if the license is valid.
- int Lgsx_LicenseDaysLeft ()
  Check the number of days left for the license.
- int Lgsx_LoadLicense ()
  Load license and hold the usage.
- int Lgsx_UnloadLicense ()
  Unload license.
- int Lgsx_IsLicensedExA (const char *lic, const char *ver)
  Check if a named license is valid.
- int Lgsx_LicenseDaysLeftExA (const char *lic, const char *ver)
  Check the number of days left for given license.
- int Lgsx_LoadLicenseExA (const char *lic, const char *ver)
  Load given license and hold usage until UnloadLicense.
- int Lgsx_UnloadLicenseExA (const char *lic)
  Unload a named license.

6.4.1 Detailed Description

6.4.2 Function Documentation

6.4.2.1 Lgsx_InitLicense()

```
int Lgsx_InitLicense (
            const wchar_t * lic,
            const wchar_t * ver)
```

Initialize the license to be loaded.

This must be called before opening a LGSx file or other functions that require an SDK license. Opening LGSx file for reading is free when SDK license is valid.

Following functions will only be unlocked when a license check is successful:

- Lgsx_ReaderOpen

Parameters

in	*lic*	The name string of the license.
in	*ver*	The version string of the license.

Returns

0 for success.

6.4.2.2 Lgsx_InitLicenseEx()

```
int Lgsx_InitLicenseEx (
            const wchar_t * lic,
            const wchar_t * ver,
            const wchar_t * product)
```

Initialize the license to be loaded.

This must be called before opening a LGSx file or other functions that require a license. This is an extended version of Lgsx_InitLicense().

Parameters

in	*lic*	The name string of the license.
in	*ver*	The version string of the license.
in	*product*	The product name.

Returns

0 for success.

6.4.2.3 Lgsx_InitLicenseEx2()

```
int Lgsx_InitLicenseEx2 (
            const wchar_t * lic,
            const wchar_t * ult,
            const wchar_t * ver,
            const wchar_t * product)
```

Initialize the license to be loaded.

This must be called before opening a LGSx File or other functions that require a license. This is an extended version of Lgsx_InitLicense().

Parameters

in	*lic*	The name string of the license.
in	*ult*	The name string of ultimate license (or empty to not support ultimate).
in	*ver*	The version string of the license.
in	*product*	The product name.

Returns

0 for success.

6.4.2.4 Lgsx_IsLicensed()

```
int Lgsx_IsLicensed ()
```

Check if the license is valid.

Returns

1 means it is licensed.

6.4.2.5 Lgsx_IsLicensedExA()

```
int Lgsx_IsLicensedExA (
            const char * lic,
            const char * ver)
```

Check if a named license is valid.

Parameters

in	*lic*	The name string of the license.
in	*ver*	The version string of the license.

Returns

1 means it is licensed.

6.4.2.6 Lgsx_LicenseDaysLeft()

```
int Lgsx_LicenseDaysLeft ()
```

Check the number of days left for the license.

Returns

The number of days.

6.4.2.7 Lgsx_LicenseDaysLeftExA()

```
int Lgsx_LicenseDaysLeftExA (
            const char * lic,
            const char * ver)
```

Check the number of days left for given license.

Parameters

in	*lic*	The name string of the license.
in	*ver*	The version string of the license.

Returns

The number of days.

6.4.2.8 Lgsx_LoadLicense()

```
int Lgsx_LoadLicense ()
```

Load license and hold the usage.

Returns

0 for success.

6.4.2.9 Lgsx_LoadLicenseExA()

```
int Lgsx_LoadLicenseExA (
            const char * lic,
            const char * ver)
```

Load given license and hold usage until UnloadLicense.

Parameters

in	*lic*	The name string of the license.
in	*ver*	The version string of the license.

Returns

0 for success.

6.4.2.10 Lgsx_UnloadLicense()

```
int Lgsx_UnloadLicense ()
```

Unload license.

Returns

0 for success.

6.4.2.11 Lgsx_UnloadLicenseExA()

```
int Lgsx_UnloadLicenseExA (
            const char * lic)
```

Unload a named license.

Parameters

in	*lic*	The name string of the license.

Returns

0 for success.

6.5 Metadata Functions

Functions for reading, writing, and parsing metadata objects.

Functions

- LgsxMetadataPtr Lgsx_MetaCreateInstance ()
  Creates an instance of metadata container.
- LgsxMetadataPtr Lgsx_MetaClone (LgsxMetadataPtr pMetadata)
  Clones a metadata container.
- int Lgsx_MetaRemove (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName)
  Remove a key pair from metadata.
- int Lgsx_MetaHas (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName)
  TRUE if the key exists in the metadata.
- int Lgsx_MetaReset (LgsxMetadataPtr pMetadata)
  Resets traverse pointer in the metadata.
- int Lgsx_MetaMoveNext (LgsxMetadataPtr pMetadata, LgsxPropertyName pName, int ∗pType)
  Move to next key pair of the metadata.
- int Lgsx_MetaIsEmpty (LgsxMetadataPtr pMetadata, bool ∗pIsEmpty)
  Check if a metadata container is empty (has no key-value pairs).
- int Lgsx_MetaMoveNext2 (LgsxMetadataPtr pMetadata, const char ∗∗ppName, int ∗pType)
  Move to next key pair of the metadata.
- int Lgsx_MetaGetBool (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, bool ∗pValue)

Get the value associated with the given name in metadata.

- int Lgsx_MetaGetInt (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, int *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetInt64 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, __int64 *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetDouble (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetString (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, wchar_t *pValue, int maxLength)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetString2 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, wchar_t **pp↩ Value)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetDouble3 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetDouble4 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetMetadata (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, LgsxMetadataPtr *pMetadataOut)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetDoubleArray (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *p↩ Values, int *pCount)

  Get the double array value associated with the given name in metadata.
- int Lgsx_MetaGetDoubleArraySize (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, int *p↩ Count)

  Get the size of a double array associated with the given name in metadata.
- int Lgsx_MetaAddBool (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, bool value)

  Add name value pair to metadata.
- int Lgsx_MetaAddInt (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, int value)

  Add name value pair to metadata.
- int Lgsx_MetaAddInt64 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, __int64 value)

  Add name value pair to metadata.
- int Lgsx_MetaAddDouble (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double value)

  Add name value pair to metadata.
- int Lgsx_MetaAddString (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, const wchar_t *p↩ Value)

  Add name value pair to metadata.
- int Lgsx_MetaAddDouble3 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, const double *pValue)

  Add name value pair to metadata.
- int Lgsx_MetaAddDouble4 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, const double *pValue)

  Add name value pair to metadata.
- int Lgsx_MetaAddMetadata (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, LgsxMetadataPtr pMetadataIn)

  Add name value pair to metadata.

6.5.1 Detailed Description

Functions for reading, writing, and parsing metadata objects.

A metadata object is a property container that holds a set of key-value pairs. Key is a unique name (should use less than 40 chars), and value can be of different types. Metadata objects can be recursive - a property value of a metadata object can be another metadata object.

6.5.2 Function Documentation

6.5.2.1 Lgsx_MetaAddBool()

```
int Lgsx_MetaAddBool (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            bool value)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*value*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.2 Lgsx_MetaAddDouble()

```
int Lgsx_MetaAddDouble (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            double value)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*value*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.3 Lgsx_MetaAddDouble3()

```
int Lgsx_MetaAddDouble3 (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            const double * pValue)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*pValue*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.4 Lgsx_MetaAddDouble4()

```
int Lgsx_MetaAddDouble4 (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            const double * pValue)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*pValue*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.5 Lgsx_MetaAddInt()

```
int Lgsx_MetaAddInt (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            int value)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*value*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.6 Lgsx_MetaAddInt64()

```
int Lgsx_MetaAddInt64 (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            __int64 value)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*value*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.7 Lgsx_MetaAddMetadata()

```
int Lgsx_MetaAddMetadata (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            LgsxMetadataPtr pMetadataIn)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*pMetadataIn*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.8 Lgsx_MetaAddString()

```
int Lgsx_MetaAddString (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            const wchar_t * pValue)
```

Add name value pair to metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
in	*pValue*	Value to add. Be careful that the type of value is important.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.9 Lgsx_MetaClone()

```
LgsxMetadataPtr Lgsx_MetaClone (
            LgsxMetadataPtr pMetadata)
```

Clones a metadata container.

Parameters

in	*pMetadata*	Metadata object to be cloned.

Returns

Metadata object. You need to delete it using Lgsx_FreeHandle.

6.5.2.10 Lgsx_MetaCreateInstance()

```
LgsxMetadataPtr Lgsx_MetaCreateInstance ()
```

Creates an instance of metadata container.

Returns

Metadata object. You need to delete it using Lgsx_FreeHandle.

6.5.2.11 Lgsx_MetaGetBool()

```
int Lgsx_MetaGetBool (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            bool * pValue)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValue*	Returned value.

Note

The pName may be of any length (const char ∗), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.12 Lgsx_MetaGetDouble()

```
int Lgsx_MetaGetDouble (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            double * pValue)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValue*	Returned value.

Note

The pName may be of any length (const char ∗), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.13 Lgsx_MetaGetDouble3()

```
int Lgsx_MetaGetDouble3 (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            double * pValue)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValue*	Returned value. Need to pre-allocate 3 doubles for returning data.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.14 Lgsx_MetaGetDouble4()

```
int Lgsx_MetaGetDouble4 (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            double * pValue)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValue*	Returned value. Need to pre-allocate 4 doubles for returning data.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.15 Lgsx_MetaGetDoubleArray()

```
int Lgsx_MetaGetDoubleArray (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            double * pValues,
            int * pCount)
```

Get the double array value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValues*	Buffer to receive the double values. Must be pre-allocated by the caller.
in,out	*pCount*	On input, the maximum number of doubles the buffer can hold. On output, the actual number of doubles written to the buffer.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.16 Lgsx_MetaGetDoubleArraySize()

```
int Lgsx_MetaGetDoubleArraySize (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            int * pCount)
```

Get the size of a double array associated with the given name in metadata.

Use this function to determine the required buffer size before calling Lgsx_MetaGetDoubleArray(). A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pCount*	The number of double values in the array.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.17 Lgsx_MetaGetInt()

```
int Lgsx_MetaGetInt (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            int * pValue)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValue*	Returned value.

Note

The pName may be of any length (const char ∗), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.18 Lgsx_MetaGetInt64()

```
int Lgsx_MetaGetInt64 (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            __int64 * pValue)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValue*	Returned value.

Note

The pName may be of any length (const char ∗), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.19 Lgsx_MetaGetMetadata()

```
int Lgsx_MetaGetMetadata (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            LgsxMetadataPtr * pMetadataOut)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pMetadataOut*	Returned value which is of metadata type.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, otherwise failed.

