StorageTek

Library Control Interface (SCI) Reference Guide

E76473-01

May 2018

ORACLE

StorageTek Library Control Interface (SCI) Reference Guide

E76473-01

Contents

3 SCI Objects

4 Enumeration Types

Preface

The StorageTek Library Control Interface (SCI) is a programmatic Web Services (SOAP) interface provided by the StorageTek SL4000 Modular Library System. Applications can use SCI to control the library. This document is intended for SCI developers.

WSDL URLs

Inbound SCI: `http://<hostname>/WebService/1.0.0?wsdl`

Outbound SCI: `http://<hostname>/OutboundWebService/`

Protocol Negotiation

You can connect to a base webservice (`http://<hostname>/WebService/base?wsdl`) that supports a single method: discover(). This method returns a list of supported web service protocols that the library understands. Use this method to ensure that the client code is compatible with the library. Currently there is only a 1.0.0 version WebService, which uses the SOAP 1.2 protocol.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documentation

Go to the Tape Storage section of the Oracle Help Center (http://docs.oracle.com/en/storage/#tape) for additional SL4000 documentation:

- *SL4000 Library Guide*
- *SL4000 SCSI Reference Guide*
- *SL4000 Safety and Compliance Guide*
- *SL4000 Licensing Information User Guide*

- *SL4000 Security Guide*

1

Library Inbound Methods

Request Methods

getRequest()

Returns the status of a submitted request. This method allows the client to query for the completion status of an asynchronous operational request.

Inputs	Outputs	Roles	Errors
long `requestId`	RequestDto	All	None

getRequests()

Returns a list of RequestDto ordered chronologically based on the request's createDateTime from newest to oldest. If you specify null for `firstTimeStamp`, the library returns the newest "`count`" requests, otherwise the returned requests will be before `firstTimeStamp`.

Inputs	Outputs	Roles	Errors
int `count` *date* `firstTimeStamp`	List of RequestDto	All	None

cancelRequest()

Cancels a submitted request.

Inputs	Outputs	Roles	Errors
long `requestId`	RequestDto	C2,S1,I	NotFoundException

getLocalizedRequestMessages()

Returns an ordered set of request output messages localized to the specified locale. Output messages are in chronoligical order, oldest to newest. Use getSupportedLocales() for a list of valid locales.

Inputs	Outputs	Roles	Errors
long `requestId`, *string* `locale`	List of RequestOutputMessage Dto	All	NotFoundException (request, locale)

getSupportedLocales()

Returns a list of strings for the locales supported for email messages and for getLocalizedRequestMessages().

Inputs	Outputs	Roles	Errors
None	List of *string* `supportedLocales`	All	None

waitForRequest()

Waits for a submitted request to complete. Returns when the request completes or when the timeout expires.

Inputs	Outputs	Roles	Errors
long `requestId`, *int* `timeout` (in seconds)	RequestDto	All	NotFoundException

Library Complex Methods

Use the library complex methods to manage the library complex as a whole.

- getLibraries()
- getLibraryComplex()
- getLibraryComplexCartridges()
- getLibraryComplexCells()
- getLibraryComplexCleaningCartridges()
- getLibraryComplexDevices()
- getLibraryComplexDevicesByControlState()
- getLibraryComplexDrives()
- getLibraryComplexDriveTrays()
- getLibraryComplexModules()
- getLibraryComplexSlots()
- setLibraryComplexName()
- setLibraryComplexBootOptions()
- resetLibraryComplexToFactoryDefault()
- exportLibraryComplexConfig()
- importLibraryComplexConfig()
- updateTimeSettings()

getLibraries()

Returns a list of information for each library in the complex.

Inputs	Outputs	Roles	Errors
None	List of LibraryDto	All	Authentication Exception, Yapi Exception

getLibraryComplex()

Returns information about the library complex.

Inputs	Outputs	Roles	Errors
None	LibraryComplexDto	All	Authentication Exception, Yapi Exception

getLibraryComplexCartridges()

Returns a list of cartridges for the entire library complex. The LibraryComplexDto includes a count of the number of cartridges in the library complex.

Inputs	Outputs	Roles	Errors
CellTypeSelector `cellType`, *int* `first`, *int* `count`	List of CartridgeDto	All	Authentication Exception, Yapi Exception

getLibraryComplexCells()

Returns a list of cells and their contents for the entire library complex. The LibraryComplexDto includes a count of the number of cells in the library complex. The CellTypeSelector determines the type of cells returned.

Inputs	Outputs	Roles	Errors
CellTypeSelector `cellType`, *int* `first`, *int* `count`	List of CellDto	All	Authentication Exception, Yapi Exception

getLibraryComplexCleaningCartridges()

Returns a list of cleaning cartridges for the entire library complex. The LibraryComplexDto includes a count of the number of cleaning cartridges in the library complex.

Inputs	Outputs	Roles	Errors
int `first`, *int* `count`	List of CartridgeDto	C1, S1, I	Authentication Exception, Yapi Exception

getLibraryComplexDevices()

Returns a list of all devices in the library complex.

Inputs	Outputs	Roles	Errors
DeviceTypeSelector `deviceType`, *int* `first`, *int* `count`	List of DeviceDto	All	Authentication Exception, Yapi Exception

getLibraryComplexDevicesByControlState()

Returns a list of all devices in the library complex with the specified control state.

Inputs	Outputs	Roles	Errors
DeviceTypeSelector `deviceType`, *ControlState* `controlState` *int* `first`, *int* `count`	List of DeviceDto	All	Authentication Exception, Yapi Exception

getLibraryComplexDrives()

Returns a list of all drives in the library complex. The LibraryComplexDto includes a count of the number of drives in the library complex.

Inputs	Outputs	Roles	Errors
int `first`, *int* `count`	List of DriveDto	All	Authentication Exception, Yapi Exception

getLibraryComplexDriveTrays()

Returns a list of all drives in the library complex. The LibraryComplexDto includes a count of the number of drive trays in the library complex.

Inputs	Outputs	Roles	Errors
int `first`, *int* `count`	List of DriveTrayDto	All	Authentication Exception, Yapi Exception

getLibraryComplexModules()

Returns a list of modules for the entire library complex. The LibraryComplexDto includes a count of the number of modules in the library complex.

Inputs	Outputs	Roles	Errors
None	List of ModuleDto	All	Authentication Exception, Yapi Exception

getLibraryComplexSlots()

Returns a list of device slots in the library complex. If a slot contains a device, the device is also returned.

Inputs	Outputs	Roles	Errors
DeviceTypeSelector `deviceType`, *int* `first`, *int* `count`	List of SlotDto	All	Authentication Exception, Yapi Exception

setLibraryComplexName()

Updates the library complex name. Maximum length is 256 characters.

Inputs	Outputs	Roles	Errors
string `name`	None	C2,S1,I	None

setLibraryComplexBootOptions()

Sets the library startup options. These options affect the next library startup, and only the next library startup. If `suppressHasBeenOpened=true`, the library does not audit on startup even if you opened the door while the library was off. Only set this option to true if you did not change the cartridge inventory.

Set `checkLibraryConfiguartion=true` after changing the library module configuration. Set this option to true when adding or removing modules, drive array assemblies, or rotary CAPs. This will force the library to re-examine the modules and rebuild the inventory of modules, devices, and cartridges. The library will perform a full audit.

Inputs	Outputs	Roles	Errors
boolean `suppressHasBeenOpened` *boolean* `checkLibraryConfiguartion`	None	C3,S2,I	Authentication Exception, Yapi Exception

resetLibraryComplexToFactoryDefault()

Resets the library complex to factory default settings. You cannot undo this operation. After performing this operation the library will be in the same state as when it left the factory. Oracle service will be required to initialize the library.

Inputs	Outputs	Roles	Errors
None	None	S3,I	Authentication Exception, Yapi Exception

exportLibraryComplexConfig()

Exports the complete configuration of the library complex to a file. This method uses MTOM to transfer the data to the caller. The client must implement the DataHandler to receive the data from the library.

Inputs	Outputs	Roles	Errors
None	DataHandler	C1,S1,I	Authentication Exception, Yapi Exception

importLibraryComplexConfig()

Imports the complete configuration of the library complex from a file. This method uses MTOM to transfer the data to the library. The client must implement the DataHandler to send data to the library. This method deletes ALL existing library configuration information. You cannot undo this operation. The physical configuration of the library doing the import operation must be identical to the library where the configuration was exported.

Inputs	Outputs	Roles	Errors
DataHandler `dataHandler`	None	C3,S2,I	Authentication Exception, Yapi Exception, InvalidConfig Exception

updateTimeSettings()

Updates time related settings for the library complex. This includes whether or not to use NTP, a list of NTP servers (at least one required if using NTP), and date and time if not using NTP.

Inputs	Outputs	Roles	Errors
string `ntpServers`, *boolean* `ntpEnabled`, *boolean* `forceEnabled`	TimeSettingsDto	C2,S1,I	Authentication Exception, Yapi Exception, InvalidConfig Exception

Library Methods

Use these methods to view and set the configuration for a library.

- getLibrary()
- getLibraryCells()
- getLibraryCartridges()
- getLibraryDevices()
- getLibraryDrives()
- getLibraryDriveTrays()

- getCell()
- getCellByLrcsr()
- setLibraryName()
- changeLibraryState()
- enableRedundantElectronics()

getLibrary()

Returns a information for the specified library.

Inputs	Outputs	Roles	Errors
long `libraryId`	LibraryDto	All	Authentication Exception, Yapi Exception

getLibraryCells()

Returns a list of cells and their contents for the specified library. The list starts at 0. The `first` parameter allows the user to page through the list by providing a different starting element in the list.

Inputs	Outputs	Roles	Errors
long `libraryId` *CellTypeSelector* `cellType`, *int* `first`, *int* `count`	List of CellDto for the library	All	Authentication Exception, Yapi Exception

getLibraryCartridges()

Returns a list of cartridges for a library. The list starts at 0. The `first` parameter allows the user to page through the list by providing a different starting element in the list.

Inputs	Outputs	Roles	Errors
long `libraryId` *CellTypeSelector* `cellType`, *int* `first`, *int* `count`	List of CartridgeDto for the library	All	Authentication Exception, Yapi Exception

getLibraryDevices()

Returns a list of devices in the specified library. The list starts at 0. The `first` parameter allows you to page through the list by providing a different starting element in the list.

Inputs	Outputs	Roles	Errors
long `libraryId` *DeviceTypeSelector* `deviceType`, *int* `first`, *int* `count`	List of DeviceDto for the library	All	Authentication Exception, Yapi Exception

getLibraryDrives()

Returns a list of drives in the library. The list starts at 0. The `first` parameter allows you to page through the list by providing a different starting element in the list.

Inputs	Outputs	Roles	Errors
long `libraryId` *int* `first`, *int* `count`	List of DriveDto for the library	All	None

getLibraryDriveTrays()

Returns a list of drive trays for a library. The list starts at 0. The `first` parameter allows you to page through the list by providing a different starting element in the list.