6.5.2.20 Lgsx_MetaGetString()

```
int Lgsx_MetaGetString (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            wchar_t * pValue,
            int maxLength)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible. If you get a positive value, it indicates that the allocated maxLength is not big enough for the value. In this case, the returned value is the actual length. You may call it again with a bigger buffer (at lease the actual length plus 1).

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*pValue*	Buffer to store value.
in	*maxLength*	Size of buffer.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, negative failed, positive value is the actual length which indicates value is longer than allocated maxLength.

6.5.2.21 Lgsx_MetaGetString2()

```
int Lgsx_MetaGetString2 (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName,
            wchar_t ** ppValue)
```

Get the value associated with the given name in metadata.

A negative returned value means the name does not exist or the data type is not compatible.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.
out	*ppValue*	Pointer to store pointer to buffer with value. Buffer must be freed using LgsxUtil_FreeMem().

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 as success, negative failed.

6.5.2.22 Lgsx_MetaHas()

```
int Lgsx_MetaHas (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName)
```

TRUE if the key exists in the metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.

Note

The pName may be of any length, Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

1 as true and 0 as false.

6.5.2.23 Lgsx_MetaIsEmpty()

```
int Lgsx_MetaIsEmpty (
            LgsxMetadataPtr pMetadata,
            bool * pIsEmpty)
```

Check if a metadata container is empty (has no key-value pairs).

Parameters

in	*pMetadata*	Metadata container object.
out	*pIsEmpty*	Pointer to store the result. Set to true if the metadata container has no key-value pairs, false otherwise.

Returns

0 for success, otherwise failed.

6.5.2.24 Lgsx_MetaMoveNext()

```
int Lgsx_MetaMoveNext (
            LgsxMetadataPtr pMetadata,
            LgsxPropertyName pName,
            int * pType)
```

Move to next key pair of the metadata.

Note that we have not implemented API to get complex data (arbitrary dimensions).

MetadataType	Value
MetadataType_BOOL	0
MetadataType_INT	1
MetadataType_INT64	2
MetadataType_DOUBLE	3
MetadataType_STRING	4
MetadataType_DOUBLE3D	5
MetadataType_QUAT4D	6
MetadataType_METADATA	7
MetadataType_COMPLEX	8

Parameters

in	*pMetadata*	Metadata container object.
out	*pName*	Returned key name. Please reserve 40 chars for returning string.
out	*pType*	Returned value type. See table for more detail of the returned type.

Deprecated 09.07.2025 - Lgsx_MetaMoveNext2 introduced that does not truncate the name.

Returns

0 for success. None-zero means end.

6.5.2.25 Lgsx_MetaMoveNext2()

```
int Lgsx_MetaMoveNext2 (
            LgsxMetadataPtr pMetadata,
            const char ** ppName,
            int * pType)
```

Move to next key pair of the metadata.

Note that we have not implemented API to get complex data (arbitrary dimensions).

MetadataType	Value
MetadataType_BOOL	0
MetadataType_INT	1
MetadataType_INT64	2

MetadataType_DOUBLE	3
MetadataType_STRING	4
MetadataType_DOUBLE3D	5
MetadataType_QUAT4D	6
MetadataType_METADATA	7
MetadataType_COMPLEX	8

Parameters

in	*pMetadata*	Metadata container object.
out	*ppName*	Returned key name. The pointer remains valid until next call to Lgsx_MetaMoveNext2 or Lgsx_MetaReset.
out	*pType*	Returned value type. See table for more detail of the returned type.

Returns

0 for success. None-zero means end.

6.5.2.26 Lgsx_MetaRemove()

```
int Lgsx_MetaRemove (
            LgsxMetadataPtr pMetadata,
            const LgsxPropertyName pName)
```

Remove a key pair from metadata.

Parameters

in	*pMetadata*	Metadata container object.
in	*pName*	Key name.

Note

The pName may be of any length (const char *), Lgsx_MetaMoveNext2() allows getting non-truncated names.

Returns

0 for success.

6.5.2.27 Lgsx_MetaReset()

```
int Lgsx_MetaReset (
            LgsxMetadataPtr pMetadata)
```

Resets traverse pointer in the metadata.

This is called before calling Lgsx_MetaMoveNext/Lgsx_MetaMoveNext2.

Parameters

in	*pMetadata*	Metadata container object.

Returns

0 for success.

Chapter 7

Data Structure Documentation

7.1 AssetHandle Struct Reference

Asset data handle for accessing Asset data without buffer truncation.

`#include <LgsxReader.h>`

7.1.1 Detailed Description

Asset data handle for accessing Asset data without buffer truncation.

7.2 CategoryEntryHandle Struct Reference

Descriptor for a Category Entry object.

`#include <LgsxReader.h>`

7.2.1 Detailed Description

Descriptor for a Category Entry object.

7.3 CategoryHandle Struct Reference

Descriptor for a Category object.

`#include <LgsxReader.h>`

7.3.1 Detailed Description

Descriptor for a Category object.

7.4 CategoryValueHandle Struct Reference

Descriptor for a Category Value object.

`#include <LgsxReader.h>`

7.4.1 Detailed Description

Descriptor for a Category Value object.

7.5 CYASSET Struct Reference

Structure for Asset.

`#include <LgsxClient.h>`

Data Fields

- wchar_t name [80]
  User-facing name of the Asset.
- wchar_t type [40]
  Asset type.
- wchar_t **mimeType** [60]
  A mime type of the Asset.
- wchar_t filename [256]
  A temporary file name in which the asset data is stored.
- wchar_t url [256]
  The url address to the external resource.
- uint64_t **id**
  The unique indentifier of the Asset.
- uint64_t **creationTime**
  Creation timestamp (UTC seconds since 1/1/1970).
- uint64_t **modificationTime**
  Modification timestamp (UTC seconds since 1/1/1970).
- bool **isExternalUrl**
  Marks an asset as external url.

7.5.1 Detailed Description

Structure for Asset.

7.5.2 Field Documentation

7.5.2.1 filename

```
wchar_t CYASSET::filename[256]
```

A temporary file name in which the asset data is stored.

You need to process it immediately as next call to get asset will cause the file unlinked.

7.5.2.2 name

```
wchar_t CYASSET::name[80]
```

User-facing name of the Asset.

In case of file Asset, the name may imply the original file name, which contains file extension about the type of data stored. For example, a model file asset may be of DXF, COE, OBJ, IFC types. Document file asset can be PDF or other types.

A file asset can also be of image types.

7.5.2.3 type

```
wchar_t CYASSET::type[40]
```

Asset type.

One of the following names.

Value	Meaning
"Image"	The blob is an image that you may use Lgsx_JsGetAssetAsImage (may fail of unsupported image).
"BackgroundImage"	Same as Image but it is being used by one or more Sitemaps as a background image.
"Slice"	Same as Image, but it was made from a snapshot of a slice.
"View"	Same as Image, but it was made from a snapshot (screen copy).
"File"	The blob can be any file type (PDF, geometry model types, or image, implied by its file extension).

7.5.2.4 url

```
wchar_t CYASSET::url[256]
```

The url address to the external resource.

Should be filled only when isExternalUrl is true.

7.6 CYBBOX Struct Reference

Structure for bounding box (min, max) corners.

`#include <LgsxClient.h>`

Data Fields

- PNT3D **minCorner**
  Corner with smaller coordinate values.
- PNT3D **maxCorner**
  Corner with bigger coordinate values.

7.6.1 Detailed Description

Structure for bounding box (min, max) corners.

7.7 CYGEOTAG Struct Reference

Structure for GeoTag.

`#include <LgsxClient.h>`

Data Fields

- PNT3D **position**
  Location is specified in project coordinates.
- bool **hasCameraPosition**
  True if camera position is specified.
- PNT3D **cameraPosition**
  Camera position in project coordinates.
- wchar_t **name** [40]
  Name of geotag (does not need to be unique)
- uint64_t **assetId**
  When specified non-zero, this GeoTag contains Asset.
- uint64_t **geotagId**
  Unique ID for the GeoTag.
- uint64_t setupId
  When specified non-zero, this GeoTag belongs to the Setup.

7.7.1 Detailed Description

Structure for GeoTag.

Note: a GeoTag's position is specified in project coordinates.

Deprecated 26.06.2025 - New GeoTag API available via GeoTagHandle

7.7.2 Field Documentation

7.7.2.1 setupId

```
uint64_t CYGEOTAG::setupId
```

When specified non-zero, this GeoTag belongs to the Setup.

Note: setupId is expected to be 0 for LGSx files.

7.8 CYMODELINFO Struct Reference

Struct of returned Model info.

```
#include <LgsxClient.h>
```

Data Fields

- uint64_t **id**
  ID of this object for direct access.
- uint64_t **sourceId**
  Asset ID for source model file (e.g. IFC file, etc.)
- uint64_t sceneRepId
  Asset ID for scene rep file (OpenInventor).
- wchar_t **name** [80]
  Model name (contains file type)
- double **scale**
  Scale from meters (e.g., mm = 1000)
- SCyRgdBdyTxf **txf**
  Translation from Model center to model CS.
- PNT3D **sceneRepOffset**
  Translation from Model CS to scene rep origin.
- uint32_t **numRef**
  When non-zero, there are more files referenced by the model file.

7.8.1 Detailed Description

Struct of returned Model info.

7.8.2 Field Documentation

7.8.2.1 sceneRepId

```
uint64_t CYMODELINFO::sceneRepId
```

Asset ID for scene rep file (OpenInventor).

This Asset is a version of the source model file that's been optimized for rendering.

7.9 CYMODELNODEINFO Struct Reference

Struct of returned ModelNode info.

`#include <LgsxClient.h>`

Data Fields

- uint64_t **id**

  ID of this object for direct access.
- uint64_t **modelId**

  ID of model (which is referenced not owned)
- wchar_t **name** [80]

  Name of this model instance.
- SCyRgdBdyTxf **txf**

  Transformation from Point Cloud render offset to Model center.
- uint32_t **numChild**

  When non-zero, this is a group of models.

7.9.1 Detailed Description

Struct of returned ModelNode info.

7.10 CYRANGECUBE Struct Reference

Structure for an arbitrary range cube (a box)

`#include <LgsxClient.h>`

Data Fields

- PNT3D **corners** [4]

  Min corner point and near corners at x, y, z directions.

7.10.1 Detailed Description

Structure for an arbitrary range cube (a box)

7.11 CYRECT Struct Reference

Structure for rectangle (left, top, right, bottom)

`#include <LgsxClient.h>`

Data Fields

- int **left**
  Left.
- int **top**
  Top.
- int **right**
  Right.
- int **bottom**
  Bottom.

7.11.1 Detailed Description

Structure for rectangle (left, top, right, bottom)

7.12 CYRGBA Struct Reference

Structure for holding the RGBA color channels.

```
#include <LgsxClient.h>
```

Data Fields

- uint8_t **r**
  Red channel.
- uint8_t **g**
  Green channel.
- uint8_t **b**
  Blue channel.
- uint8_t **a**
  Alpha channel.

7.12.1 Detailed Description

Structure for holding the RGBA color channels.

7.13 CYSETUPCAMERAINFO Struct Reference

Structure for setup camera info.

```
#include <LgsxClient.h>
```


Data Fields

- uint64_t **setupCameraId**

  Setup Camera ID.
- uint64_t **setupId**

  Parent Setup ID.
- wchar_t **name** [40]

  Setup Camera name.
- M3DPOSETYPE **m_type**

  Camera model type.
- int **m_widthImage**

  Image width.
- int **m_heightImage**

  Image height.
- int **padding**

  Padding to 8-byte alignment.
- union {
  M3DPINHOLE **m_Pinhole**
  Parameters for PINHOLE_CAMERA.
  M3DDUALFISHEYE **m_DualFisheye**
  Parameters for DUAL_FISHEYE_CAMERA.
  M3DLUTCAMERA **m_Lut**
  Parameters for LUT_CAMERA.
  M3DFLAT **m_Flat**
  Parameters for FLAT.
  };

  Camera model parameters (varies by camera model type)

7.13.1 Detailed Description

Structure for setup camera info.

7.14 CYSETUPINFO Struct Reference

Structure for setup info.

```
#include <LgsxClient.h>
```

Data Fields

- SCyRgdBdyTxf **pose**

  Pose translation(xyz) + quaternion(uvws) in project coordinates.
- wchar_t **name** [80]

  Setup name.
- uint64_t time

  Mobile only: timestamp in UTC seconds.
- int vertexIndex

  Mobile only: index of the closest trajectory vertex.
- int **setupIndex**

  SETUPIDX that is used in point cloud property PM_SETUPIDX.

7.14.1 Detailed Description

Structure for setup info.

This has been extended to support both TLS setup and waypoints along run.

7.14.2 Field Documentation

7.14.2.1 time

```
uint64_t CYSETUPINFO::time
```

Mobile only: timestamp in UTC seconds.

UTC time is the number of seconds since 1/1/1970.

7.14.2.2 vertexIndex

```
int CYSETUPINFO::vertexIndex
```

Mobile only: index of the closest trajectory vertex.

If -1 is specified it will be determined by time.

7.15 CYSITEMAPIMAGECORNERS2D Struct Reference

Structure for sitemap image footprint in project coordinates.