Inputs	Outputs	Roles	Errors
long `libraryId` *int* `first`, *int* `count`	List of DriveTrayDto for the library	All	Authentication Exception, Yapi Exception

getCell()

Returns a single cell and its contents.

Inputs	Outputs	Roles	Errors
long `cellId`	CellDto	All	Authentication Exception, Yapi Exception

getCellByLrcsr()

Returns a single cell based on the library, rail, column, side, row numbers.

Inputs	Outputs	Roles	Errors
int `libraryNumber`, *int* `railNumber`, *int* `columnNumber`, *int* `sideNumber`, *int* `rowNumber`	CellDto	All	Authentication Exception, Yapi Exception

setLibraryName()

Updates the library name. Maximum name length is 256 characters.

Inputs	Outputs	Roles	Errors
long `libraryId` *string* `name`	None	C3,S2,I	Authentication Exception, Yapi Exception

changeLibraryState()

Takes the library offline or brings the library online. If the library cannot change state immediately, the call returns a RequestDto so you can track the state of the request. When taking the library offline, the library completes any in-progress moves, but rejects new moves.

Inputs	Outputs	Roles	Errors
long `libraryId` *ControlState* `state`	RequestDto	C2,S1,I	Authentication Exception, Yapi Exception

enableRedundantElectronics()

Informs the library to use dual controller cards. If you enable RE with only one controller installed, the library will assume one controller has failed and will generate a fault. If there are two controllers installed and you do not enable RE, the library only uses the card in library controller slot 1.

Inputs	Outputs	Roles	Errors
boolean `enable`	None	S1,I	None

Module Methods

Most module methods require a `moduleId`. You can use the output of getLibraryComplexModules() to obtain the `moduleId`.

- getModule()
- getModuleCartridges()
- getModuleCells()
- getModuleDevices()
- getModuleDriveTrays()
- getModuleSlots()

getModule()

Returns module information for a specific `moduleId`.

Inputs	Outputs	Roles	Errors
long `moduleId`	ModuleDto	All	None

getModuleCartridges()

Returns a list of cartridges for a module. The ModuleDto includes a count of the number of cartridges in the module.

Inputs	Outputs	Roles	Errors	Status
long `moduleId`, *int* `first`, *int* `count`	List of CartridgeDto for the module	All	None	Functional. returns empty list

getModuleCells()

Returns a list of cells and their contents for a module. The ModuleDto includes a count of the number of cells in the module. The CellTypeSelector determines the type of cells returned.

Inputs	Outputs	Roles	Errors	Status
long `moduleId`, *CellTypeSelector* `cellType`, *int* `first`, *int* `count`	List of CellDto for the module	All	NotFoundException (module)	Functional. returns empty list

getModuleDevices()

Returns a list of devices in the module.

Inputs	Outputs	Roles	Errors	Status
long `moduleId`, *DeviceTypeSelector* `deviceType`, *int* `first`, *int* `count`	List of DeviceDto for the module	All	NotFoundException (module)	Functional. Missing Service

getModuleDriveTrays()

Returns a list of drive trays present in the specified module.

Inputs	Outputs	Roles	Errors	Status
long `moduleId`, *int* `first`, *int* `count`	List of DriveTrayDto for the module	All	NotFoundException (module)	Functional. returns empty list

getModuleSlots()

Returns a list of device slots in the module. If a slot contains a device, this method also returns the device.

Inputs	Outputs	Roles	Errors
long `moduleId`, *DeviceTypeSelector* `deviceType`, *int* `first`, *int* `count`	List of SlotDto for the module	All	NotFoundException (module)

Device Methods

Most device methods require a `deviceId`. You can use getLibraryComplexDevices(), getLibraryDevices(), or getModuleDevices() to obtain a `deviceId`.

- getDevice()
- getDeviceCartridges()
- getDeviceCells()
- getDeviceSensors()
- getDeviceSlots()
- getDeviceTelemetry()

- getContainedDevices()
- getFruIdData()
- bringDeviceOnline()
- takeDeviceOffline()
- enableOnlining()
- inhibitOnlining()
- turnDeviceOn()
- turnDeviceOff()
- setLedState()
- locateDevice()
- pingDevice()
- resetDevice()

getDevice()

Returns a DeviceDto for a specific `deviceId`.

Inputs	Outputs	Roles	Errors
long `deviceId`	DeviceDto	All	None

getDeviceCartridges()

Returns a list of cartridges for a device.

Inputs	Outputs	Roles	Errors
long `deviceId`, *int* `first`, *int* `count`	List of CartridgeDto for the device	All	NotFountException (device)

getDeviceCells()

Returns a list of cells and their contents for the device. The type parameter determines the type of cells returned

Inputs	Outputs	Roles	Errors
long `deviceId`, *CellTypeSelector* `cellType`, *int* `first`, *int* `count`	List of CellDto for the device	All	NotFountException (device)

getDeviceSensors()

Inputs	Outputs	Roles	Errors
long `deviceId`	List of SensorDto	All	NotFountException (device)

getDeviceSlots()

Returns a list of slots in the device. If a slot contains a device, the method also returns the device.

Inputs	Outputs	Roles	Errors
long `deviceId`, *int* `first`, *int* `count`	List of SlotDto for the slots in the device	All	NotFountException (device)

getDeviceTelemetry()

Returns device telemetry for the specified `deviceId`. Returns `count` measurements taken after the `timestamp`.

Inputs	Outputs	Roles	Errors
long `deviceId`, *long* `sensorId`, *date* `timestamp`, *int* `count`	TelemetryDto	All	NotFountException (device)

getContainedDevices()

Returns a list of devices contained within a device. For instance, the card cage in the Base module will contain multiple devices. The `deviceId` you input corresponds to the `parentDeviceId` that returns in the DeviceDto.

Inputs	Outputs	Roles	Errors
long `deviceId`, *DeviceTypeSelector* `deviceType`, *int* `first`, *int* `count`	List of DeviceDto contained in the device	All	NotFountException (device)

getFruIdData()

Returns FRU data for a specific device.

Inputs	Outputs	Roles	Errors
long `deviceId`	FruIdDto	S1, I	None

bringDeviceOnline()

Brings a device online (changes the control state). When using this method to bring a rotational CAP online, all CAPs upstream (closer to the Base) of the specified CAP must be online. After the specified CAP is online, the library brings all CAPs downstream (farther from the Base) of the specified CAP online.

Inputs	Outputs	Roles	Errors
long `deviceId`	RequestDto	C2,S1,I	None

takeDeviceOffline()

Takes a device offline (changes the control state). You cannot take an FC port offline.

Inputs	Outputs	Roles	Errors
long `deviceId`, *boolean* `force`	RequestDto	C2,S1,I	None

enableOnlining()

Enables the device in the specified slot to change to the "online" control state. When you insert a device into a slot that has onlining enabled, the library will automatically bring the device online. This is the default behavior for all slots.

Inputs	Outputs	Roles	Errors
DeviceTypeSelector `deviceType`, *int* `moduleNumber`, *int* `slotNumber`	None	C2,S1,I	None

inhibitOnlining()

Prevents the device in the specified slot from changing to the "online" control state. When you insert a device into a slot that has onlining inhibited, the library will hold the device offline. You must manually bring the device online.

Inputs	Outputs	Roles	Errors
DeviceTypeSelector `deviceType`, *int* `moduleNumber`, *int* `slotNumber`	None	C2,S1,I	None

turnDeviceOn()

Turns on a device.

Inputs	Outputs	Roles	Errors
long `deviceId`	RequestDto	C2,S1,I	None

turnDeviceOff()

Turns off a device.

Inputs	Outputs	Roles	Errors
long `deviceId`	RequestDto	C2,S1,I	None

setLedState()

Sets the state of an LED.

Inputs	Outputs	Roles	Errors
long `deviceId`, *ServiceIndicatorName* `ledName`, *SeviceIndicatorState* `ledState`	DeviceDto	C2,S1,I	None

locateDevice()

Enables or disables the flashing LED on a FRU. The OK to remove LED flashes. If the device does not have an OK to remove LED, the service LED flashes.

Inputs	Outputs	Roles	Errors
long `deviceId`, *boolean* `enabled`	None	C1,S1,I	NotFountException (device)

pingDevice()

This method attempts to communicate with the specified device to verify the device is not only on the internal network, but also that the device is functioning property. You can only successfully ping devices that communicate on the internal network.

Inputs	Outputs	Roles	Errors
long `deviceId`	PingDeviceResultsDto	C2,S1,I	NotFountException (device)

resetDevice()

Resets a device and returns a request. This method may not complete immediately if the device has in-progress work to complete. The force option overrides any delays and immediately resets the device. You can reset the library controllers, root switches, storage, video, dc converter, rail controller, drive switch, drive controller, robot controller, feature card, robot, and encryption card.

Inputs	Outputs	Roles	Errors
long `deviceId`, *boolean* `force`	RequestDto	C2,S1,I	NotFountException (device) DeviceNotResettableException

Robot Methods

- downloadRobotLog()
- getCellDepth()
- getRailSegmentResourceHistory()
- getRobot()
- getRobots()
- getRobotCalibration()
- getRobotHistory()
- getRobotMetrics()
- getRobotParameters()

- getRobotRange()
- getRobotStatistics()

downloadRobotLog()

Returns device log for the specified robot. The client must provide a DataHandler to receive and store the contents of the log file.

Inputs	Outputs	Roles	Errors
long `deviceId`	DataHandler	C2,S1,I	NotFoundException (robot)

getCellDepth()

Returns the cell depth from the last reach operation. Depth of cells is in mils (thousandths of an inch.)

Inputs	Outputs	Roles	Errors
long `deviceId`	RobotCellDepthDto	S2,I	NotFoundException (robot)

getRailSegmentResourceHistory()

Returns a list of rail segment resource history.

Inputs	Outputs	Roles	Errors
int `first`, *int* `count`	List of RailSegmentResourceUsageDto	S1,I	Authentication Exception, Yapi Exception

getRobot()

Returns information about a robot. This method is similar to getDevice() for a robot device, but RobotDto has more details than DeviceDto.

Inputs	Outputs	Roles	Errors
long `robotId`	RobotDto	All	None

getRobots()

Returns information for all robots in a library.

Inputs	Outputs	Roles	Errors
long `libraryId`	List of RobotDto	All	None

getRobotCalibration()

Returns robot calibration data for a given cell. Even though the method takes a `cellId`, calibration actually refers to the cell array that contains the specified cell.

Inputs	Outputs	Roles	Errors
long `deviceId`, *long* `cellId`	RobotCalibrationDto	S2,I	NotFoundException (robot, cell)

getRobotHistory()

Returns a list of robot movement records.