```
#include <LgsxClient.h>
```

Data Fields

- PNT2D **topLeft**
  Top-left corner.
- PNT2D **topRight**
  Top-right corner.
- PNT2D **bottomRight**
  Bottom-right corner.
- PNT2D **bottomLeft**
  Bottom-left corner.

7.15.1 Detailed Description

Structure for sitemap image footprint in project coordinates.

7.16 CYTARGETINFO Struct Reference

Structure for target info.

`#include <LgsxClient.h>`

Data Fields

- int **type**
  Target type.
- PNT3D **origin**
  Location specified in Setup coordinate system.
- PNT3D **normal**
  Direction specified in Setup coordinate system. 0,0,0 means no normal.
- double **radius**
  0.0 means none, sphere target needs it
- wchar_t **label** [40]
  User-facing target name. Does not need to be unique.

7.16.1 Detailed Description

Structure for target info.

Note that origin and normal are specified in Setup coordinate system. A target always belongs to a Setup.

7.17 CYTRAJECTORYINFO Struct Reference

Structure for trajectory (a Run, Walk or Fly) header info of mobile scanning (such as BLK2GO)

`#include <LgsxClient.h>`

Data Fields

- wchar_t **jobName** [40]
  Job name (this could be empty if there is not multiple jobs)
- wchar_t **runName** [40]
  Trajectory (run) name: needs to be unique within job.
- uint64_t startTime
  Start timestamp in UTC seconds (accurate start time should be computed from first vertex)
- uint64_t stopTime
  End timestamp in UTC seconds (accurate end time should be computed from last vertex)
- double **timeScale**
  Scale from the ticks stored in each vertex to actual time lapse.
- int **startSetupIndex**
  Start SETUPIDX (used in point cloud property PM_SETUPIDXof) the run.
- double **scaleFactor**
  Scale from vertex index to the SETUPIDX.
- SCyRgdBdyTxf **pose**
  Pose translation(xyz) + quaternion(uvws) in project coordinates.

7.17.1 Detailed Description

Structure for trajectory (a Run, Walk or Fly) header info of mobile scanning (such as BLK2GO)

7.17.2 Field Documentation

7.17.2.1 startTime

```
uint64_t CYTRAJECTORYINFO::startTime
```

Start timestamp in UTC seconds (accurate start time should be computed from first vertex)

UTC time is the number of seconds since 1/1/1970.

7.17.2.2 stopTime

```
uint64_t CYTRAJECTORYINFO::stopTime
```

End timestamp in UTC seconds (accurate end time should be computed from last vertex)

UTC time is the number of seconds since 1/1/1970.

7.18 CYTRAJECTORYVERTEX Struct Reference

Structure for trajectory (a Run, Walk or Fly) node of mobile scanning (such as BLK2GO)

```
#include <LgsxClient.h>
```

Data Fields

- uint64_t time
  Number of ticks since the start of trajectory.
- PNT3D **trans**
  Position of vertex in project coordinates.
- QUA4D **orientation**
  Orientation (quaternion UVW + Scalar) of vertex in project coordinates.
- int setupIndex
  SETUPIDX identify points that belong (or are closest) to this vertex.
- double **quality**
  Quality (0.0 means none)

7.18.1 Detailed Description

Structure for trajectory (a Run, Walk or Fly) node of mobile scanning (such as BLK2GO)

7.18.2 Field Documentation

7.18.2.1 setupIndex

```
int CYTRAJECTORYVERTEX::setupIndex
```

SETUPIDX identify points that belong (or are closest) to this vertex.

Note that the storage model does not currently have this property for each vertex. Because of that, when you read trajectory vertexs you may notice that they do not match the values used to build it. This is not a problem. The value from the reader is a computed value from CWTRAJECTORYINFO parameters startSetupIndex and scaleFactor.

7.18.2.2 time

```
uint64_t CYTRAJECTORYVERTEX::time
```

Number of ticks since the start of trajectory.

This times a scale to get actual lapse from trajectory's startTime.

7.19 EnumPointsConfigHandle Struct Reference

Configuration handle for Lgsx_ReaderEnumPointsEx2().

```
#include <LgsxReader.h>
```

7.19.1 Detailed Description

Configuration handle for Lgsx_ReaderEnumPointsEx2().

Allocate with Lgsx_InitEnumPointsConfig(), configure with setters, pass to Lgsx_ReaderEnumPointsEx2(), then release with Lgsx_ReleaseEnumPointsConfig().

7.20 FieldEntryHandle Struct Reference

Descriptor for a Field Entry object.

```
#include <LgsxReader.h>
```

7.20.1 Detailed Description

Descriptor for a Field Entry object.

7.21 FieldHandle Struct Reference

Descriptor for a Field object.

`#include <LgsxReader.h>`

7.21.1 Detailed Description

Descriptor for a Field object.

7.22 GeoTagHandle Struct Reference

GeoTag data handle for accessing the GeoTag data.

`#include <LgsxReader.h>`

7.22.1 Detailed Description

GeoTag data handle for accessing the GeoTag data.

7.23 M3DDUALFISHEYE Struct Reference

Structure for M3D dual fish eye camera model.

`#include <LgsxClient.h>`

Data Fields

- double **m_nStartLimit**
  Start angle of the first position center.
- double **m_nEndLimit**
  End angle of the first position center.
- double **m_nRadius**
  Radius of the sphere in meters.
- double **m_ptCamPosition_1** [3]
  Position 3D (double x,y,z) of the first center of half sphere in project coordinates.
- double **m_ptCamPosition_2** [3]
  Position 3D (double x,y,z) of the second center of half sphere in project coordinates.
- double **m_orientation** [4]
  Quaternion [w, x, y, z] in project coordinates.

7.23.1 Detailed Description

Structure for M3D dual fish eye camera model.

7.24 M3DFLAT Struct Reference

Structure for M3D flat camera model.

`#include <LgsxClient.h>`

Data Fields

- double **m_Flat** [20]
  Flat camera model parameters.

7.24.1 Detailed Description

Structure for M3D flat camera model.

7.25 M3DLUTCAMERA Struct Reference

Structure for M3D LUT camera model.

`#include <LgsxClient.h>`

Data Fields

- double **m_scale**
  Angle of every pixel.
- double **m_radius**
  Radius used for generation of the panorama in meters.
- double **m_sphereOrigin** [3]
  Origin of the sphere in project coordinates.
- int **m_roi** [4]
  Region of interest of the panorama in pixel.
- double **m_orientation** [4]
  Quaternion [w, x, y, z] in project coordinates.

7.25.1 Detailed Description

Structure for M3D LUT camera model.

7.26 M3DPINHOLE Struct Reference

Structure for M3D pinhole camera model.

`#include <LgsxClient.h>`

Data Fields

- double **m_xPrincipalPoint**

  In meters: principal point x.
- double **m_yPrincipalPoint**

  In meters: principal point y.
- double **m_xFocal**

  In meters: focal length x.
- double **m_yFocal**

  In meters: focal length y.
- double **m_coeffDistortions** [M3DPINHOLEDISTORTION_COUNT]

  Distortion coefficients.
- double **m_position** [3]

  Position of the camera in project coordinates.
- double **m_orientation** [4]

  Quaternion [w, x, y, z] in project coordinates.

7.26.1 Detailed Description

Structure for M3D pinhole camera model.

7.27 PNT2D Struct Reference

Structure for 2D point (or vector, defined as 2 doubles)

```
#include <LgsxClient.h>
```

Data Fields

- double **x**

  X coordinate.
- double **y**

  Y coordinate.

7.27.1 Detailed Description

Structure for 2D point (or vector, defined as 2 doubles)

7.28 PNT3D Struct Reference

Structure for 3D point (or vector, defined as 3 doubles)