Inputs	Outputs	Roles	Errors
RobotSelector `robot`, *int* `first`, *int* `count`	List of RobotPositionHistoryDto	S1,I	Authentication Exception, Yapi Exception

getRobotMetrics()

Returns metrics for the robot. The `direction` can be LEFT or RIGHT. Returns a list with one element for each mechanism.

Inputs	Outputs	Roles	Errors
long `deviceId`, *string* `direction`	List of RobotMetricsDto	S1,I	NotFoundException (robot)

getRobotParameters()

Returns robot parameters. These are set by the library during initialization.

Inputs	Outputs	Roles	Errors
long `deviceId`	RobotParametersDto	S1,I	NotFoundException (robot)

getRobotRange()

Returns range data for the robot. Returns a list with one element for each mechanism.

Inputs	Outputs	Roles	Errors
long `deviceId`	List of MotionRangeDto	S1,I	NotFoundException (robot)

getRobotStatistics()

Returns robot telemetry data.

Inputs	Outputs	Roles	Errors
long `deviceId`	RobotStatisticsDto	S1,I	NotFoundException (robot)

CAP Methods

The CAP methods apply to rotational and AEM CAPs.

- changeCapState()
- getCap()
- getCaps()
- getCapRange()
- getCapStatistics()
- lockCap()
- unlockCap()

- openCap()
- closeCap()
- setCapOwner()
- freeCap()
- forceFreeCap()
- getCapPool()
- getCapPools()
- createCapPool()
- deleteCapPool()
- moveCapToCapPool()

See Also

- setPartitionCapPool()
- clearPartitionCapPool()

What is a capHandle?

Before sending any of the CAP operational methods (open, close, and so on), use setCapOwner() to acquire a capHandle and reserve the CAP. Even if the CAP is dedicated to a partition, you must use setCapOwner() to acquire a capHandle.

changeCapState()

Places the CAP online or offline.

Inputs	Outputs	Roles	Errors
long `capId` *ControlState* `state`	RequestDto	C2, S1, I	NotFoundException (CAP)

getCap()

Returns information about a specific CAP.

Inputs	Outputs	Roles	Errors
long `capId`	CapDto	All	None

getCaps()

Returns information about a all CAPs in the library complex.

Inputs	Outputs	Roles	Errors
None	List of CapDto	All	None

getCapRange()

Returns range data for a CAP.

Inputs	Outputs	Roles	Errors
long `capId`	MotionRangeDto	S2,I	NotFoundException (CAP)

getCapStatistics()

Returns CAP telemetry data.

Inputs	Outputs	Roles	Errors
long `capId`	CapStatisticsDto	S1,I	NotFoundException (CAP)

lockCap()

Logically locks a CAP. While in the locked state, the library disables all means of opening the CAP, allowing the robot to safely access the CAP. To lock a CAP, it must be closed, online, and owned by a non-SCSI partition.

Inputs	Outputs	Roles	Errors
long `capHandle`	CapDto	C1,S1,I	InvalidCapHandle, NotClosedException

unlockCap()

Unlocks a CAP so that an operator can open it. When unlocked, the robot cannot access the CAP cells. To unlock a CAP, it must be closed, online, owned by a partition (or controlled by the UI), and not currently in use by the robot.

Inputs	Outputs	Roles	Errors
long `capHandle`	CapDto	C1,S1,I	InvalidCapHandle, NotClosedException

openCap()

Opens an unlocked CAP. Equivalent to pressing the button on a closed CAP.

Inputs	Outputs	Roles	Errors
long `capHandle`	RequestDto	C1,S1,I	InvalidCapHandle, NotLockedException

closeCap()

Closes an open CAP. Equivalent to pressing the button on an open CAP. Once closed, the library audits the CAP cells and adds the cartridges to the partition that owns the CAP. The library moves any cleaning cartridges to system cells.

Inputs	Outputs	Roles	Errors
long `capHandle`	RequestDto	C1,S1,I	InvalidCapHandle, NotLockedException

setCapOwner()

Assigns ownership of a CAP to a specific partition and provides a capHandle. A partition must own a CAP to export or import cartridges with the CAP.

Note: Even if the CAP is dedicated to a partition, you must use setCapOwner() to acquire a capHandle.

Any moves to or from a CAP not owned by the partition will fail. A partition can only own CAPs in the CAP pool assigned to the partition. If a CAP pool is assigned to only one partition, the partition automatically owns the CAPs in that pool. `setCapOwner()` returns `capHandle`, which you can use as input on operations that require an owned CAP.

Inputs	Outputs	Roles	Errors
long `capId`, *long* `partitionId`	*long* `capHandle`	C1,S1,I	NotFoundException (partition, cap), OwnedException, NotLockedException

freeCap()

Removes ownership of the CAP from a partition. The CAP must be locked and empty. An application should free a shared CAP after completing CAP operations so that the CAP is available for other partitions. This method has no effect on CAPs in a dedicated CAP pool.

Inputs	Outputs	Roles	Errors
long `capHandle`	CapDto	C1,S1,I	InvalidCapHandle, NotLockedException, NotEmptyException

forceFreeCap()

Removes ownership of the CAP from a partition. An administrator can remove ownership of a CAP from a partition in the event the library or an application fails to properly release the CAP. This will result in a call to the capOwnershipOverridden() outbound method.

Inputs	Outputs	Roles	Errors
long `capId`	CapDto	C2,S1,I	NotFoundException (CAP), NotLockedException, NotEmptyException

getCapPool()

Returns the CAPs in the CAP pool.

Inputs	Outputs	Roles	Errors
long `CapPoolId`	CapPoolDto	All	None

getCapPools()

Returns a list of all CAP pools, including the CAPs in each pool.

Inputs	Outputs	Roles	Errors
None	List of CapPoolDto	All	None

createCapPool()

Creates a CAP pool. The partitioning hardware activation file (HWAF) must be installed to create CAP pools.

Inputs	Outputs	Roles	Errors
string `name`	CapPoolDto	C2, S1, I	HwafNotPresentException

deleteCapPool()

Deletes a CAP pool if there are no CAPs in the pool.

Inputs	Outputs	Roles	Errors
long `capPoolId`	None	C2, S1, I	NotFoundException (CAP)

moveCapToCapPool()

Moves a CAP from its current CAP pool to another CAP pool.

Inputs	Outputs	Roles	Errors
long `capId` *long* `capPoolId`	CapPoolDto	C2, S1, I	NotFoundException (CAP, CAP pool), CapErrorException

Drive Methods

Drive trays contain a tape drive, an LOD card, an LOID card, a power supply for the drive, and (in some cases) an encryption card. Drive trays slide into drive slots at the back of the library. Drive tray and drive are sometimes used interchangeably.

- getCompatibleDriveTypes()
- getCompatibleMediaTypes()
- getDriveTray()
- getDriveTypes()
- getDrive()
- updateDrive()
- cleanDrive()
- getFirmwareUpgradeActivity()

getCompatibleDriveTypes()

Returns a list of drive types that are compatible with the cartridge `family` and `generation`. The `inLibrary` parameter determines if the method returns all compatible drive types or only those drive types currently found in the library.

Inputs	Outputs	Roles	Errors
string `cartridgeFamily` *string* `cartridgeGeneration` *boolean* `inLibrary`	List of DriveTypeDto	All	None

getCompatibleMediaTypes()

Returns a list of cartridges types compatible with the specified drive family and generation. The `inLibrary` parameter determines if the method returns all compatible cartridge types or only the cartridge types currently in the library.

Inputs	Outputs	Roles	Errors
string `driveFamily` *string* `driveGeneration` *boolean* `inLibrary`	List of CartridgeTypeDto	All	None

getDriveTray()

Returns information about a tray and its contents. This is similar to getDevice() for a drive tray device, but returns a DriveTrayDto which has additional, drive tray specific information.

Inputs	Outputs	Roles	Errors
long `driveTrayId`	DriveTrayDto	All	None

getDriveTypes()

Returns a list of DriveTypeDto, each with info about one drive type. The `inLibrary` parameter determines if the method returns all supported drive types or only those drive types currently found in the library.

Inputs	Outputs	Roles	Errors
boolean `inLibrary`	List of DriveTypeDto	All	None

getDrive()

Returns information about a drive. This is similar to getDevice() for a drive tray device, but returns a DriveDto which has additional, drive specific information.

Inputs	Outputs	Roles	Errors
long `driveId`	DriveDto	All	None

updateDrive()

Updates attributes of a drive. The drive ID from the input drive object is used to locate a drive in the database. All updated attributes in the drive object are then updated into the database.

Inputs	Outputs	Roles	Errors
DriveDto	DriveDto	C2,S1,I	NotFoundException (drive)

cleanDrive()

Cleans the specified drive using a compatible cleaning tape from a system cell.

Inputs	Outputs	Roles	Errors
long `driveId`	RequestDto	C1, S1, I	None

getFirmwareUpgradeActivity()

Display a history of firmware upgrade operations for a drive.

Inputs	Outputs	Roles	Errors
long `driveId`	List of FirmwareUpgradeEvent Dto	C3,S1,I	None

Cartridge Methods

The async parameter determines when the method returns. If `async=false`, the method does not return until the cartridge mounts. If `async=true`, the method returns once the request is submitted. If you submit the request asynchronously, you can use getRequest()to determine when the move completes.

- getCartridgeByCellId()
- getCartridgeByVolser()
- getAllCartridgesByVolser()
- getAllCartridgesByVolsers()
- getCartridgeTypes()
- mountCartridgeByCellId()
- mountCartridgeByVolser()
- moveCartridgeByCellId()
- moveCartridgeByVolser()
- dismountCartridgeByCellId()
- dismountCartridgeByVolser()
- getLostCartridges()
- moveLostCartridge()

getCartridgeByCellId()

Returns information about a single cartridge in the specified a cellId.

Inputs	Outputs	Roles	Errors
long `cellId`	CartridgeDto	All	Authentication Exception, Yapi Exception

getCartridgeByVolser()

Returns information about one cartridge for the given volser. Only use this method when you know there is only one cartridge with that volser. If multiple results are possible, use getAllCartridgesByVolser() instead.

Inputs	Outputs	Roles	Errors
string `volser`	CartridgeDto	All	Authentication Exception, Yapi Exception

getAllCartridgesByVolser()

Returns information about all cartridges for the given volser (use if there are duplicate volsers).

Inputs	Outputs	Roles	Errors
string `volser`	List of CartridgeDto	All	Authentication Exception, Yapi Exception

getAllCartridgesByVolsers()

Returns information about all cartridges for the given set of volsers.

Inputs	Outputs	Roles	Errors
Set of *string* `volsers`	List of CartridgeDto	All	Authentication Exception, Yapi Exception

getCartridgeTypes()

Returns a list of supported cartridge types. The `inLibrary` parameter determines if the call retrieves information on all supported cartridge types (false) or only those currently in the library (true).

Inputs	Outputs	Roles	Errors
boolean `inLibrary`	List of CartridgeTypeDto	All	Authentication Exception, Yapi Exception

mountCartridgeByCellId()

Mounts a cartridge on a tape drive by specifying the cell id of the cartridge source (`cellId`) and the cell id of the destination drive (`destinationCellId`). The `destinationCellId` is the cell id for the drive, not the drive id. If `readOnly` is true, the drive makes the cartridge read-only.

Inputs	Outputs	Roles	Errors
long `sourceCellId`, *long* `destinationCellId`, *boolean* `readOnly`, *boolean* `async`	RequestDto	C2[1],S1,I	IllegalArgumentException if source is a drive or destination is not a drive. NotFoundException (source cell/cartridge (if volser), drive), InvalidMoveException (source empty, destination full)

[1] C2 users can only mount cartridges to drives within the same partition, while C3, service, and installation roles can mount cartridges to any drive.

mountCartridgeByVolser()

Mounts a cartridge on a tape drive by specifying the volume serial number (`volser`) of the cartridge and the cell id of the destination drive (`destinationCellid`). The `destinationCellId` is the cell id for the drive, not the drive id. If `readOnly` is true, the drive makes the cartridge read-only.

Inputs	Outputs	Roles	Errors
string `volser`, *long* `destinationCellId`, *boolean* `readOnly`, *boolean* `async`	RequestDto	C2[1],S1,I	IllegalArgumentException if source is a drive or destination is not a drive. NotFoundException (source cell/cartridge (if volser), drive), InvalidMoveException (source empty, destination full)

[1] C2 users can only mount cartridges to drives within the same partition, while C3, service, and installation roles can mount cartridges to any drive.

moveCartridgeByCellId()

Moves a cartridge from the source cell (`cellId`) to the destination cell (`destinationCellId`). Neither source nor destination can be a tape drive.

Caution: Moving cartridges between partitions may confuse some applications requiring you to re-sync the application with the library.

Inputs	Outputs	Roles	Errors
long `sourceCellId`, *long* `destinationCellId`, *boolean* `async`	RequestDto	C1[1],S1,I	IllegalArgumentException if source or destination is a drive. NotFoundException (source cell/cartridge (if volser), destination cell), InvalidMoveException (source empty, destination full)

[1] C1 users can only move cleaning cartridges to and from a CAP. C2 users can only move cartridges to cells within the same partition, while C3, service, and installation roles can move cartridges to any cell.

moveCartridgeByVolser()

Moves a cartridge by specifying the volume serial number (`volser`) of the cartridge and a destination cell id (`destinationCellId`). Neither source nor destination can be a tape drive.

Caution: Moving cartridges between partitions may confuse some applications requiring you to re-sync the application with the library.

Inputs	Outputs	Roles	Errors
string `volser`, *long* `destinationCellId`, *boolean* `async`	RequestDto	C1[1],S1,I	IllegalArgumentException if source or destination is a drive. NotFoundException (source cell/cartridge (if volser), destination cell), InvalidMoveException (source empty, destination full)

[1] C1 users can only move cleaning cartridges to and from a CAP. C2 users can only move cartridges to cells within the same partition, while C3, service, and installation roles can move cartridges to any cell.

dismountCartridgeByCellId()

Dismounts a tape cartridge from the drive specified by `cellId` and moves the tape to the cell specified by `destinationCellId`. The `cellId` is the cell id for the drive, not the drive id. If `force` is true, the library issues a rewind/unload command to the drive before removing the tape.

Caution: Moving tapes between partitions may confuse some applications requiring you to re-sync the application with the library.

Inputs	Outputs	Roles	Errors
long `sourceCellId`, *long* `destinationCellId`, *boolean* `force`, *boolean* `async`	RequestDto	C2[1],S1,I	IllegalArgumentException if source is not a drive or destination is a drive. NotFoundException (drive, destination cell), InvalidMoveException (source empty, destination full)

[1] C2 users can only dismount cartridges to cells within the same partition, while C3, service, and installation roles can dismount cartridges to any cell.

dismountCartridgeByVolser()

Dismounts a tape cartridge from a tape drive. The `volser` specifies the tape to be dismounted and `destinationCellId` specifies the destination. When drive dismounts a tape, the library returns it to the specified cell. If `force` is true, the library issues a rewind/unload command to the drive before removing the tape.

Inputs	Outputs	Roles	Errors
string `volser`, *long* `destinationCellId`, *boolean* `force`, *boolean* `async`	RequestDto	C2[1],S1,I	IllegalArgumentException if source is not a drive or destination is a drive. NotFoundException (drive, destination cell), InvalidMoveException (source empty, destination full)

[1] C2 users can only dismount the cartridges to cells within the same partition, while C3, service, and installation roles can dismount cartridges to any cell.

getLostCartridges()

Returns a list of cartridge information for the lost cartridges.

Lost cartridges can occur when the library finds a cartridges in a robot hand during library startup and the library cannot determine the proper location for the cartridge. The library will examine jobs that were in progress when the library went down to attempt to find the source cell for the cartridge. It it cannot find the source cell, the library will leave the cartridge in system a cell and notify all connected applications using the lostCartridges() outbound SCI method.

Inputs	Outputs	Roles	Errors
None	List of CartridgeDto	C1,S1,I	None

moveLostCartridge()

Moves a lost cartridge by specifying the ID of the cartridge and a destination cell. The destination cannot be a tape drive. The `lostCartridgeId` is from the CartridgeDto returned by getLostCartridges().

Inputs	Outputs	Roles	Errors
long `lostCartridgeId`, *long* `destinationCellId` *boolean* `async`	RequestDto	C1,S1,I	NotFoundException (cartridge, cell)

Partitioning Methods

The initial configuration of a library complex has a single default physical partition that contains all drive bays, all drives, all active storage cells, and all cartridges. You can create additional partitions if partitioning is enabled for the library complex. The output of getLibraryComplex() contains a list of `partitionIds`. Most partition methods take a `partitionId` as an input.

- changePartitionState()
- createPartition()
- deletePartition()
- getPartition()
- getPartitions()
- getPartitionCaps()
- getPartitionCartridges()
- getPartitionCell()
- getPartitionCells()
- getPartitionDrives()
- getPartitionDriveTrays()
- moveCellsToPartitionByCellIdList()
- moveDriveBaysToPartition()
- updatePartition()
- setPartitionCapPool()
- clearPartitionCapPool()

changePartitionState()

Sets the partition ONLINE or OFFLINE. If offline, the partition will reject any host commands. If online, the partition will accept and process host commands.

The library automatically takes a partition offline if you modify the partition and then brings the partition back online when the modification completes. This can result in many state changes. To limit the number of state changes, you can use changePartitionState() before performing a series of modifications to a partition. If the state change cannot happen immediately, a RequestDto returns so that you can track the state of the request.

Inputs	Outputs	Roles	Errors
long `partitionId`, *ControlState* `state`	RequestDto	C3,S2,I	NotFoundException (partition)

createPartition()

Creates a partition. The partition can be a SCSI medium changer device, which means you can access it as a SCSI medium changer device over the FC interfaces. The resulting partition will not contain any cells or drives.

To use this method, you enable partitioning on the library complex.

Inputs	Outputs	Roles	Errors
string `partitionName`, *boolean* `scsiAllowed`, *boolean* `driveSpoofing`, *boolean* `libraryAutoCleaningEnabled`, *LabelWindowing* `labelWindowSetting`, *int* `capPoolId` *FastLoadType* `fastLoadType`	PartitionDto	C3,S2,I	HwafNotFoundException

deletePartition()

Deletes a partition if there are no cartridges and no drive slots in the partition.

Inputs	Outputs	Roles	Errors
long `partitionId`	None	C3,S2,I	NotFoundException (partition)

getPartition()

Returns information for a specific partition.

Inputs	Outputs	Roles	Errors
long `partitionId`	PartitionDto	All	Authentication Exception, Yapi Exception

getPartitions()

Returns a list of all partitions.

Inputs	Outputs	Roles	Errors
None	List of PartitionDto	All	Authentication Exception, Yapi Exception

getPartitionCaps()

Returns a list of CAPs available to the partition. These are the CAPs in the CAP pool assigned to the partition.

Inputs	Outputs	Roles	Errors
long `partitionId`	List of CapDto	All	NotFoundException (partition)

getPartitionCartridges()

Returns a list of cartridges for a partition. getPartition() includes a count of the number of cartridges in the partition.

Inputs	Outputs	Roles	Errors
long `partitionId`, *int* `first`, *int* `count`	List of CartridgeDto	All	Authentication Exception, Yapi Exception, NotFoundException (partition)

getPartitionCell()

Returns a single cell and its contents.

Inputs	Outputs	Roles	Errors
long `partitionId`, *long* `cellId`	CellDto	All	Authentication Exception, Yapi Exception

getPartitionCells()

Returns a list of cells and their contents for a partition. getPartition() returns PartitionDto, which includes a count of the number of cells in the partition. The optional *CellTypeSelector* parameter determines the type of cells returned. If cellType is omitted, the method returns all cells.

Inputs	Outputs	Roles	Errors
long `partitionId`, *CellTypeSelector* `cellType`, *int* `first`, *int* `count`	List of CellDto	All	Authentication Exception, Yapi Exception, NotFoundException (partition)

getPartitionDrives()

Returns a list of drives present in the specified partition.

Inputs	Outputs	Roles	Errors
long `partitionId`, *int* `first`, *int* `count`	List of DriveDto	All	NotFoundException (partition)

getPartitionDriveTrays()

Returns a list of drive trays present in the specified partition.

Inputs	Outputs	Roles	Errors
long `partitionId`, *int* `first`, *int* `count`	List of DriveTrayDto	All	Authentication Exception, Yapi Exception, NotFoundException (partition)

moveCellsToPartitionByCellIdList()

Moves cells and cartridges from one partition to another partition. One variant of this method takes a list of cells in the source partition and moves them to the destination partition. If you move a storage cell containing a cartridge, the cartridge also moves to the destination partition. If you move a cartridge, the cell that contains it will also move to the destination partition.

Inputs	Outputs	Roles	Errors
long `sourcePartitionId`, *long* `destinationPartitionId` Set of *long* `cellIds`	None	C3,S2,I	NotFoundException (source partition, destination partition — only if source is valid, cell/cartridge — if volsers used)

moveDriveBaysToPartition()

Moves drive slots from one partition to another partition. If you move a drive slot containing a drive tray, the drive also moves to the destination partition. The drive must be empty before you move it to a new partition. Only one partition can own a drive slot. The drive slot ids are the `cellIds` for cells of type "Drive".

Inputs	Outputs	Roles	Errors
long `sourcePartitionId`, *long* `destinationPartitionId` Set of *long* `cellIds`	None	C3,S2,I	NotFoundException (source partition, destination partition — only if source is valid, cell)

updatePartition()

Updates settings for the partition.