```
#include <LgsxClient.h>
```


Data Fields

- double **x**

  X coordinate.
- double **y**

  Y coordinate.
- double **z**

  Z coordinate.

7.28.1 Detailed Description

Structure for 3D point (or vector, defined as 3 doubles)

7.29 PointCloudFileHandle Struct Reference

Point Cloud handle for accessing the raw point cloud data.

`#include <LgsxReader.h>`

7.29.1 Detailed Description

Point Cloud handle for accessing the raw point cloud data.

7.30 ProcessingInfoHandle Struct Reference

ProcessingInfo data handle for accessing setup ProcessingInfo data.

`#include <LgsxReader.h>`

7.30.1 Detailed Description

ProcessingInfo data handle for accessing setup ProcessingInfo data.

7.31 QUA4D Struct Reference

Structure for Quaternion (or vector, defined as 4 doubles)

`#include <LgsxClient.h>`

Data Fields

- double **u**
  Quaternion U component.
- double **v**
  Quaternion V component.
- double **w**
  Quaternion W component.
- double **s**
  Quaternion S component.

7.31.1 Detailed Description

Structure for Quaternion (or vector, defined as 4 doubles)

7.32 ScanHandle Struct Reference

Scan data handle for accessing the Scan data.

`#include <LgsxReader.h>`

7.32.1 Detailed Description

Scan data handle for accessing the Scan data.

7.33 SCyRgdBdyTxf Struct Reference

Rigid transformation represented by a translation and a quaternion.

`#include <LgsxClient.h>`

Data Fields

- double **mDx**
  X translation.
- double **mDy**
  Y translation.
- double **mDz**
  Z translation.
- double **mU**
  Quaternion U.
- double **mV**
  Quaternion V.
- double **mW**
  Quaternion W.
- double **mS**
  Quaternion S.

7.33.1 Detailed Description

Rigid transformation represented by a translation and a quaternion.

Quaternion is represented as rotating around given axis a given angle: (mDx, mDy, mDz) is the translation component. (mU, mV, mW, mS) is the rotation as quaternion in which the first 3 relates to rotation axis. The values are normalized: So (mU,mV,mW) = axis $*$ sin(theta/2)), mS = cos(theta/2).

7.34 SensorInfoHandle Struct Reference

SensorInfo data handle for accessing the SensorInfo data.

`#include <LgsxReader.h>`

7.34.1 Detailed Description

SensorInfo data handle for accessing the SensorInfo data.

7.35 SitemapHandle Struct Reference

Sitemap data handle for accessing Sitemap data without reader-level cursor state.

`#include <LgsxReader.h>`

7.35.1 Detailed Description

Sitemap data handle for accessing Sitemap data without reader-level cursor state.

7.36 SitemapItemHandle Struct Reference

Sitemap item handle bound to the lifetime of its parent SitemapHandle.

`#include <LgsxReader.h>`

7.36.1 Detailed Description

Sitemap item handle bound to the lifetime of its parent SitemapHandle.

Chapter 8

File Documentation

8.1 /LeicaProjectSdk/LgsSdkClient/inc/LgsxSdk/LgsxClient.h File Reference

Data Structures

- struct CYRGBA

  Structure for holding the RGBA color channels.
- struct PNT3D

  Structure for 3D point (or vector, defined as 3 doubles)
- struct PNT2D

  Structure for 2D point (or vector, defined as 2 doubles)
- struct QUA4D

  Structure for Quaternion (or vector, defined as 4 doubles)
- struct SCyRgdBdyTxf

  Rigid transformation represented by a translation and a quaternion.
- struct CYBBOX

  Structure for bounding box (min, max) corners.
- struct CYRECT

  Structure for rectangle (left, top, right, bottom)
- struct CYRANGECUBE

  Structure for an arbitrary range cube (a box)
- struct CYSETUPINFO

  Structure for setup info.
- struct CYTARGETINFO

  Structure for target info.
- struct M3DFLAT

  Structure for M3D flat camera model.
- struct M3DPINHOLE

  Structure for M3D pinhole camera model.
- struct M3DDUALFISHEYE

  Structure for M3D dual fish eye camera model.
- struct M3DLUTCAMERA

Structure for M3D LUT camera model.

- struct CYSETUPCAMERAINFO

  Structure for setup camera info.
- struct CYGEOTAG

  Structure for GeoTag.
- struct CYSITEMAPIMAGECORNERS2D

  Structure for sitemap image footprint in project coordinates.
- struct CYASSET

  Structure for Asset.
- struct CYTRAJECTORYVERTEX

  Structure for trajectory (a Run, Walk or Fly) node of mobile scanning (such as BLK2GO)
- struct CYTRAJECTORYINFO

  Structure for trajectory (a Run, Walk or Fly) header info of mobile scanning (such as BLK2GO)
- struct CYMODELNODEINFO

  Struct of returned ModelNode info.
- struct CYMODELINFO

  Struct of returned Model info.

Macros

- #define **BitmapFormat__NoBitmap** 0

  Bitmap format - No bitmap.
- #define **BitmapFormat__RGB** 1

  Bitmap format - RGB.
- #define **BitmapFormat__RGBA** 2

  Bitmap format - RGBA.
- #define **BitmapFormat__RRGGBBScanlines** 3

  Bitmap format - RRGGBB with scanlines.
- #define **DrawPurpose__Summary** 0

  Draw purpose - Summary.
- #define **DrawPurpose__Visible** 1

  Draw purpose - Visible.
- #define **DrawPurpose__Dynamic** 2

  Draw purpose - Dynamic.
- #define **DrawPurpose__Plot** 3

  Draw purpose - Plot.
- #define **ScalingPolicy__ByWidth** 0

  Scaling policy - By width.
- #define **ScalingPolicy__ByHeight** 1

  Scaling policy - By height.
- #define **ScalingPolicy__ByMinimal** 2

  Scaling policy - By minimal dimension.
- #define **ScalingPolicy__ByBoth** 3

  Scaling policy - By both width and height.
- #define **ColorMappingMode__Unspecified** 0

  Color mapping mode - Unspecified.
- #define **ColorMappingMode__ColorFromScanner** 1

  Color mapping mode - Color from scanner.
- #define **ColorMappingMode__IntensityMap** 2

  Color mapping mode - Intensity.

- #define **ColorMappingMode__ElevationMap** 3
  Color mapping mode - Elevation.
- #define **ColorMappingMode__DistanceMap** 4
  Color mapping mode - Distance map.
- #define **ColorMappingMode__SingleColor** 5
  Color mapping mode - Single color.
- #define **ColorMappingMode__ClassMap** 6
  Color mapping mode - Class map.
- #define **ColorMappingMode__MaxValue** 6
  Color mapping mode - Maximum mode value.
- #define **ColorMappingScheme__Unspecified** 0
  Color mapping scheme - Unspecified.
- #define **ColorMappingScheme__MultiHue** 1
  Color mapping scheme - Multihue (multicolor)
- #define **ColorMappingScheme__GrayScale** 2
  Color mapping scheme - Gray scale range.
- #define **ColorMappingScheme__Red** 3
  Color mapping scheme - Red color range.
- #define **ColorMappingScheme__Green** 4
  Color mapping scheme - Green color range.
- #define **ColorMappingScheme__Blue** 5
  Color mapping scheme - Blue color range.
- #define **ColorMappingScheme__MaxValue** 5
  Color mapping scheme - Maximum scheme value.
- #define **PatchResultShape__ConvexHull** 1
  Patch result shape - Convex hull.
- #define **PatchResultShape__Rectangle** 2
  Patch result shape - Rectangle.
- #define **SteelSectionType__I** 0
  Steel section type - Wide flange (I-beam)
- #define **SteelSectionType__T** 1
  Steel section type - Tee.
- #define **SteelSectionType__C** 2
  Steel section type - Channel (U-shape)
- #define **SteelSectionType__O** 3
  Steel section type - Tubular (rectangular tube)
- #define **SteelSectionType__L** 4
  Steel section type - Angle (L-shape)
- #define **PM_XYZ_DATA** 1
  Property type mask - XYZ.
- #define **PM_INTENSITY_DATA** 2
  Property type mask - Intensity.
- #define **PM_COLOR_DATA** 4
  Property type mask - Color.
- #define **PM_NORMAL_DATA** 8
  Property type mask - Normal.
- #define **PM_CLASSIFICATION** 16
  Property type mask - Classification.
- #define **PM_SETUPIDX** 32
  Property type mask - Setup index.
- #define **PM_UserType1** 256

Property type mask - User type 1.

- #define **PM_UserType2** 512

  Property type mask - User type 2.

- #define **PM_UserType3** 1024

  Property type mask - User type 3.

- #define **PM_UserType4** 2048

  Property type mask - User type 4.

- #define **SYNC_STATE** 0

  State of global sync.

- #define **SYNC_DATAPOINT** 1

  State of Sync - 3D Point.

- #define **SYNC_PICKPOINT** 2

  State of Sync - Pick On Cloud.

- #define **SYNC_VIEWPOINT** 3

  State of Sync - View Point.

- #define **SYNC_QLIMITBOX** 4

  State of Sync - QLimitBox, QLimitBox range.

- #define **TargetType_BLACK_AND_WHITE** 0

  Target type - Black and white.

- #define **TargetType_SPHERE** 1

  Target type - Sphere.

- #define **TargetType_POINT_3D** 2

  Target type - 3D point.

- #define **TargetType_BLUE_CIRCULAR** 3

  Target type - Blue circular.

- #define **TargetType_BLUE_SQUARE** 4

  Target type - Blue square.

- #define **TargetType_IMAGE_ARUCO** 5

  Target type - ArUco marker.

- #define **TargetType_UNKNOWN** 6

  Target type - Unknown.

- #define **PROPERTY_NAME_LENGTH** 40

  Maximum length of property name (including null terminator)

- #define **MetadataType_BOOL** 0

  Metadata type - Bool.

- #define **MetadataType_INT** 1

  Metadata type - Integer.

- #define **MetadataType_INT64** 2

  Metadata type - 64-bit integer.

- #define **MetadataType_DOUBLE** 3

  Metadata type - Double.

- #define **MetadataType_STRING** 4

  Metadata type - String.

- #define **MetadataType_DOUBLE3D** 5

  Metadata type - 3D double (3 double precision numbers)

- #define **MetadataType_QUAT4D** 6

  Metadata type - Quaternion (4 double precision numbers)

- #define **MetadataType_METADATA** 7

  Metadata type - Metadata (another metadata object)

- #define **MetadataType_COMPLEX** 8

  Metadata type - Complex (arbitrary dimensions)

- #define **MetadataType_DOUBLEARRAY** 9

  Metadata type - Double array (arbitrary length array of doubles)

Typedefs

- typedef void * **LgsxMetadataPtr**

  Metadata container object.
- typedef char **LgsxPropertyName**[PROPERTY_NAME_LENGTH]

  Array to store property name.

Enumerations

- enum ImagePixelType {
  ImagePixelType_GRAY ,
  ImagePixelType_RGB ,
  ImagePixelType_RGBA ,
  ImagePixelType_BGR ,
  ImagePixelType_BGRA ,
  ImagePixelType_RGBE }

  Constants of type ImagePixelType used for images.
- enum SitemapItemType {
  SitemapItemType_TlsSetup ,
  SitemapItemType_Run }

  Sitemap item type.
- enum SitemapImageType {
  SitemapImageType_Unknown = 0 ,
  SitemapImageType_Background ,
  SitemapImageType_Overview ,
  SitemapImageType_Acceptance }

  Sitemap image type.
- enum CubemapFaceIndex {
  CubemapFaceIndex_PosY = 0 ,
  CubemapFaceIndex_NegY = 1 ,
  CubemapFaceIndex_NegX = 2 ,
  CubemapFaceIndex_PosX = 3 ,
  CubemapFaceIndex_PosZ = 4 ,
  CubemapFaceIndex_NegZ = 5 }

  Cubemap face index.
- enum GeotagAnchorType {
  GeotagAnchorType_None ,
  GeotagAnchorType_TlsSetup ,
  GeotagAnchorType_Run }

  Geotag anchor type.
- enum M3DPOSETYPE {
  PINHOLE_CAMERA ,
  LUT_CAMERA ,
  DUAL_FISHEYE_CAMERA ,
  FLAT }

  M3D camera model type (see CYSETUPCAMERAINFO.m_type)
- enum M3DPINHOLEDISTORTION {
  M3DPINHOLEDISTORTION_K1 = 0 ,
  M3DPINHOLEDISTORTION_K2 = 1 ,
  M3DPINHOLEDISTORTION_P1 = 2 ,
  M3DPINHOLEDISTORTION_P2 = 3 ,
  M3DPINHOLEDISTORTION_K3 = 4 ,
  M3DPINHOLEDISTORTION_K4 = 5 ,
  M3DPINHOLEDISTORTION_K5 = 6 ,
  M3DPINHOLEDISTORTION_K6 = 7 ,
  M3DPINHOLEDISTORTION_COUNT }

  Distortion coefficient indices for M3DPINHOLE.

Functions

- const wchar_t * LgsxUtil_GetCurrentTimeString ()

  Convenience function: Return current local time formatted as "HHMMSS".
- const wchar_t * LgsxUtil_GetCurrentDateString ()

  Convenience function: Return local date formatted as "YYMMDD".
- void * LgsxUtil_TimeStart ()

  Create a new timer and starts it.
- double LgsxUtil_TimeGetLapse (void *pTime)

  Returns elapsed time for the given timer in seconds.
- const wchar_t * LgsxUtil_TimeGetLapseStr (void *pTime, int format)

  Returns elapsed time for the given timer in seconds as a string.
- int LgsxUtil_TimeEnd (void **pTime)

  Frees the timer object.
- int LgsxUtil_Sleep (int millisec)

  Suspends execution for the given number of milliseconds.
- void * LgsxUtil_AllocMem (int size)

  Allocate a block of memory with the given size and return a pointer to it.
- void LgsxUtil_FreeMem (void **ptr)

  Free a block of memory previously created by the SDK.
- int LgsxUtil_StrDupA (const char *value, char **ptrDupValue)

  Duplicate string.
- int LgsxUtil_StrDupW (const wchar_t *value, wchar_t **ptrDupValue)

  Duplicate string.