Inputs	Outputs	Roles	Errors
long `partitionId` *string* `partitionName`, *boolean* `scsiAllowed`, *boolean* `driveSpoofing`, *boolean* `libraryAutoCleaningEnabled`, *LabelWindowing* `labelWindowSetting`, *int* `capPoolId` *FastLoadType* `fastLoadType`	PartitionDto	C3,S2,I	None

setPartitionCapPool()

Sets the CAP pool for a partition.

Inputs	Outputs	Roles	Errors	Status
long `partitionId`, *long* `capPoolId`	None	C2,S1,I	None	Not implemented. Not in WSDL

clearPartitionCapPool()

Unassociates all CAP pools from the partition. When complete, the partition has no access to any CAPs.

Inputs	Outputs	Roles	Errors
long `partitionId`	None	C2, S1, I	ValidationException, NotFoundException, CapErrorException

SCSI Host Methods

- createScsiHost()
- getScsiHosts()

createScsiHost()

Creates a SCSI host. This is useful when adding a host to the library before adding the actual host to the SAN.

Inputs	Outputs	Roles	Errors
long `wwnn`, *long* `wwpn`, *string* `name`	ScsiHostDto	C3, S2, I	Authentication Exception, Yapi Exception

getScsiHosts()

Returns a list of known ScsiHostDto, which include hosts automatically detected by the library or hosts added explicitly by a user.

Inputs	Outputs	Roles	Errors
None	List of ScsiHostDto	All	Authentication Exception, Yapi Exception

Audit Methods

An audit verifies cartridge locations and updates the library database. During a physical audit the robot to scans the barcode labels cartridges. The library automatically performs an audit if you open and then close a door, after a restart if you opened a door while the library was powered off, or if you closed a CAP. A user can also initiate an audit.

- runAuditOnLibraryComplex()
- runAuditOnRange()

runAuditOnLibraryComplex()

Initiates a physical audit of the entire library. The robot scans the barcode on all cartridges, and then updates the database. This method always runs asynchronously. Use getRequest() for status of the audit and waitForRequest() to wait for completion or update.

Inputs	Outputs	Roles	Errors
None	RequestDto	C1,S1,I	None

runAuditOnRange()

Initiates a physical audit on a range of cells. The robot scans the barcode on the cartridges in the range, and then updates the database. This method always runs

asynchronously. Use getRequest()for the status of the audit and waitForRequest() to wait for completion or update.

The inputs are a upper and lower limits on the library, rail, column, side and row values which define the start and end of the range to be audited, and a set of CellTypeSelector values. The set of CellTypeSelector values filters the result to only the specified type of cells.

Inputs	Outputs	Roles	Errors
int `startLibraryNumber`, *int* `startRailNumber`, *int* `startColumnNumber`, *int* `startSideNumber`, *int* `startRowNumber`, *int* `endLibaryNumber`, *int* `endRailNumber`, *int* `endColumnNumber`, *int* `endSideNumber`, *int* `endRowNumber`, Set of *CellTypeSelector* `cellTypes`	RequestDto	C2,S1,I	NotFoundException (cell - first cell address pair to fail)

Media Validation Methods

During media validation, the library mounts a specified cartridge into a drive from the media validation pool (MV pool). The drive reads all or part of the tape and evaluates its quality. You can only use drives in the MV pool for media validation.

- validateCartridgeByCellId()
- validateCartridgeByVolser()

validateCartridgeByCellId()

Initiates media validation on the specified tape cartridge. Select a scan type using the ScanType — EDGE (drive scans the edges of a cartridge for readability), FULL (drive scans the full cartridge for readability), and DIV (T10000C drive scans a cartridge for DIV checksum integrity). This method always runs asynchronously.

Inputs	Outputs	Roles	Errors
long `driveId`, *long* `cellId`, *ScanType* `scanType`	RequestDto	C1,S1,I	NotFoundException (drive, cell/cartridge)

validateCartridgeByVolser()

Initiates media validation on the specified tape cartridge. Select a scan type using the ValidationScanType — EDGE (drive scans the edges of a cartridge for readability), FULL (drive scans the full cartridge for readability), and DIV (T10000C drive scans a cartridge for DIV checksum integrity). This method always runs asynchronously.

Inputs	Outputs	Roles	Errors
long `driveId`, *string* `volser`, *ScanType* `scanType`	RequestDto	C1,S1,I	NotFoundException (drive, cell/cartridge (when using volser))

Diagnostic Testing Methods

- getDiagnosticTests()
- runDiagnosticTest()

getDiagnosticTests()

Returns a list of diagnostic test definitions that a user can execute using runDiagnosticTest(). Refer to the *SL4000 Library Guide* for a description of available diagnostic tests.

Inputs	Outputs	Roles	Errors
None	List of DiagnosticTestDto	C2,S1,I	None

runDiagnosticTest()

Starts a diagnostic test based on the `testName` (which you can get from getDiagnosticTests()). You supply the test parameters using two sting lists. One list is the names of the parameters and the second list contains the values.

The async parameter determines when the method returns. If `async=false`, the method does not return until the test completes. If `async=true`, the method returns once the request is submitted. You can then use getRequest()to determine when the test completes.

Inputs	Outputs	Roles	Errors
string `testName`, List of *string* `paramNames`, List of *string* `paramValues`, *boolean* `async`	RequestDto	C2,S1,I[1]	NotFoundException (diagnostic test) InvalidTestParameters

[1] Only S2 and I users can run robot related diagnostic tests.

Hardware Feature Activation (HWAF) Methods

- installHwaf()
- removeHwaf()
- removeAllHwafs()
- getHwafs()
- getActivtedFeatures()
- getHwafActivtyLog()

installHwaf()

Installs a hardware activation file (HWAF). HWAFs activate optional library features such as dual TCP/IP feature and active capacity.

Inputs	Outputs	Roles	Errors
String of HAWF contents in BASE64 format	HwafDto	C2, S1, I	

removeHwaf()

Removes a hardware activation file (HWAF) and its corresponding feature.

Inputs	Outputs	Roles	Errors
long `id`	None	C2, S1, I	InvalidConfigException (malformed hwaf)

removeAllHwafs()

Removes all installed hardware activation files (HWAFs).

Inputs	Outputs	Roles	Errors
None	None	C3, S1, I	

getHwafs()

Returns a list of installed hardware activation files (HWAFs).

Inputs	Outputs	Roles	Errors
None	List of HwafDto	All	NotFoundException (hwaf)

getActivtedFeatures()

Returns a list the features enabled by hardware activation files (HWAFs).

Inputs	Outputs	Roles	Errors
None	List of ActivatedFeatureDto	All	

getHwafActivtyLog()

Displays a log of hardware activation file (HWAF) activity, listing the date and type of every hardware activation file installed or removed.

Inputs	Outputs	Roles	Errors
None	List of HwafActionDto	C2, S1, I	

Network Configuration Methods

- getCustomerNetworkSettings()
- getOkmNetworkSettings()
- getServiceNetworkSettings()

- libraryTraceRoute()
- pingNetworkAddress()
- resetNetworkSettings()
- setPingResponse()

getCustomerNetworkSettings()

Returns all configuration settings of the customer network interfaces for the library complex.

Inputs	Outputs	Roles	Errors
long `libraryId`	NetworkSettingsDto	All	None

getOkmNetworkSettings()

Returns all configuration settings of the OKM network interfaces for the library complex.

Inputs	Outputs	Roles	Errors
long `libraryId`	NetworkSettingsDto	All	None

getServiceNetworkSettings()

Returns all configuration settings of the Service network interfaces for the library complex.

Inputs	Outputs	Roles	Errors
long `libraryId`	NetworkSettingsDto	All	None

libraryTraceRoute()

Traces the route to a specified network address or hostname from the library.

Inputs	Outputs	Roles	Errors
string `destinationIPAddress` (IP address or Hostname)	TraceRouteResultsDto	C2,S1,I	TraceRouteResultsDto

pingNetworkAddress()

Performs a ping from the library to a specified network address or hostname.

Inputs	Outputs	Roles	Errors
string `destinationIpAddress` (IP address or Hostname)	List of *string* `pingResults`	C2,S1,I	None

resetNetworkSettings()

Resets all the network settings to factory defaults.

Inputs	Outputs	Roles	Errors
None	None	C2,S2,I	None

setPingResponse()

Enables or disables responding to ping protocol on the library interfaces. You can optionally specify a specific network to disable/enable.

Inputs	Outputs	Roles	Errors
boolean `enable`	None	C2,S2,I	None

Library Firmware Upgrade Methods

The getLibraryComplex() and getLibrary() outputs include information about the current and available versions of library firmware. All libraries have two versions of code: active and inactive.

- uploadLibraryFirmware()
- activateLibraryFirmware()

See Also

- getFirmwareUpgradeActivity() - shows drive firmware activity.

uploadLibraryFirmware()

Uploads and validates the firmware upgrade. The uploaded firmware replaces the inactive version. The library will block installation of a software version if it is incompatible with the hardware.

Inputs	Outputs	Roles	Errors
DataHandler	LibraryFirmwareDto	C2,S1,I	IO Exception

activateLibraryFirmware()

Activates a firmware upgrade and causes the library to restart. The `force` parameter skips checking the version compatibility, which wipes out all configuration data on the library.

Inputs	Outputs	Roles	Errors
boolean `force`	RequestDto	C2,S1,I	None

User Role Methods

- createUser()
- deleteUser()
- changeUserRole()
- changeUserPassword()

createUser()

Creates a new user.

Inputs	Outputs	Roles	Errors
string `username`, *string* `password` *string* `description`	None	C3, S2, I	None

deleteUser()

Deletes a user.

Inputs	Outputs	Roles	Errors
string `username`	None	C3, S1, I	None

changeUserRole()

Changes the role of a user.

Inputs	Outputs	Roles	Errors
string `username`, *LibraryRole* `role`	None	C3, S2, I	None

changeUserPassword()

Changes the password of a user.

Inputs	Outputs	Roles	Errors
string `username`, *string* `oldPassword` *string* `newPassword`	None	C3, S1, I	None

Fault and Library Log Methods

- clearFault()
- createSupportBundle()
- createTestEvent()
- deleteSupportBundle()
- downloadSupportBundle()
- getFault()
- getFaults()
- getLogEntries()
- getLoggerLogEntries()
- getLoggingLevel()
- getLoggingLevels()
- getSupportBundle()
- getSupportBundles()
- getSystemReport()
- getSystemReports()

- resetLoggingLevel()
- setLoggingLevel()

clearFault()

Sets the fault status to "cleared". This does not delete the fault, but acknowledges you have seen it.

Inputs	Outputs	Roles	Errors
long `id`	FaultDto	S1,I	NotFoundException (fault)

createSupportBundle()

Creates a support bundle manually (which includes a snapshot of the library log). If `async=true`, the bundle generates asynchronously and you will have to query for the status of the support bundle using the `requestId`. If `async=false`, the method returns when it completes.