- int LgsxUtil_A2W (const char *value, wchar_t *tValue)

  Convert UTF8 encoded text to UTF16.
- int LgsxUtil_A2WEx (const char *value, wchar_t *tValue, int maxLen)

  Convert UTF8 encoded text to UTF16 into a bounded output buffer.
- int LgsxUtil_AllocA2W (const char *value, wchar_t **pTValuePtr)

  Convert UTF8 encoded text to UTF16 and allocate output buffer for the whole converted string.
- int LgsxUtil_W2A (const wchar_t *value, char *sValue)

  Convert UTF16 encoded text to UTF8.
- int LgsxUtil_W2AEx (const wchar_t *value, char *sValue, int maxLen)

  Converts a wide string to a UTF-8 encoded string with buffer size limit.
- int LgsxUtil_AllocW2A (const wchar_t *value, char **pSValuePtr)

  Convert UTF16 encoded text to UTF8 and allocate output buffer for the whole converted string.
- void Lgsx_FreeHandle (void *handle)

  Frees the SDK object referenced by the given handle.
- const wchar_t * Lgsx_GetLastError ()

  Returns error string for most recent SDK error.
- int Lgsx_GetLastErrorNo ()

  Returns error number for most recent SDK error.
- int Lgsx_Initialize (const wchar_t *pTempPath, bool enableLog)

  Initialize the LGSx SDK APIs.
- int Lgsx_Uninitialize ()

  Unintialize LGSx SDK APIs.
- int Lgsx_InitLicense (const wchar_t *lic, const wchar_t *ver)

  Initialize the license to be loaded.
- int Lgsx_InitLicenseEx (const wchar_t *lic, const wchar_t *ver, const wchar_t *product)

  Initialize the license to be loaded.
- int Lgsx_InitLicenseEx2 (const wchar_t *lic, const wchar_t *ult, const wchar_t *ver, const wchar_t *product)

Initialize the license to be loaded.

- int Lgsx_IsLicensed ()

  Check if the license is valid.
- int Lgsx_LicenseDaysLeft ()

  Check the number of days left for the license.
- int Lgsx_LoadLicense ()

  Load license and hold the usage.
- int Lgsx_UnloadLicense ()

  Unload license.
- int Lgsx_IsLicensedExA (const char *lic, const char *ver)

  Check if a named license is valid.
- int Lgsx_LicenseDaysLeftExA (const char *lic, const char *ver)

  Check the number of days left for given license.
- int Lgsx_LoadLicenseExA (const char *lic, const char *ver)

  Load given license and hold usage until UnloadLicense.
- int Lgsx_UnloadLicenseExA (const char *lic)

  Unload a named license.
- int Lgsx_GetProductVersion (wchar_t *version, long *buildNum)

  Get LGSx SDK application's version and build number.
- int Lgsx_GetProductVersion2 (wchar_t **pVersionPtr, long *buildNum)

  Get LGSx SDK application's version and build number, without buffer truncation.
- int Lgsx_InitProductVersion (const wchar_t *customVersion)

  Initialize LGSx SDK application's version string.
- int Lgsx_InitProductVersionEx (const wchar_t *customVersion, const wchar_t *fileVersion)

  Initialize LGSx SDK application's version string and file version.
- int Lgsx_InitProductName (const wchar_t *prodName)

  Define the name of the product you want to display in About dialog.
- int Lgsx_InitProductBuildNumber (int build)

  Initialize LGSx SDK application's build number.
- LgsxMetadataPtr Lgsx_MetaCreateInstance ()

  Creates an instance of metadata container.
- LgsxMetadataPtr Lgsx_MetaClone (LgsxMetadataPtr pMetadata)

  Clones a metadata container.
- int Lgsx_MetaRemove (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName)

  Remove a key pair from metadata.
- int Lgsx_MetaHas (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName)

  TRUE if the key exists in the metadata.
- int Lgsx_MetaReset (LgsxMetadataPtr pMetadata)

  Resets traverse pointer in the metadata.
- int Lgsx_MetaMoveNext (LgsxMetadataPtr pMetadata, LgsxPropertyName pName, int *pType)

  Move to next key pair of the metadata.
- int Lgsx_MetaIsEmpty (LgsxMetadataPtr pMetadata, bool *pIsEmpty)

  Check if a metadata container is empty (has no key-value pairs).
- int Lgsx_MetaMoveNext2 (LgsxMetadataPtr pMetadata, const char **ppName, int *pType)

  Move to next key pair of the metadata.
- int Lgsx_MetaGetBool (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, bool *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetInt (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, int *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetInt64 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, __int64 *pValue)

  Get the value associated with the given name in metadata.

- int Lgsx_MetaGetDouble (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetString (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, wchar_t *pValue, int maxLength)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetString2 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, wchar_t **pp↩ Value)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetDouble3 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetDouble4 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *pValue)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetMetadata (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, LgsxMetadataPtr *pMetadataOut)

  Get the value associated with the given name in metadata.
- int Lgsx_MetaGetDoubleArray (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double *p↩ Values, int *pCount)

  Get the double array value associated with the given name in metadata.
- int Lgsx_MetaGetDoubleArraySize (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, int *p↩ Count)

  Get the size of a double array associated with the given name in metadata.
- int Lgsx_MetaAddBool (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, bool value)

  Add name value pair to metadata.
- int Lgsx_MetaAddInt (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, int value)

  Add name value pair to metadata.
- int Lgsx_MetaAddInt64 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, __int64 value)

  Add name value pair to metadata.
- int Lgsx_MetaAddDouble (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, double value)

  Add name value pair to metadata.
- int Lgsx_MetaAddString (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, const wchar_t *p↩ Value)

  Add name value pair to metadata.
- int Lgsx_MetaAddDouble3 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, const double *pValue)

  Add name value pair to metadata.
- int Lgsx_MetaAddDouble4 (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, const double *pValue)

  Add name value pair to metadata.
- int Lgsx_MetaAddMetadata (LgsxMetadataPtr pMetadata, const LgsxPropertyName pName, LgsxMetadataPtr pMetadataIn)

  Add name value pair to metadata.

8.1.1 Detailed Description

Copyright 2026 by Leica Geosystems AG, Heerbrugg.

8.1.2 Enumeration Type Documentation

8.1.2.1 CubemapFaceIndex

`enum CubemapFaceIndex`

Cubemap face index.

Enumerator

CubemapFaceIndex_PosY	Cubemap face index - Positive Y.
CubemapFaceIndex_NegY	Cubemap face index - Negative Y.
CubemapFaceIndex_NegX	Cubemap face index - Negative X.
CubemapFaceIndex_PosX	Cubemap face index - Positive X.
CubemapFaceIndex_PosZ	Cubemap face index - Positive Z.
CubemapFaceIndex_NegZ	Cubemap face index - Negative Z.

8.1.2.2 GeotagAnchorType

`enum GeotagAnchorType`

Geotag anchor type.

Enumerator

GeotagAnchorType_None	Geotag anchor type - None (attached to project)
GeotagAnchorType_TlsSetup	Geotag anchor type - TLS Setup.
GeotagAnchorType_Run	Geotag anchor type - Run.

8.1.2.3 ImagePixelType

`enum ImagePixelType`

Constants of type ImagePixelType used for images.

Enumerator

ImagePixelType_GRAY	1 byte
ImagePixelType_RGB	3 bytes (R is low byte)
ImagePixelType_RGBA	4 bytes (R is low byte)
ImagePixelType_BGR	3 bytes (B is low byte)
ImagePixelType_BGRA	4 bytes (B is low byte)
ImagePixelType_RGBE	4 bytes (R is low byte)

8.1.2.4 M3DPINHOLEDISTORTION

`enum M3DPINHOLEDISTORTION`

Distortion coefficient indices for M3DPINHOLE.

Enumerator

M3DPINHOLEDISTORTION_K1	Distortion coefficient K1.
M3DPINHOLEDISTORTION_K2	Distortion coefficient K2.
M3DPINHOLEDISTORTION_P1	Distortion coefficient P1.

M3DPINHOLEDISTORTION_P2	Distortion coefficient P2.
M3DPINHOLEDISTORTION_K3	Distortion coefficient K3.
M3DPINHOLEDISTORTION_K4	Distortion coefficient K4.
M3DPINHOLEDISTORTION_K5	Distortion coefficient K5.
M3DPINHOLEDISTORTION_K6	Distortion coefficient K6.
M3DPINHOLEDISTORTION_COUNT	Number of distortion coefficients.

8.1.2.5 M3DPOSETYPE

enum M3DPOSETYPE

M3D camera model type (see CYSETUPCAMERAINFO.m_type)

Enumerator

PINHOLE_CAMERA	Pinhole camera.
LUT_CAMERA	LUT camera.
DUAL_FISHEYE_CAMERA	Dual fishaye camera.
FLAT	Flat.

8.1.2.6 SitemapImageType

enum SitemapImageType

Sitemap image type.

Enumerator

SitemapImageType_Unknown	Sitemap image type - Unknown.
SitemapImageType_Background	Sitemap image type - Background.
SitemapImageType_Overview	Sitemap image type - Overview.
SitemapImageType_Acceptance	Sitemap image type - Acceptance.

8.1.2.7 SitemapItemType

enum SitemapItemType

Sitemap item type.

Enumerator

SitemapItemType_TlsSetup	Sitemap item type - TLS Setup.
SitemapItemType_Run	Sitemap item type - Run.

8.2 /LeicaProjectSdk/LgsSdkClient/inc/LgsxSdk/LgsxReader.h File Reference

Data Structures

- struct AssetHandle

  Asset data handle for accessing Asset data without buffer truncation.
- struct CategoryHandle

  Descriptor for a Category object.
- struct CategoryValueHandle

  Descriptor for a Category Value object.
- struct CategoryEntryHandle

  Descriptor for a Category Entry object.
- struct FieldHandle

  Descriptor for a Field object.
- struct FieldEntryHandle

  Descriptor for a Field Entry object.
- struct SensorInfoHandle

  SensorInfo data handle for accessing the SensorInfo data.
- struct ProcessingInfoHandle

  ProcessingInfo data handle for accessing setup ProcessingInfo data.
- struct ScanHandle

  Scan data handle for accessing the Scan data.
- struct GeoTagHandle

  GeoTag data handle for accessing the GeoTag data.
- struct SitemapHandle

  Sitemap data handle for accessing Sitemap data without reader-level cursor state.
- struct SitemapItemHandle

  Sitemap item handle bound to the lifetime of its parent SitemapHandle.
- struct EnumPointsConfigHandle

  Configuration handle for Lgsx_ReaderEnumPointsEx2().
- struct PointCloudFileHandle

  Point Cloud handle for accessing the raw point cloud data.

Macros

Sitemap corner correction flags

Bitmask flags for the `pFlags` parameter of Lgsx_SitemapImageGetCorners().

- #define **LGSX_SITEMAP_CORNERS_CALCULATE_MISSING** 0x01

  Reconstruct corners from setup/track pixel positions when corner data is absent.
- #define LGSX_SITEMAP_CORNERS_FIX_INCONSISTENT 0x02

  Detect and replace corners that are inconsistent with the recorded setup pixel positions.
- #define LGSX_SITEMAP_CORNERS_REJECT_DEGENERATE 0x04

  Treat geometrically degenerate stored corners as missing.

Typedefs

- typedef void * **LgsxReaderPtr**

  Handle for LGSx reader object.

Enumerations

- enum LgsxMigratedMetaKeyField {
  LgsxMigratedMetaKeyField_Category = 0 ,
  LgsxMigratedMetaKeyField_Label ,
  LgsxMigratedMetaKeyField_TagIndex }

  Enum for the migrated metadata.

Functions

- LgsxReaderPtr Lgsx_ReaderCreate (int *rError)

  Create a Reader and return its handle.
- int Lgsx_ReaderOpen (LgsxReaderPtr reader, const wchar_t *pFilePath, const char *password)

  Open the given LGSx file with the given Reader using the given password.
- int Lgsx_ReaderQueryPropertyTypes (LgsxReaderPtr reader)

  Returns the Property Types available for points in the project's Point Cloud.
- int Lgsx_ReaderClose (LgsxReaderPtr reader)

  Closes the LGSx file for this Reader.
- int Lgsx_ReaderGetLastError (LgsxReaderPtr reader, char *pError, int maxLen)

  Returns a string description of the last error that occurred with the given Reader.
- int Lgsx_ReaderGetLastError2 (LgsxReaderPtr reader, char **pErrorPtr)

  Returns a string description of the last error that occurred with the given Reader.
- int Lgsx_ReaderHasPassword (const wchar_t *pFilePath, bool *hasPassword)

  Checks if file is password protected.
- int Lgsx_ReaderGetMetadata (LgsxReaderPtr reader, uint64_t *pPointCount, CYBBOX *pBbox, LgsxMetadataPtr *pMetadata)

  Returns the project metadata for the opened project.