Inputs	Outputs	Roles	Errors
boolean `async`	SupportBundleDto	C2,S1,I	None

createTestEvent()

Generates a "test event". You can use this to verify the event sends properly to notification destinations. See also test().

Inputs	Outputs	Roles	Errors
None	RequestDto	C2,S1,I	NotFoundException (fault)

deleteSupportBundle()

Deletes a support bundle from the library complex.

Inputs	Outputs	Roles	Errors
long `supportBundleId`	None	C2,S1,I	NotFoundException (support bundle)

downloadSupportBundle()

Returns the contents of a support bundle. The client application must create a DataHandler to receive the data and store locally.

Inputs	Outputs	Roles	Errors
long `supportBundleId`	DataHandler	C2,S1,I	NotFoundException (support bundle)

getFault()

Returns a specific Device Event object.

Inputs	Outputs	Roles	Errors
long `id`	FaultDto	C2,S1,I	IllegalArgumentException

getFaults()

Returns a list of device event objects where event type is FAULT, limited by `count`. The library sorts faults chronologically, from newest to oldest. You can optional use `firstTimestamp` to specify the starting timestamp of the faults retrieved. If you specify `firstTimestamp`, this method returns faults before this time. To retrieve a long list of faults, use multiple calls with the time from the last fault for the newest parameter on the next call.

Inputs	Outputs	Roles	Errors
int `count`, *date* `firstTimeStamp`	List of FaultDto	C2,S1,I	None

getLogEntries()

Returns a list of log entries, limited by `count`. You can optional use `firstLine` to specify the starting line number the log entries retrieved. To retrieve a long list of entries, use multiple calls with the next line number from the last log entry for the `firstLine` parameter on the next call.

Use the `filter` string to select only matching entries from the log. Initially, filtering is limited to specifying a single value for a specific parameter. The method only returns entries containing that value.

Inputs	Outputs	Roles	Errors
sting `filter` (optional), *int* `firstLine`, *int* `count`	List of logs as string	All	None

getLoggerLogEntries()

Returns a list of logger log entries, limited by `count`. The library sorts log entries chronologically, from newest to oldest. You can optional use `firstLine` to specify the starting timestamp of the entries retrieved. To retrieve a long list of entries, use multiple calls with the time from the last fault for the newest parameter on the next call.

Use the `filter` string to select only matching entries from the log. Initially, filtering is limited to specifying a single value for a specific parameter. The method only returns entries containing that value.

Inputs	Outputs	Roles	Errors
int `firstLine`, *int* `count` *sting* `filter` (optional),	List of logs as string	All	None

getLoggingLevel()

Returns the current log level settings for a specific SL4000 logger.

Inputs	Outputs	Roles	Errors
string `loggerName`	*LogLevel* `loggingLevel`	All	NotFoundException (logger)

getLoggingLevels()

Returns the current log level settings for all SL4000 loggers.

Inputs	Outputs	Roles	Errors
None	List of LoggingLevelDto	All	None

getSupportBundle()

Returns a Support Bundle object with information about a specific support bundle (does not return the actual support bundle). Use downloadSupportBundle() to retrieve the actual support bundle.

Inputs	Outputs	Roles	Errors
long `supportBundleId`	SupportBundleDto	C2,S1,I	None

getSupportBundles()

Returns the support bundles present in the library complex (does not return the actual support bundle). Use downloadSupportBundle() to retrieve the actual support bundle.

Inputs	Outputs	Roles	Errors
None	List of SupportBundleDto	C2,S1,I	None

getSystemReport()

Returns a specific system report object.

Inputs	Outputs	Roles	Errors
long `id`	SystemReportDto	C2,S1,I	IllegalArgumentException

getSystemReports()

Returns a list of system report objects, limited by `count`. The library sorts system reports chronologically, from newest to oldest. You can optional use `firstTimestamp` to specify the starting timestamp of the reports retrieved. To retrieve a long list of reports, use multiple calls with the time from the last system report for the newest parameter on the next call.

Inputs	Outputs	Roles	Errors
int `count`, *date* `firstTimeStamp`	List of SystemReportDto	C2,S1,I	None

resetLoggingLevel()

Resets all logging levels to the defaults.

Inputs	Outputs	Roles	Errors
None	None	C2,S1,I	None

setLoggingLevel()

Changes which log messages the library retains in the logs.

Inputs	Outputs	Roles	Errors
string `loggerName`, *LogLevel* `loggingLevel`	None	C2,S1,I	NotFoundException (logger)

Notification Configuration Methods

- createEmailDestination()
- createSciDestination()
- deleteDestination()
- getDestinations()
- getEmailDestination()
- getEmailDestinations()
- getSciDestination()
- getSciDestinations()
- getSnmpDestination()
- updateEmailDestination()
- updateSciDestination()

createEmailDestination()

Creates a new email destination.

Inputs	Outputs	Roles	Errors
List of *EventCategory* `eventCategories`, *string* `emailAddress`, *string* `localeId`	EmailDestinationDto	C2,S2,I	InvalidConfigException

createSciDestination()

Creates a new outbound SCI destination. See also Outbound Methods. The `destinationPath` should start with a slash "/". An empty string for `destinationPath` is not permitted. Protocol is "http" or "https". Username and password are allowed, but optional, with https protocol. If the library cannot connect to a SCI destination it retains un-sent events up to `retentionTimeLimit`. Later, if the library can communication with the destination, it will send all unsent events in the order they occurred. If the library cannot communicate with the detestation within `retentionTimeLimit`, the library discards all unsent and new events until communication is restored.

Inputs	Outputs	Roles	Errors
List of *EventCategory* `eventCategories`, *string* `destinationProtocol`, *string* `destinationIpAddress`, *string* `destinationPort`, *string* `destinationPath`, *string* `username`, *string* `password`, *int* `retentionTimeLimit`	SciDestinationDto	C2,S2,I	InvalidConfigException

deleteDestination()

Deletes the notification destination.

Inputs	Outputs	Roles	Errors
long `destinationId`	None	C2,S2,I	NotFoundException (destination)

getDestinations()

Returns a list of notification destination's currently configured in the library.

Inputs	Outputs	Roles	Errors
None	List of DestinationDto	All	None

getEmailDestination()

Returns the email destination as specified by the `destinationId`.

Inputs	Outputs	Roles	Errors
long `id`	EmailDestinationDto	All	None

getEmailDestinations()

Returns a list of defined email destinations.

Inputs	Outputs	Roles	Errors
None	List of EmailDestinationDto	All	None

getSciDestination()

Returns a SCI destinations. SCI destinations must implement the outbound SCI specification. The library will invoke the outbound SCI methods when the specified list of events occur.

Inputs	Outputs	Roles	Errors
long `id`	SciDestinationDto	All	None

getSciDestinations()

Returns a list of SCI destinations. SCI destinations must implement the outbound SCI specification. The library will invoke the outbound SCI methods when the specified list of events occur.

Inputs	Outputs	Roles	Errors
None	List of SciDestinationDto	All	None

getSnmpDestination()

Returns the snmp destination as specified by the `id`.

Inputs	Outputs	Roles	Errors
long `id`	SnmpDestinationDto	All	None

updateEmailDestination()

Modifies an existing email destination.

Inputs	Outputs	Roles	Errors
long `destinationId` List of *EventCategory* `eventCategories`, *DestinationType* `type`, *string* `emailAddress`, *string* `locale`	EmailDestinationDto	C2,S2,I	NotFoundException (email destination)

updateSciDestination()

Modifies an existing outbound SCI destination. The `path` should start with a slash "/". An empty string for `path` is not permitted.

Inputs	Outputs	Roles	Errors
long `destinationId`, List of *EventCategory* `eventCategories`, *string* `protocol`, *string* `host`, *string* `port`, *sting* `path`, *string* `userName`, *string* `password`, *int* `retentionTimeLimit`	SciDestinationDto	C2,S2,I	NotFoundException (osci destination)

2

Outbound Methods

The library calls outbound SCI methods when specific events occur. To receive these calls, you must configure a connection using createSciDestination(). Outbound SCI configuration has a protocol option that allows http or https. If the protocol is https, the library includes a WS-Security username and password token in the SOAP header to the outbound SCI destination. If the protocol is http, the library does not send a username and password token as it would be visible in clear text.

The library will make calls to the destination until you delete or disable the connection. Outbound methods are grouped by EventCategory. You can configure a destination to receive SCI calls for one, several, or all categories.

If communication with the destination is disrupted, the library will queue the calls for the `retentionTimeLimit` specified for the destination. The time period is limited because of available space. This is intended to deal with temporary network outages, not as a means to have the library retain history for later delivery. If the library cannot connect to a SCI destination it retains un-sent events up to `retentionTimeLimit`. Later, if the library can communication with the destination, it will send all unsent events in the order they occurred. If the library cannot communicate with the detestation within `retentionTimeLimit`, the library discards all unsent and new events until communication is restored.

Outbound methods only specify input parameters, which are the data objects sent from the library to the called SCI service. The library does not expect a response from any of the methods except for ping(), which must return a string indicating success.

- auditComplete()
- capacityChanged()
- capClosed()
- capOpened()
- capOwnershipOverridden()
- capReadyToOpen()
- deviceControlStateChange()
- deviceFailed()
- deviceInstalled()
- deviceRemoved()
- doorClosed()
- doorOpened()
- driveCleaningNeeded()

- faultDetected()
- libraryComplexStateChange()
- libraryStateChange()
- lostCartridges()
- moveData()
- intermediateData()
- mediaValidationDrivePoolModified()
- partitionChanged()
- ping()
- railStateChange()
- test()

auditComplete()

Sends a notification when a physical audit (scan of barcodes) completes.

- **Category:** LIBRARY
- **Inputs:** AuditEventDataDto

capacityChanged()

Sends a notification when the licensed capacity of the library complex changes.

- **Category:** LIBRARY
- **Inputs:** LicensedCapacityChangeEventDataDto

capClosed()

Sends a notification when a CAP closes and the audit completes.

- **Category:** CAP
- **Inputs:** CapMoveEventDataDto

capOpened()

Sends a notification when a CAP opens.

- **Category:** CAP
- **Inputs:** CapMoveEventDataDto

capOwnershipOverridden()

Sends a notification when the ownership of the CAP has been forcibly removed from a partition (usually by using forceFreeCap()).

- **Category:** CAP
- **Inputs:** CapOwnerOverriddenEventDataDto

capReadyToOpen()

Sends a notification when you need to open a CAP. The library sends this call when ejectExpiredCleaningCartridges, ejectCartridges, or continueEject processes require the operator to open the CAP.

- **Category:** CAP
- **Inputs:** CapReadyToOpenEventDataDto

deviceControlStateChange()

Sends data about a device when its control state changes. This is usually for a device coming online or going offline, though some device types may have other states.

- **Category:** DEVICE
- **Inputs:** DeviceEventDataDto

deviceFailed()

Sends data when a device fails. Typically, the library also sends an ASR for the failure.

- **Category:** DEVICE
- **Inputs:** DeviceEventDataDto

deviceInstalled()

Sends data when a devices is installed into the library. Only sent when a device is installed while the library is operating.

- **Category:** DEVICE
- **Inputs:** DeviceEventDataDto

deviceRemoved()

Sends data when you remove a device from the library. Typically, the library only makes this call for a FRU because you must power off the library to remove most other devices.

- **Category:** DEVICE
- **Inputs:** DeviceEventDataDto

doorClosed()

Sends a notification when a door closes and the audit completes.

- **Category:** DOOR
- **Inputs:** DoorEventDataDto

doorOpened()

Sends a notification when a door opens.

- **Category:** DOOR

- **Inputs:** DoorEventDataDto

driveCleaningNeeded()

Sends a notification when a drive reports that it needs cleaning by an application. The library does not send this call if library auto-cleaning is enabled.

- **Category:** CLEANING_REQUIRED
- **Inputs:** DriveCleanNeededEventdataDto

faultDetected()

Sends data about an alert condition. Alerts are conditions in the library that require intervention. Generally, the library also sends ASRs or SNMP traps. Over temp events might trigger an alert. Not all alerts are device failures, but all device failures are alerts.

- **Category:** DEVICE
- **Inputs:** FaultEventDataDto

libraryComplexStateChange()

Sends data about the library complex when the library complex's functional state (such as normal or fault) or control state (online, offline) changes.

- **Category:** LIBRARY
- **Inputs:** LibraryComplexEventDataDto

libraryStateChange()

Sends data about a library when the library's functional state (such as normal or fault) or control state (online, offline) changes.

- **Category:** LIBRARY
- **Inputs:** LibraryEventDataDto

lostCartridges()

Sends a notification whenever the library identifies a "lost" cartridge. Lost cartridges can occur when the library finds a cartridges, but the library cannot determine the proper location for the cartridge.

- **Category:** LIBRARY
- **Inputs:** LostCartridgesEventDataDto

moveData()

Sends a notification for any cartridge move.

- **Category:** CARTRIDGE_MOVEMENT
- **Inputs:** CartridgeMoveEventDataDto

intermediateData()

Sends a notification one minute after media validation starts and then every 10 minutes. This call can also be made available for normal (non media validation) mounts.

- **Category:** MEDIA_VALIDATION
- **Inputs:** IntermediateMountDriveEventDataDto

mediaValidationDrivePoolModified()

Sends a notification when the media validation drive pool changes. The call returns one element for each drive in the media validation pool.

- **Category:** MEDIA_VALIDATION
- **Inputs:** List of MediaValidationDrivePoolModifiedEventDataDto

partitionChanged()

Sends a notification when a partition changes. Generally, any partitioning changes involve at least two partitions, because the commands that change partitions move cells and drives from one partition to another.

- **Category:** PARTITION
- **Inputs:** List of PartitionEventDataDto for affected partitions

ping()

The library calls ping() when an outbound SCI destination is created to verify connectivity. The library passes in a string containing the library's hostname. The ping() method should return a string indicating success that will be appended into the library logs.

- **Category:** None
- **Inputs:** String

railStateChange()

Sends data about a rail when the rail's functional state changes (such as normal or fault).

- **Category:** LIBRARY
- **Inputs:** RailEventDataDto

test()

Sends a test event to verify connectivity after a call to createTestEvent().

- **Category:** TEST
- **Inputs:** TestEventDataDto

3

SCI Objects

Primitive Types

- *int* - a 32-bit signed, two's complement integer value.
- *long* - a 64-bit signed, two's complement integer value.
- *float* - a 32-bit floating point number.
- *double* - a 64-bit floating point number.
- *string* - a text string of unicode characters.
- *boolean* - true or false.

- *date* - a representation of a date and time. The SOAP implementation is an xsd:dateTime. The dateTime is specified in the following form "`YYYY-MM-DDThh:mm:ssZ`" where:
  - `YYYY` indicates the year
  - `MM` indicates the month
  - `DD` indicates the day
  - `T` indicates the start of the required time section
  - `hh` indicates the hour
  - `mm` indicates the minute
  - `ss` indicates the second
  - (optional) adding a `Z` behind the time indicates UTC timezone. All times on the library are in UTC by default.

  A sample dateTime would be "`2016-10-31T15:30:10Z`". Details are language-specific once the Web Services Description Language (WSDL) has been transformed into a specific language.

Lists and Sets

Some data transfer object (DTO) attributes are lists. These are documented as "list of <something>" where <something> is a primitive type or DTO type. For example, if the WSDL is translated into Java, these attributes become java List objects. Other languages will result in a similar, but language specific, translation.

Some input values allow a set of values. Sets are similar to lists, but are unordered.

DataHandler

SCI uses MTOM (Message Transmission Optimization Mechanism) when uploading or downloading files. For methods that upload or download content, the client must provide a "DataHandler" that performs the client side actions.

Downloading means transferring a file, such as a log file, from the library to the machine where the SCI client is running. The DataHandler receives the data from the library and must store it locally. Methods that download data will return a DataHandler object. The caller must then get an input stream from the returned data handler. The caller then loops, reading data from the input stream constructed from the Data Handler and writing the read data to the desired location.

Uploading means transferring a file to the library, such as for a firmware upgrade. The DataHandler reads the file from a location that is accessible on the machine where the SCI client is running and sends the contents. Methods that upload data require that the caller construct a DataHandler and pass this into the upload call. Typically this data handler would be associated with the file that is to be read. The library will then use the DataHandler to read the contents of the file.

The details of the DataHandler are language specific. Java provides an implementation for a DataHandler. The Java DataHandler can be constructed with a DataSource. A DataSource can be used to read and write files, which is the appropriate usage of MTOM for these SCI methods. A DataSource requires an InputStream or OutputStream. A client must open the input or output file as a FileInputStream or FileOutputStream. The InputStream or OutputStream is then used to create a

DataSource which is in turn used to create the DataHandler that is an input to the methods that transfer files.

Subclass of an Object

Some object classes are subclasses of other object classes. This means all elements in the base object are included. For instance, if TigerDto is a subclass of the base WildAnimalDto, then TigerDto will contain all the attributes from WildAnimalDto, plus new attributes unique to TigerDto.

Data Transfer Objects (DTOs)

Data Transfer Objects (DTOs) are objects returned by SCI methods and sent by OSCI methods. These translate into language-specific classes when the WSDL files are processed. Data Transfer Objects, unlike true object oriented objects, contain only data elements. They do not provide any behavior methods.

Requests, Jobs, and Resources Objects

- RequestDto
- JobDto
- JobParameter
- ResourceDto
- ResourceUsageDto

RequestDto

RequestDtos record library requests which most commonly come from the GUI, SCI, and SCSI interfaces. However, the library may also create requests internally, in particular when detecting a hardware problem.

- *long* `requestId` - unique identifier for the request.
- *date* `createDateTime` - date/time stamp when the request was created.
- *string* `parameters` - a string representation of the request inputs.
- *RequestSource* `source` - the interface that created the request.
- *RequestStatus* `status` - current status of the request.
- *JobDto* `jobs` - a list of zero or more JobDtos that were created to perform the request. Not all requests result in jobs.
- *RequestErrorDto* `error` - an object that contains more detailed information about the outcome of the request for failed requests.
- *RequestOutputMessageDto* `outputMessages` - a list of raw messages produced by the request. You can retrieve a localized, string version of these messages using getLocalizedRequestMessages().

RequestErrorDto

Included in a RequestDto. RequestErrorDto provides details about errors encountered while executing the request.

Only consider the request error object final if the state of the request is "Failed". During execution, a request may encounter recoverable errors that will be reflected in the request error if the state of the request is "Active". The request error will not apply if the state of the request is "Submitted", "Complete", or "Cancelled".

- *long* `requestErrorId` - unique id for the error.
- *RequestErrorType* `errorType` - type of the most recent error encountered.
- *LibraryControllerError* `errorCode` - a more detailed error code.

RequestOutputMessageDto

Included in a RequestDto. RequestOutputMessageDto contains raw data for each message produced as the library processes the request. You can retrieve a human consumable version of the messages with getLocalizedRequestMessages(). However, a program may find the raw messages useful.

- *long* `requestOutputMessageId` - a unique internal identifier for each request output message.
- *date* `createDateTime` - date/time stamp when the message was created.
- *string* `messageKey` - identifies the message. Used to look up the localized message format strings.
- *string* `outputParameters` list - an array of zero or more parameters associated with the message. To create the localized version, parameter values are substituted into the message format string.

JobDto

Requests may result in the creation of jobs. Jobs are individual tasks inside the library. Jobs typically either interact with devices or create child jobs.

- *long* `jobId` - unique id for the job.
- *date* `createDateTime` - date/time stamp when the job was created.
- *date* `startDateTime` - date/time stamp when the job was started executing.
- *date* `completedDateTime` - date/time stamp when the job was completed.
- *JobStateType* `jobState` - an enumeration showing the current state of the job.
- *JobType* `jobType` - an enumeration showing the type of the job.
- *JobParameter* `jobParameter` - a list of key/value pairs that are the parameters used by the job.
- *long* `parentJobId` - job id of the parent job, if any.
- *JobDto* `childJobs` - a list of child jobs of this job, if any.
- *long* `requestId` - unique id of the request that created (directly or indirectly) this job.
- *boolean* `markedForCancellation` - true if a request has been made to cancel the job and the cancellation is in progress.
- *ResourceDto* `resources` - a list of resources needed or used by the job. Once a job completes this shows the specific devices, cells, or rail segments used by the job.

JobParameter

Most jobs have parameters that control what the job does. The JobParameter DTO contains one parameter. These are name/value pairs.

- *string* `key` - name of the parameter.
- *string* `value` - value of the parameter.

ResourceDto

As jobs execute, they use resources in the library. This can include devices and segments of a rail or cells. In the JobDto, the resources attribute is a list of ResourceDtos. The actual objects will be one of three specific object types:

- CellResourceDto
- DeviceResourceDto
- RailSegmentResourceUsageDto

In object-oriented terms, these are subclasses of ResourceDto. The list of resources will never contain a ResourceDto object. It will only contain objects of the three subtypes.

- *ResourceType* `type` - the type of resource.
- *long* `jobId` - job ID for the job using this resource.
- *ResourceState* `state` - current state of this resource.
- *ResourceName* `name` - name for the resource in the context of its job.
- *date* `allocatedDateTime` - date/time stamp when the resource was allocated.
- *date* `freedDateTime` - date/time stamp when the resource was freed.

CellResourceDto

Represents the use of a cell by a job.

- *long* `cellID` - unique ID of the cell used by the job.

DeviceResourceDto

Represents a device used by a job.