- int Lgsx_ReaderGetProjectInfo (LgsxReaderPtr reader, LgsxMetadataPtr *pOwnerInfo, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char **pThumbImage)

  Returns the project metadata and thumbnail for the opened project.
- int Lgsx_ReaderGetProjectDescription (LgsxReaderPtr reader, wchar_t *pDesc, int maxLen)

  Returns the project description string for the opened project.
- int Lgsx_ReaderGetProjectDescription2 (LgsxReaderPtr reader, wchar_t **pDescPtr)

  Returns the project description string for the opened project.
- int Lgsx_ReaderEnumAssets (LgsxReaderPtr reader, int *pNumAsset, bool bGetAll)

  Collects assets in the project and returns the count.
- int Lgsx_ReaderEndAssets (LgsxReaderPtr reader)

  Ends the current Asset enumeration.
- int Lgsx_ReaderMoveNextAsset (LgsxReaderPtr reader, uint64_t *assetId)

  Advance to the next Asset in the active collection and return its ID.
- int Lgsx_ReaderGetAsset (LgsxReaderPtr reader, uint64_t assetId, CYASSET *pAsset, LgsxMetadataPtr *pMetadata)

  Get Info and metadata for current asset as per Lgsx_ReaderMoveNextAsset() context.
- int Lgsx_ReaderAssetGetGuid (LgsxReaderPtr reader, const CYASSET *pAsset, char **pGuidPtr)

  Returns the Asset GUID for the asset passed as parameter.

- int Lgsx_GetAssetAsImage (LgsxReaderPtr reader, uint64_t assetId, wchar_t *pName, int maxLenName, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char **pImage)

  Get the "payload" image for current asset as per Lgsx_ReaderMoveNextAsset() context.
- int Lgsx_GetAssetAsImage2 (LgsxReaderPtr reader, uint64_t assetId, wchar_t **pNamePtr, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char **pImage)

  Get the "payload" image for current asset as per Lgsx_ReaderMoveNextAsset() context.
- int Lgsx_AssetReadImage (const CYASSET *pAsset, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **pImage)

  Get the "payload" as image for the given Asset.
- int Lgsx_AssetReadBinary (const CYASSET *pAsset, unsigned char **pData, int *pSize)

  Get the "payload" as binary data for the given Asset.
- int Lgsx_GetAssetThumbImage (LgsxReaderPtr reader, uint64_t assetId, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char **pImage)

  Get the thumbnail image for current asset as per Lgsx_ReaderMoveNextAsset() context.
- int Lgsx_AssetHasType (LgsxReaderPtr reader, uint64_t assetId, bool *pHasAssetType)

  Check if an Asset has a type set.
- int Lgsx_ReaderGetAssetHandle (LgsxReaderPtr reader, uint64_t assetId, AssetHandle *pHandle)

  Acquire an Asset handle for the Asset with the given ID.
- int Lgsx_AssetRelease (AssetHandle *pHandle)

  Release the Asset handle returned by Lgsx_ReaderGetGeoTagAssetHandle() or Lgsx_ReaderGetAssetHandle().
- int Lgsx_AssetGetName (const AssetHandle handle, const wchar_t **pNamePtr)

  Get the name of the Asset.
- int Lgsx_AssetGetType (const AssetHandle handle, const wchar_t **pTypePtr)

  Get the type of the Asset.
- int Lgsx_AssetGetMimeType (const AssetHandle handle, const wchar_t **pMimeTypePtr)

  Get the MIME type of the Asset.
- int Lgsx_AssetGetFilename (const AssetHandle handle, const wchar_t **pFilenamePtr)

  Get the extracted file path of the Asset payload.
- int Lgsx_AssetGetUrl (const AssetHandle handle, const wchar_t **pUrlPtr)

  Get the external URL of the Asset (for URL-type Assets).
- int Lgsx_AssetGetGuid (const AssetHandle handle, const char **pGuidPtr)

  Get the GUID of the Asset.
- int Lgsx_AssetGetId (const AssetHandle handle, uint64_t *pId)

  Get the ID of the Asset.
- int Lgsx_AssetGetCreationTime (const AssetHandle handle, uint64_t *pCreationTime)

  Get the creation timestamp of the Asset.
- int Lgsx_AssetGetModificationTime (const AssetHandle handle, uint64_t *pModificationTime)

  Get the modification timestamp of the Asset.
- int Lgsx_AssetIsExternalUrl (const AssetHandle handle, bool *pIsExternalUrl)

  Check whether the Asset is an external URL.
- int Lgsx_AssetGetMetadata (const AssetHandle handle, LgsxMetadataPtr *pMetadata)

  Get the metadata of the Asset.
- int Lgsx_AssetHandleReadImage (const AssetHandle handle, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **pImage)

  Read the Asset payload as an image.
- int Lgsx_AssetHandleReadBinary (const AssetHandle handle, unsigned char **pData, int *pSize)

  Read the Asset payload as binary data.
- int Lgsx_AssetHandleReadImageBinary (LgsxReaderPtr reader, const AssetHandle handle, wchar_t **pImageType, int *pWidth, int *pHeight, unsigned char **pData, int *pSize)

  Read the raw encoded image bytes from the Asset, stripping SDK metadata.
- int Lgsx_ReaderEnumCategories (LgsxReaderPtr reader, int *pNumCategories)

Collects Categories in the project and returns the count.

- int Lgsx_ReaderGetCategory (LgsxReaderPtr reader, int pos, CategoryHandle *pHandle)

  Read Category data at the given position in the active collection.
- int Lgsx_CategoryRelease (CategoryHandle *pHandle)

  Release the Category data pointer returned by Lgsx_ReaderGetCategory().
- int Lgsx_CategoryGetGuid (const CategoryHandle handle, const char **pId)

  Get the GUID of the Category.
- int Lgsx_CategoryGetName (const CategoryHandle handle, const wchar_t **pName)

  Get the name of the Category.
- int Lgsx_CategoryGetPriority (const CategoryHandle handle, int *pPriority)

  Get the priority of the Category.
- int Lgsx_CategoryGetCreationTimestamp (const CategoryHandle handle, uint64_t *pTimestamp)

  Get the creation timestamp of the Category.
- int Lgsx_CategoryGetModificationTimestamp (const CategoryHandle handle, uint64_t *pTimestamp)

  Get the modification timestamp of the Category.
- int Lgsx_CategoryEnumValues (const CategoryHandle handle, int *pNumValues)

  Enumerate the values within a Category and return the count.
- int Lgsx_CategoryGetValue (const CategoryHandle handle, int pos, CategoryValueHandle *pValueHandle)

  Get a Category Value at the specified position.
- int Lgsx_CategoryValueGetGuid (const CategoryValueHandle handle, const char **pId)

  Get the GUID of a Category Value.
- int Lgsx_CategoryValueGetValue (const CategoryValueHandle handle, const wchar_t **pValue)

  Get the value of a Category Value.
- int Lgsx_CategoryValueGetPriority (const CategoryValueHandle handle, int *pPriority)

  Get the priority of a Category Value.
- int Lgsx_CategoryValueGetCreationTimestamp (const CategoryValueHandle handle, uint64_t *pTimestamp)

  Get the creation timestamp of a Category Value.
- int Lgsx_CategoryValueGetModificationTimestamp (const CategoryValueHandle handle, uint64_t *p↩ Timestamp)

  Get the modification timestamp of a Category Value.
- int Lgsx_CategoryValueHasColor (const CategoryValueHandle handle, bool *pStatus)

  Check if a Category Value has a color associated with it.
- int Lgsx_CategoryValueGetColor (const CategoryValueHandle handle, CYRGBA *pColor)

  Get the color of a Category Value.
- int Lgsx_CategoryEntryGetCategory (const CategoryEntryHandle handle, CategoryHandle *pCategory↩ Handle)

  Get the Category from a Category Entry.
- int Lgsx_CategoryEntryHasValue (const CategoryEntryHandle handle, bool *pStatus)

  Check if a Category Entry has a value associated with it.
- int Lgsx_CategoryEntryGetValue (const CategoryEntryHandle handle, CategoryValueHandle *pValue↩ Handle)

  Get the Category Value from a Category Entry.
- int Lgsx_GetMigratedFieldGuid (LgsxMigratedMetaKeyField field, const char **pId)

  Get the GUID of the migrated Field specified by "field" argument.
- int Lgsx_ReaderEnumFields (LgsxReaderPtr reader, int *pNumFields)

  Collects Fields in the project and returns the count.
- int Lgsx_ReaderGetField (LgsxReaderPtr reader, int pos, FieldHandle *pHandle)

  Read Field data at the given position in the active collection.
- int Lgsx_FieldRelease (FieldHandle *pHandle)

  Release the Field data pointer returned by Lgsx_ReaderGetField().
- int Lgsx_FieldGetGuid (const FieldHandle handle, const char **pId)

Get the GUID of the Field.

- int Lgsx_FieldGetName (const FieldHandle handle, const wchar_t **pName)

  Get the name of the Field.
- int Lgsx_FieldGetPriority (const FieldHandle handle, int *pPriority)

  Get the priority of the Field.
- int Lgsx_FieldGetCreationTimestamp (const FieldHandle handle, uint64_t *pTimestamp)

  Get the creation timestamp of the Field.
- int Lgsx_FieldGetModificationTimestamp (const FieldHandle handle, uint64_t *pTimestamp)

  Get the modification timestamp of the Field.
- int Lgsx_FieldEntryGetField (const FieldEntryHandle handle, FieldHandle *pFieldHandle)

  Get the Field from a Field Entry.
- int Lgsx_FieldEntryGetValue (const FieldEntryHandle handle, const wchar_t **pValue)

  Get the value from a Field Entry.
- int Lgsx_ReaderEnumSetups (LgsxReaderPtr reader, int *pNumSetup)

  Collects Setups in the project and returns the count.
- int Lgsx_ReaderEnumSetupsEx (LgsxReaderPtr reader, PNT3D *pLocation, double range, int *pNumSetup)

  Collects Setups in the project that are within the given range from the given position.
- int Lgsx_ReaderMoveNextSetup (LgsxReaderPtr reader, uint64_t *setupId, CYSETUPINFO *pSetup, LgsxMetadataPtr *pMetadata)

  Advances to the next setup in the active collection and returns ID, info, and metadata.
- int Lgsx_ReaderGetSetupGuid (LgsxReaderPtr reader, uint64_t setupId, char **pGuidPtr)

  Get GUID of the setup for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_ReaderHasSetupSensorInfo (LgsxReaderPtr reader, bool *pHasInfo)

  Check if setup has SensorInfo for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_ReaderGetSetupSensorInfo (LgsxReaderPtr reader, SensorInfoHandle *pHandle)

  Get SensorInfo of the setup for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_SensorInfoGetGuid (const SensorInfoHandle handle, const char **pGuidPtr)

  Get the GUID of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetDeviceInfoCaptureTime (const SensorInfoHandle handle, uint64_t *pTimestamp)

  Get the capture time of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetManufacturer (const SensorInfoHandle handle, const wchar_t **pManufacturer)

  Get the manufacturer of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetScannerType (const SensorInfoHandle handle, const wchar_t **pScannerType)

  Get the scanner type of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetSerialNumber (const SensorInfoHandle handle, const wchar_t **pSerialNumber)

  Get the serial number of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetArticleNumber (const SensorInfoHandle handle, const wchar_t **pArticleNumber)

  Get the article number of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetHardwareVersion (const SensorInfoHandle handle, const wchar_t **pHardware↩ Version)

  Get the hardware version of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetFirmwareVersion (const SensorInfoHandle handle, const wchar_t **pFirmware↩ Version)

  Get the firmware version of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoGetDeviceInfoSnapshot (const SensorInfoHandle handle, LgsxMetadataPtr *pSnapshot)

  Get the device info snapshot of the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_SensorInfoRelease (SensorInfoHandle *pHandle)

  Release the SensorInfo data pointer returned by Lgsx_ReaderGetSetupSensorInfo().
- int Lgsx_ReaderEnumSetupProcessingInfos (LgsxReaderPtr reader, int *pNumProcessingInfo)

  Enumerate ProcessingInfo entries associated with current setup as per Lgsx_ReaderMoveNextSetup() context and return the count.

- int Lgsx_ReaderGetSetupProcessingInfo (LgsxReaderPtr reader, int pos, ProcessingInfoHandle *pHandle)

  Get ProcessingInfo entry at the specified position for current setup as per Lgsx_ReaderMoveNextSetup() context.
- int Lgsx_ProcessingInfoGetGuid (const ProcessingInfoHandle handle, const char **pGuidPtr)

  Get GUID of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetProcessingTimestamp (const ProcessingInfoHandle handle, uint64_t *pTimestamp)

  Get processing timestamp of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetAppName (const ProcessingInfoHandle handle, const wchar_t **pAppName)

  Get processing app name of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetAppVersion (const ProcessingInfoHandle handle, const wchar_t **pAppVersion)

  Get processing app version of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoHasLibraryVersion (const ProcessingInfoHandle handle, bool *pHasLibraryVersion)

  Check whether ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo() has library version.
- int Lgsx_ProcessingInfoGetLibraryVersion (const ProcessingInfoHandle handle, const wchar_t **pLibraryVersion)

  Get processing library version of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetOrderIndex (const ProcessingInfoHandle handle, int *pOrderIndex)

  Get processing order index of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoGetPipelineInfo (const ProcessingInfoHandle handle, LgsxMetadataPtr *pPipelineInfo)

  Get pipeline metadata of ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ProcessingInfoRelease (ProcessingInfoHandle *pHandle)

  Release ProcessingInfo data pointer returned by Lgsx_ReaderGetSetupProcessingInfo().
- int Lgsx_ReaderEnumSetupScans (LgsxReaderPtr reader, int *pNumScans)

  Collects Scans for the current Setup and returns the count.
- int Lgsx_ReaderGetSetupScan (LgsxReaderPtr reader, int pos, ScanHandle *pHandle)

  Read Scan data at the given position in the active collection.
- int Lgsx_ScanRelease (ScanHandle *pHandle)

  Release the Scan data pointer returned by Lgsx_ReaderGetSetupScan().
- int Lgsx_ScanGetGuid (const ScanHandle handle, const char **pGuid)

  Get GUID of the Scan.
- int Lgsx_ScanGetScanType (const ScanHandle handle, int *pScanType)

  Get scan type of the Scan.
- int Lgsx_ScanGetScanRole (const ScanHandle handle, int *pScanRole)

  Get scan role of the Scan.
- int Lgsx_ScanGetPointCount (const ScanHandle handle, uint64_t *pPointCount)

  Get point count of the Scan.
- int Lgsx_ScanGetCreationTimestamp (const ScanHandle handle, uint64_t *pTimestamp)

  Get creation timestamp of the Scan.
- int Lgsx_ScanGetModificationTimestamp (const ScanHandle handle, uint64_t *pTimestamp)

  Get modification timestamp of the Scan.
- int Lgsx_ScanHasWindow (const ScanHandle handle, bool *pHasWindow)

  Check whether the Scan has a scan window.
- int Lgsx_ScanGetWindowMinAzimuth (const ScanHandle handle, double *pMinAzimuth)

  Get minimum azimuth angle of the scan window (radians).
- int Lgsx_ScanGetWindowMinElevation (const ScanHandle handle, double *pMinElevation)

  Get minimum elevation angle of the scan window (radians).
- int Lgsx_ScanGetWindowMaxAzimuth (const ScanHandle handle, double *pMaxAzimuth)

  Get maximum azimuth angle of the scan window (radians).
- int Lgsx_ScanGetWindowMaxElevation (const ScanHandle handle, double *pMaxElevation)

  Get maximum elevation angle of the scan window (radians).

- int Lgsx_ScanGetCaptureTimestamp (const ScanHandle handle, uint64_t *pTimestamp)

  Get capture timestamp of the Scan.
- int Lgsx_ScanGetScanDensity (const ScanHandle handle, double *pDensity)

  Get scan density of the Scan.
- int Lgsx_ScanGetCaptureInfoSnapshot (const ScanHandle handle, LgsxMetadataPtr *pSnapshot)

  Get capture info snapshot metadata of the Scan.
- int Lgsx_ReaderGetCurrentSetupName (LgsxReaderPtr reader, wchar_t **pNamePtr)

  Get the full (untruncated) name of the most recently returned Setup.
- int Lgsx_ReaderEndSetups (LgsxReaderPtr reader)

  Frees the active collection.
- int Lgsx_ReaderGetImageLayerNames (LgsxReaderPtr reader, wchar_t *pLayers, int maxLen)

  Retrieve the Image Layer names in the active Setup.
- int Lgsx_ReaderGetImageLayerNames2 (LgsxReaderPtr reader, wchar_t **pLayersPtr)

  Retrieve the Image Layer names in the active Setup.
- int Lgsx_ReaderGetPanoImage (LgsxReaderPtr reader, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char **panoImage)

  Get the pano image for the active setup.
- int Lgsx_ReaderGetCubemapImageType (LgsxReaderPtr reader, const wchar_t *pLayer, wchar_t **p↩ ImageType)

  Retrieves the image type string for a cubemap image layer in the active setup.
- int Lgsx_ReaderGetCubemapMaxLevel (LgsxReaderPtr reader, const wchar_t *pLayer, int *pLevel)

  Retrieves the maximum mipmap level available for a cubemap image layer in the active setup.
- int Lgsx_ReaderGetCubemapFaceSize (LgsxReaderPtr reader, const wchar_t *pLayer, int level, int *pWidth, int *pHeight)

  Retrieves the size of a cubemap face for a given layer and mipmap level in the active setup.
- int Lgsx_ReaderGetCubemapMetadata (LgsxReaderPtr reader, const wchar_t *pLayer, LgsxMetadataPtr *pMetadata)

  Retrieves the metadata for a cubemap in the active setup.
- int Lgsx_ReaderGetCubemapFaceMetadata (LgsxReaderPtr reader, const wchar_t *pLayer, int face, LgsxMetadataPtr *pMetadata)

  Retrieves the metadata for a specific face of a cubemap in the active setup.
- int Lgsx_ReaderGetCubemapImageBytes (LgsxReaderPtr reader, const wchar_t *pLayer, int level, int p↩ BufferSizes[6], unsigned char *images[6])

  Retrieves the raw image bytes for each face of a cubemap image layer in the active setup.
- int Lgsx_ReaderGetCubeImage (LgsxReaderPtr reader, const wchar_t *pLayer, SCyRgdBdyTxf *txfFrom↩ Setup, int *width, int *height, int *widthInBytes, ImagePixelType *pixelType, unsigned char *cubeImages[6])

  Get the cubemap corresponding to the given layer name for the active setup.
- int Lgsx_GetCubemapFaceOrientation (CubemapFaceIndex faceIndex, QUA4D *pOrientation)

  Returns the default orientation quaternion for a specific cubemap face.
- int Lgsx_ReaderEnumTarget (LgsxReaderPtr reader, int *pNumTarget)

  Collects Targets in the project and returns the count.
- int Lgsx_ReaderEnumTargetOfSetup (LgsxReaderPtr reader, uint64_t setupId, int *pNumTarget)

  Collects Targets in the given Setup and returns the count.
- int Lgsx_ReaderMoveNextTarget (LgsxReaderPtr reader, uint64_t *targetId, CYTARGETINFO *pTarget, LgsxMetadataPtr *pMetadata)

  Advance to the next target in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderGetCurrentTargetLabel (LgsxReaderPtr reader, wchar_t **pLabelPtr)

  Get the full (untruncated) label of the most recently returned Target.
- int Lgsx_ReaderEndTargets (LgsxReaderPtr reader)

  Frees the active Targets collection.
- int Lgsx_ReaderEnumRuns (LgsxReaderPtr reader, int *pNumRun)

  Collects Runs in the project and returns the count.

- int Lgsx_ReaderMoveNextRun (LgsxReaderPtr reader, uint64_t *runId, CYTRAJECTORYINFO *info, int *nVertex, CYTRAJECTORYVERTEX **ppPath, LgsxMetadataPtr *pMetadata)

  Advance to the next Run in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderMoveNextRun2 (LgsxReaderPtr reader, uint64_t *runId, CYTRAJECTORYINFO *info, int *nVertex, CYTRAJECTORYVERTEX **ppPath, LgsxMetadataPtr *pMetadata, wchar_t **pJobNamePtr, wchar_t **pRunNamePtr)

  Advance to the next Run and return full (untruncated) job and run names alongside the complete Run info.
- int Lgsx_ReaderEndRuns (LgsxReaderPtr reader)

  Frees the active Run collection.
- int Lgsx_ReaderRunGetGuid (LgsxReaderPtr reader, char **pGuidPtr)

  Gets the Run GUID for the current Run in the enumeration (Lgsx_ReaderEnumRuns).
- int Lgsx_ReaderHasRunSensorInfo (LgsxReaderPtr reader, bool *pHasInfo)

  Check if current run has SensorInfo as per Lgsx_ReaderMoveNextRun() context.
- int Lgsx_ReaderGetRunSensorInfo (LgsxReaderPtr reader, SensorInfoHandle *pHandle)

  Get SensorInfo of the current run as per Lgsx_ReaderMoveNextRun() context.
- int Lgsx_ReaderEnumRunProcessingInfos (LgsxReaderPtr reader, int *pNumProcessingInfo)

  Enumerate ProcessingInfo entries associated with current run as per Lgsx_ReaderMoveNextRun() context and return the count.
- int Lgsx_ReaderGetRunProcessingInfo (LgsxReaderPtr reader, int pos, ProcessingInfoHandle *pHandle)

  Get ProcessingInfo entry at the specified position for current run as per Lgsx_ReaderMoveNextRun() context.
- int Lgsx_ReaderEnumRunScans (LgsxReaderPtr reader, int *pNumScans)

  Collects Scans for the current Run's Setup and returns the count.
- int Lgsx_ReaderGetRunScan (LgsxReaderPtr reader, int pos, ScanHandle *pHandle)

  Read Scan data at the given position in the active run-scoped collection.
- int Lgsx_ReaderGetLUTImage (LgsxReaderPtr reader, wchar_t *typeName, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the current run context.
- int Lgsx_ReaderGetLUTImage2 (LgsxReaderPtr reader, wchar_t **pTypeNamePtr, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the desired type, without buffer truncation.
- int Lgsx_ReaderGetLUTImageOfSetup (LgsxReaderPtr reader, uint64_t setupId, wchar_t *typeName, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the given Setup.
- int Lgsx_ReaderGetLUTImageOfSetup2 (LgsxReaderPtr reader, uint64_t setupId, wchar_t **pTypeNamePtr, int *nBytes, void **lut)

  Gets the Setup Camera LUT (Lookup Table) for the given Setup, without buffer truncation.
- int Lgsx_ReaderEnumSetupCamera (LgsxReaderPtr reader, int *pNumSetupCamera)

  Collects Setup Cameras for the current setup and returns the count.
- int Lgsx_ReaderEnumSetupCameraOfSetup (LgsxReaderPtr reader, uint64_t setupId, int *pNumSetup↩ Camera)

  Collects Setup Cameras for the given Setup returns the count.
- int Lgsx_ReaderEndSetupCameras (LgsxReaderPtr reader)

  Frees the active Setup Cameras collection.
- int Lgsx_ReaderMoveNextSetupCamera (LgsxReaderPtr reader, uint64_t *pSetupCameraId, CYSETUPCAMERAINFO *pSetupCamera, LgsxMetadataPtr *pMetadata)

  Advance to the next Setup Camera in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderGetCurrentSetupCameraName (LgsxReaderPtr reader, wchar_t **pNamePtr)

  Get the full (untruncated) name of the most recently returned Setup Camera.
- int Lgsx_ReaderGetSetupCameraImage (LgsxReaderPtr reader, wchar_t *typeName, int *nBytes, void **blob, int *nThumbBytes, void **thumbBlob)

  Get the Setup Camera image and thumbnail image corresponding to the given layer name for the active Setup Camera.
- int Lgsx_ReaderGetSetupCameraImage2 (LgsxReaderPtr reader, wchar_t **pTypeNamePtr, int *nBytes, void **blob, int *nThumbBytes, void **thumbBlob)

Get the Setup Camera image and thumbnail image for the active Setup Camera, without buffer truncation.

- int Lgsx_ReaderEnumGeoTags (LgsxReaderPtr reader, int *pNumGeoTag)

  Collects GeoTags in the project and returns the count.
- int Lgsx_ReaderEnumGeoTagsOfSetup (LgsxReaderPtr reader, uint64_t setupId, int *pNumGeoTag)

  Enumerate all geotags that are visible or related to the given setup.
- int Lgsx_ReaderMoveNextGeoTag (LgsxReaderPtr reader, uint64_t *geoTagId, CYGEOTAG *pGeoTag, LgsxMetadataPtr *pMetadata)

  Advance to the next GeoTag in the active collection and return the ID, info, and metadata.
- int Lgsx_ReaderEndGeoTags (LgsxReaderPtr reader)

  Frees the active GeoTags collection.
- int Lgsx_ReaderGetGeoTagName (LgsxReaderPtr reader, uint64_t geoTagId, wchar_t **pNamePtr)

  Get name attribute of the GeoTag with the given ID.
- int Lgsx_ReaderHasGeoTagDescription (LgsxReaderPtr reader, uint64_t geoTagId, bool *pStatus)

  Check if the GeoTag with the given ID has a description attribute.
- int Lgsx_ReaderGetGeoTagDescription (LgsxReaderPtr reader, uint64_t geoTagId, wchar_t **pDescPtr)

  Get description attribute of the GeoTag with the given ID.
- int Lgsx_ReaderGetGeoTag (LgsxReaderPtr reader, int pos, GeoTagHandle *pHandle)

  Read GeoTag data at the given position in the active collection.
- int Lgsx_GeoTagRelease (GeoTagHandle *pHandle)

  Release the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetId (const GeoTagHandle handle, uint64_t *pGeoTagId)

  Get the numeric ID of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetGuid (const GeoTagHandle handle, const char **pGuidPtr)

  Get the GUID of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetName (const GeoTagHandle handle, const wchar_t **pNamePtr)

  Get name attribute of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagHasDescription (const GeoTagHandle handle, bool *pStatus)

  Check if the GeoTag has a description attribute.
- int Lgsx_GeoTagGetDescription (const GeoTagHandle handle, const wchar_t **pDescPtr)

  Get description attribute of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetPosition (const GeoTagHandle handle, PNT3D *pGeoTagPosition)

  Get the position of the GeoTag.
- int Lgsx_GeoTagGetRelativePosition (const GeoTagHandle handle, PNT3D *pGeoTagPosition)

  Get the relative position of the GeoTag.
- int Lgsx_GeoTagHasCameraPosition (const GeoTagHandle handle, bool *pStatus)

  Check if the GeoTag has a camera position.
- int Lgsx_GeoTagGetCameraPosition (const GeoTagHandle handle, PNT3D *pCameraPosition)

  Get the camera position of the GeoTag.
- int Lgsx_GeoTagGetCameraRelativePosition (const GeoTagHandle handle, PNT3D *pCameraPosition)

  Get the relative camera position of the GeoTag.
- int Lgsx_GeoTagGetAnchor (const GeoTagHandle handle, GeotagAnchorType *pAnchorType, uint64_t *p↩ AnchorId)

  Get the anchor of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetCreationTimestamp (const GeoTagHandle handle, uint64_t *pCreationTimestamp)

  Get the creation timestamp of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetModificationTimestamp (const GeoTagHandle handle, uint64_t *pModification↩ Timestamp)

  Get the modification timestamp of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagGetMetadata (const GeoTagHandle handle, LgsxMetadataPtr *pMetadata)

  Get the metadata of the GeoTag data pointer returned by Lgsx_ReaderGetGeoTag().
- int Lgsx_GeoTagEnumCategoryEntries (const GeoTagHandle handle, int *pNumEntries)

Enumerate the Category Entries associated with the GeoTag and return the count.

- int Lgsx_GeoTagGetCategoryEntry (const GeoTagHandle handle, int pos, CategoryEntryHandle *pEntry↩ Handle)

  Get a Category Entry at the specified position for the GeoTag.
- int Lgsx_GeoTagEnumFieldEntries (const GeoTagHandle handle, int *pNumEntries)

  Enumerate the Field Entries associated with the GeoTag and return the count.
- int Lgsx_GeoTagGetFieldEntry (const GeoTagHandle handle, int pos, FieldEntryHandle *pEntryHandle)

  Get a Field Entry at the specified position for the GeoTag.
- int Lgsx_GeoTagEnumAssets (const GeoTagHandle handle, int *pNumAssets)

  Collects Assets associated with the given GeoTag and returns the count.
- int Lgsx_ReaderGetGeoTagAsset (LgsxReaderPtr reader, const GeoTagHandle handle, int pos, CYASSET *pAsset, LgsxMetadataPtr *pMetadata)

  Get Info and metadata for the GeoTag Asset at the given position.
- int Lgsx_ReaderGetGeoTagAssetThumbnail (LgsxReaderPtr reader, const GeoTagHandle handle, int pos, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **pImage)

  Get the thumbnail image for GeoTag Asset at the given position.
- int Lgsx_ReaderGetGeoTagAssetHandle (LgsxReaderPtr reader, const GeoTagHandle handle, int pos, AssetHandle *pHandle)

  Acquire an Asset handle for the GeoTag Asset at the given position.
- int Lgsx_GeoTagHasThumbnail (const GeoTagHandle handle, bool *pHasThumbnail)

  Checks if GeoTag has a thumbnail.
- int Lgsx_GeoTagGetThumbnail (const GeoTagHandle handle, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **pImage)

  Get the thumbnail image for the GeoTag itself.
- int Lgsx_ReaderEnumSitemaps (LgsxReaderPtr reader, int *pNumSitemap)

  Collects Sitemaps in the project and returns the count.
- int Lgsx_ReaderGetSitemapHandle (LgsxReaderPtr reader, int pos, SitemapHandle *pHandle)

  Get a SitemapHandle at the given position in the active collection.
- int Lgsx_SitemapRelease (SitemapHandle *pHandle)

  Release a SitemapHandle returned by Lgsx_ReaderGetSitemapHandle().
- int Lgsx_SitemapGetId (const SitemapHandle handle, uint64_t *pId)

  Get the numeric ID of the Sitemap.
- int Lgsx_SitemapGetGuid (const SitemapHandle handle, const char **pGuid)

  Get the GUID of the Sitemap.
- int Lgsx_SitemapGetName (const SitemapHandle handle, const wchar_t **pName)

  Get the display name of the Sitemap.
- int Lgsx_SitemapGetOrderingIndex (const SitemapHandle handle, int *pIndex)

  Get the display ordering index of the Sitemap.
- int Lgsx_SitemapGetIsMaster (const SitemapHandle handle, bool *pIsMaster)

  Returns true if this is a master sitemap.
- int Lgsx_SitemapGetMetadata (const SitemapHandle handle, LgsxMetadataPtr *pMetadata)

  Get the metadata of the Sitemap.
- int Lgsx_SitemapHasImage (const SitemapHandle handle, SitemapImageType imageType, bool *pHas↩ Image)

  Check whether a sitemap image of the given type exists.
- int Lgsx_SitemapImageGetHandle (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, AssetHandle *pAssetHandle)

  Acquire an AssetHandle for one of the sitemap images.
- int Lgsx_SitemapImageGetMetadata (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, LgsxMetadataPtr *pMetadata)

  Get metadata for one of the sitemap images from image file metadata.

- int Lgsx_SitemapImageGetIsBlank (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, bool *pIsBlank)

  Returns true if the given sitemap image is a blank placeholder.
- int Lgsx_SitemapImageGetCorners (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, CYSITEMAPIMAGECORNERS2D *pCorners, int *pFlags)

  Get the world-space image footprint for one of the sitemap images.
- int Lgsx_SitemapEnumItems (const SitemapHandle handle, int *pCount)

  Collect all items belonging to the sitemap and return the count.
- int Lgsx_SitemapGetItem (const SitemapHandle handle, int pos, SitemapItemHandle *pItemHandle)

  Get the sitemap item at the given position.
- int Lgsx_SitemapItemGetId (const SitemapItemHandle handle, uint64_t *pId)

  Get the numeric ID of a sitemap item.
- int Lgsx_SitemapItemGetType (const SitemapItemHandle handle, SitemapItemType *pType)

  Get the type of a sitemap item.
- int Lgsx_SitemapItemGetGuid (const SitemapItemHandle handle, const char **pGuid)

  Get the GUID of a sitemap item.
- int Lgsx_SitemapItemGetPose (const SitemapItemHandle handle, SCyRgdBdyTxf *pPose)

  Get the resolved project-space pose of a sitemap item.
- int Lgsx_SitemapGetActiveCoordSystemId (const SitemapHandle handle, uint64_t *pCoordSysId)

  Get the numeric ID of the active UCS associated with the sitemap.
- int Lgsx_SitemapImageRead (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, int *pWidth, int *pHeight, int *pWidthInBytes, ImagePixelType *pPixelType, unsigned char **p↩Image)

  Read a selected sitemap image through the decoded-image path.
- int Lgsx_SitemapImageReadBinary (LgsxReaderPtr reader, const SitemapHandle handle, SitemapImageType imageType, wchar_t **pImageType, int *pWidth, int *pHeight, unsigned char **pData, int *pSize)

  Retrieves the raw encoded image bytes for a sitemap image.
- int Lgsx_ReaderEndSitemaps (LgsxReaderPtr reader)

  Frees the active Sitemaps collection.
- int Lgsx_ReaderGetModelNodes (LgsxReaderPtr reader, int *pNumModelNode, uint64_t **ppModelNode↩Ids)

  Retrieve the Model Nodes for this project.
- int Lgsx_ReaderGetModelNodeInfo (LgsxReaderPtr reader, uint64_t nodeId, CYMODELNODEINFO *p↩NodeInfo, uint64_t **ppChildIds)

  Retrieve the info for a given Model Node.
- int Lgsx_ReaderGetModelNodeInfo2 (LgsxReaderPtr reader, uint64_t nodeId, CYMODELNODEINFO *p↩NodeInfo, uint64_t **ppChildIds, wchar_t **pNamePtr)

  Retrieve the info for a given Model Node, with the full (untruncated) node name.
- int Lgsx_ReaderGetModelInfo (LgsxReaderPtr reader, uint64_t modelId, CYMODELINFO *pModelInfo, LgsxMetadataPtr *pMetadata, uint64_t **ppRefAssetIds)

  Retrieve the info for a given Model.
- int Lgsx_ReaderGetModelInfo2 (LgsxReaderPtr reader, uint64_t modelId, CYMODELINFO *pModelInfo, LgsxMetadataPtr *pMetadata, uint64_t **ppRefAssetIds, wchar_t **pNamePtr)

  Retrieve the info for a given Model, with the full (untruncated) model name.
- int Lgsx_ReaderGetPropertyTypeInfo (LgsxReaderPtr reader, int typeMask, const wchar_t *name, int *pn↩Bytes)

  Retrieve Property Type information from the Point Cloud.
- int Lgsx_ReaderEnumPoints (LgsxReaderPtr reader, int desiredTypes, int subSample, bool visible)

  Collects points in the Point Cloud.
- int Lgsx_ReaderEnumPointsEx (LgsxReaderPtr reader, int desiredTypes, int subSample, bool visible, bool scrictOrder)

  Collects points in the Point Cloud.

- int Lgsx_InitEnumPointsConfig (EnumPointsConfigHandle *pHandle)

  Allocate and initialise an EnumPointsConfigHandle with default values.
- int Lgsx_ReleaseEnumPointsConfig (EnumPointsConfigHandle *pHandle)

  Release an EnumPointsConfigHandle previously allocated by Lgsx_InitEnumPointsConfig().
- int Lgsx_EnumPointsConfigSetDesiredTypes (const EnumPointsConfigHandle handle, int desiredTypes)

  Set the desired property-type mask on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetSubSample (const EnumPointsConfigHandle handle, int subSample)

  Set the sub-sample rate on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetVisible (const EnumPointsConfigHandle handle, bool visible)

  Set the visibility filter on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetStrictOrder (const EnumPointsConfigHandle handle, bool strictOrder)

  Set the strict-order flag on an EnumPointsConfigHandle.
- int Lgsx_EnumPointsConfigSetMaxMemoryHintBytes (const EnumPointsConfigHandle handle, uint64_t maxMemoryHintBytes)

  Set the maximum memory hint on an EnumPointsConfigHandle.
- int Lgsx_ReaderEnumPointsEx2 (LgsxReaderPtr reader, const EnumPointsConfigHandle config)

  Collects points in the Point Cloud using the given configuration.
- int Lgsx_ReaderMoveNextPoints (LgsxReaderPtr reader, int chunkSize, int bmpFormat, double *points, float *intens, uchar *colors, float *normals)

  Advances to the next group of points in the Point Cloud and returns the basic properties.
- int Lgsx_ReaderMoveNextFields (LgsxReaderPtr reader, int chunkSize, int bmpFormat, void *ppData[ ])

  Advances to the next group of points in the Point Cloud and returns the desired properties.
- int Lgsx_ReaderExtractPointCloud (LgsxReaderPtr reader, char *inOutPathname)

  Extract the entire point cloud from the LGSx file and store it to the specified path/file.
- int Lgsx_ReaderPointCloudFileOpen (LgsxReaderPtr reader, PointCloudFileHandle *pHandle)

  Opens a virtual file handle to access the raw point cloud data.
- int Lgsx_PointCloudFileClose (PointCloudFileHandle *pHandle)

  Closes the Point Cloud file returned by Lgsx_ReaderPointCloudFileOpen().
- int Lgsx_PointCloudFileGetSize (const PointCloudFileHandle handle, uint64_t *pBytes)

  Get the size in bytes of the Point Cloud file.
- int Lgsx_PointCloudFileRead (const PointCloudFileHandle handle, uint64_t offset, uint64_t numBytes, void *pBuffer, uint64_t *pBytesRead)

  Read a chunk of raw Point Cloud file into the provided buffer.
- int Lgsx_ReaderGetClassModels (LgsxReaderPtr reader, wchar_t **ppClassModels)

  Retrieve the classification model names for this Point Cloud.
- int Lgsx_ReaderGetClassVisibles (LgsxReaderPtr reader, int modelIdx, wchar_t **ppClassNames, int **ppClassVisibles, int **ppClassColors)

  Retrieve the classification properties for the given classification model index.
- int Lgsx_ReaderEnumCoordSystems (LgsxReaderPtr reader, int *pNumCs)

  Collects User Coordinate Systems.
- int Lgsx_ReaderMoveNextCoordSystems (LgsxReaderPtr reader, bool *pActive, wchar_t *pName, int maxName, wchar_t *pWkt, int maxWkt, SCyRgdBdyTxf *pTxf)

  Advances to the next User Coordinate System (UCS) object and returns its properties.
- int Lgsx_ReaderEndCoordSystems (LgsxReaderPtr reader)

  Frees the active User Coordinate Systems collection.
- int Lgsx_ReaderMoveNextCoordSystems2 (LgsxReaderPtr reader, bool *pActive, wchar_t **pNamePtr, wchar_t **pWktPtr, SCyRgdBdyTxf *pTxf)

  Advances to the next User Coordinate System (UCS) object and returns its properties.
- int Lgsx_ReaderGetCurrentCoordSystemId (LgsxReaderPtr reader, uint64_t *pId)

  Get the numeric ID of the User Coordinate System most recently returned by Lgsx_ReaderMoveNextCoordSystems2().
- int Lgsx_ReaderGetCurrentCoordSystemGuid (LgsxReaderPtr reader, char **pGuidPtr)

  Get the GUID of the User Coordinate System most recently returned by Lgsx_ReaderMoveNextCoordSystems2().

8.2.1 Detailed Description

Index