- *long* `deviceId` - unique ID of the device used by the job.
- *DeviceType* `deviceType` - enumeration for the type of device.

RailSegmentResourceDto

Represents a portion of the rail used by a job.

- *long* `railNumber` - unique ID for the rail.
- *int* `startMil` - mil (one thousandth of an inch) position of the start of the rail segment, as measured from the left-most position a robot can occupy on the rail.
- *int* `endMil` - mil (on thousandth of an inch) position of the end of the rail segment, as measured from the left-most position a robot can occupy on the rail.

ResourceUsageDto

Records the resources used after a job completes. You can use this data to analyze the usage of resources over time. The library stores this data for seven days after the job that used the resource completes.

- *long* `jobId` - id of the job that used this resource
- *JobType* `jobType` - type of the job that used this resource
- *long* `parentJobId` - job id of the parent job of the job that used this resource
- *long* `requestId` - request id of the request that resulted in the job that used this resource.
- *ResourceName* `name` - An internal name for this resource, as defined by the job.
- *ResourceType* `type` - Type of this resource (CELL, DEVICE, or RAIL_SEGMENT).
- *ResourceState* `state` - Current state of this resource. Can be NEEDED (the job needs this resource and has not yet been able to allocate it), ALLOCATED (the job has allocated this resource), or COMPLETE (the job is finished with this resource).
- *date* `allocatedDateTime` - Date and time when the resource was allocated.
- *date* `freedDateTime` - Date and time when the job was finished with the resource and it was freed.

RailSegmentResourceUsageDto

Represents a portion of the rail used by a job. RailSegmentResourceDto provides information about the resource used for an active job. RailSegmentResourceUsageDto is used for historical data after jobs have completed.

- *long* `railNumber` - unique ID for the rail.
- *int* `startMil` - mil (one thousandth of an inch) position of the start of the rail segment, as measured from the left-most position a robot can occupy on the rail.
- *int* `endMil` - mil (on thousandth of an inch) position of the end of the rail segment, as measured from the left-most position a robot can occupy on the rail.
- *int* `sourceMilPosition` - mil (one thousandth of an inch) position of the starting point of the robot when performing the move.
- *int* `targetMilPosition` - mil (one thousandth of an inch) position of the ending point of the robot when performing the move.
- *int* `safeColumn` - column number where the robot can safely swing the wrist.

Library Objects

- LibraryComplexDto
- LibraryDto
- ModuleDto
- RailDto
- CellDto
- SlotDto
- DoorStateDto
- PartitionDto

- ScsiHostDto
- ScsiLunDto
- TimeSettingsDto

LibraryComplexDto

- *boolean* `ready` - TRUE means the library has completed the initial audit that may occur during startup.
- *string* `name` - string name for the library complex.
- *string* `currentLibraryTime` - current library time as a string.
- *TimeSettingsDto* `timeSettings` - library complex time settings.
- *ControlState* `controlState` - current user-controlled control state for the library complex.
- *LibraryComplexStateType* `operationalState` - Current operational (functional) state of the library complex.
- *DoorStateDto* `doorState` - Current state of the library doors.
- *LibraryComplexCountsDto* `counts` - Counts of various objects within the library.
- *boolean* `suppressHasBeenOpened` - TRUE indicates the library will not audit after a restart if the door has been opened. This value affects only the next power cycle for the library. After powering up, it will be set back to FALSE. When selecting TRUE, you must guarantee the contents of the cells and drives will not be modified while the library is powered off, even if doors are opened. This setting applies only to the next library startup.
- *boolean* `checkLibraryConfiguration` - TRUE indicates the library should scan the module id blocks on the next power up. This value affects only the next power cycle for the library. After powering up, it will be set back to FALSE.
- *boolean* `redundantFcPortsEnabled` - TRUE indicates that the second FC port on each controller card is enabled. This feature requires the Redundant FC HWAF or the Redundant Ethernet HWAF.
- *boolean* `redundantEthernetPortsEnabled` - TRUE indicates the second customer network Ethernet port on each controller card is enabled. This feature requires the Redundant Ethernet HWAF Redundant FC HWAF.
- *boolean* `redundantControllersEnabled` - TRUE indicates the second LOC controller card is enabled.
- *boolean* `redundantRoboticsEnabled` - TRUE indicates the library has dual robots.
- *boolean* `partitioningEnabled` - TRUE indicates the library can have more than one partition and multiple CAP pools.
- *int* `licensedCapacity` - the total number of cells allowed by all installed Capacity HWAFs. This is adjusted by adding or removing capacity HWAFs.
- *long* `auditRequestId` - If an audit is in progress, this is the request id for that audit. Use getRequest() for details about the audit.
- *LabelWindowing* `labelWindowing` - the presentation of cartridge label volsers to the client. This setting is obsolete and replaced by the partition setting of the same name.

LibraryComplexCountsDto

- *int* `libraryCount` - number of libraries in the complex. For SL4000, this is always 1.
- *int* `partitionCount` - number of partitions defined in the library complex.
- *int* `deviceCount` - number of devices in the library complex. This is a total count that includes all nested devices that are inside modules and other devices.
- *int* `driveCount` - number of drives in the library complex. Same as the number of drive trays in the library complex.
- *int* `cellCount` - number of cells of all types (storage, drives, CAPs, and so on).
- *int* `storageCellCount` - number of storage cells (application-accessible cells that can hold cartridges).
- *int* `systemCellCount` - number of system cells. These are cells reserved for internal use by the library and are not usable by external applications.
- *int* `capCellCount` - number of CAP cells.
- *int* `driveBayCount` - number of drive bays in the library complex, whether they contain a drive tray or not.
- *int* `cartridgeCount` - number of cartridges in the library complex.
- *int* `failedDeviceCount` - number of devices that are in a failed state.
- *int* `failedRailCount` - number of rails that are in a failed state.
- *int* `degradedRailCount` - number of rails that are in a degraded state.
- *int* `robotCount` - number of robots in the library.
- *int* `slotCount` - number of slots in the library complex. This is a total count that includes all nested slots that are inside modules and other devices.
- *int* `moduleCount` - number of modules in the library complex.
- *int* `diagnosticsCartridgeCount` - number of diagnostic cartridges in the library complex that are in system cells.
- *int* `cleaningCartridgeCount` - number of cleaning cartridges in the library complex that are in system cells. Cleaning cartridges in storage cells are managed by applications and are not included in this count.

LibraryDto

- *long* `libraryId` - the unique database identifier for the library.
- *string* `name` - string name for the library.
- *int* `number` - always 1 for SL4000 libraries.
- *LibraryIdentityDto* `identity` - identity information for the library.
- *CardCageIdentityDto* `cardCageIdentity` - identity information for the card cage.
- *LibraryFirmwareDto* `activeFirmware` - information about the version of firmware that is currently running on the library.
- *LibraryFirmwareDto* `oldFirmware` - information about the version of firmware that was previously running on the library. The version is still present on the library and the library can be rolled back to this version.

- *LibraryFirmwareDto* `newFirmware` - information about the version of firmware that has been installed but is not currently running on the library. This version can be activated and the library will begin running this new version.
- *ControlState* `controlState` - current user-controlled control state for the library.
- *LibraryStateType* `operationalState` - current operational (functional) state of the library.
- *long* `wwnSeed` - WWN seed value for the library. Used to assign WWNs to FC ports on the library and tape drives.
- *long* `originalWwnSeed` - WWN seed value for libraries that have been upgraded from SL3000 to SL4000 libraries. For upgraded libraries, this is used for the base and first drive module to the left of the base. For upgraded libraries, any other drive module will have WWNs assigned using the wwnSeed value.
- *long* `fcNodeName` - WWNN for the library.
- *RailDto* `rails` - list of information about the rails.
- *LibraryCountsDto* `counts` - count information for objects in the library.
- *long* `AuditRequestId` - if an audit is in progress for the library, this is the request ID for that audit.
- *LibraryProductionState* `libraryProductionState` - production state for the library. Normally, "Production" is when the library has been installed at a customer site.
- *RedStackInfoDto* `redStackInfo` - the version information for the Oracle Red Stack components used in the software.

LibraryIdentityDto

- *string* `marketingPartNumber` - marketing part number for the library
- *string* `systemRevision` - revision level for the library.
- *string* `systemSerialNumber` - serial number of the library.
- *string* `systemModelName` - description of the library on the bill of materials.
- *string* `manufacturingPartNumber` - manufacturing part number for the library.
- *string* `qPartNumber` - part number used for some service functions.
- *string* `vendorId` - vendor name for this part.

CardCageIdentityDto

- *string* `cardCagePartNumber` - manufacturing part number for the base card cage.
- *string* `cardCageRevision` - revision level for the base card cage.
- *string* `cardCageSerialNumber` - serial number of the base card cage.
- *string* `cardCageModelName` - description of the base card cage on the bill of materials.

LibraryCountsDto

- *int* `deviceCount` - number of devices in the library complex. This is a total count that includes all nested devices that are inside modules and other devices.
- *int* `driveCount` - number of drives in the library complex. Same as the number of drive trays in the library complex.

- *int* `cellCount` - number of cells of all types.
- *int* `storageCellCount` - number of storage cells (application-accessible cells that can hold cartridges).
- *int* `systemCellCount` - number of system cells. These are cells reserved for internal use by the library and are not usable by external applications.
- *int* `capCellCount` - number of CAP cells.
- *int* `driveBayCount` - number of drive bays in the library complex, whether they contain a drive tray or not.
- *int* `cartridgeCount` - number of cartridges in the library complex.
- *int* `failedDeviceCount` - number of devices that are in a failed state.
- *int* `failedRailCount` - number of rails that are in a failed state.
- *int* `degradedRailCount` - number of rails that are in a degraded state.
- *int* `robotCount` - number of robots in the library.
- *int* `slotCount` - number of slots in the library complex. This is a total count that includes all nested slots that are inside modules and other devices.

RedStackInfoDto

Version information about the Oracle software components used internally by the library.

- *string* `webLogicAppServerVersion`
- *string* `oracleClusterwareVersion`
- *string* `oracleAdfVersion`
- *string* `databaseServerVersion`
- *string* `databaseDriverVersion`
- *string* `libraryOsVersion`
- *string* `javaRuntimeVersion`

ModuleDto

The SL4000 consists of modules that are attached together to form a library. At minimum, a library has a base module. It can also have Access Modules, Drive Modules, Cartridge Modules, and Parking Modules.

- *long* `moduleId` - unique ID for the module.
- *int* `moduleNumber` - the module number used to identify the module that contains this slot. The base module has a module number of 0. Modules to the left of the base (when viewed from the front of the library) have negative values, starting at -1 for the module immediately to the left of the base. Modules to the right of the base have positive numbers, starting with 1 for the module immediately to the right of the base.
- *ModuleCountsDto* `moduleCounts` - counts of various objects in the module.
- *ModuleType* `type` - module type for this module.

ModuleCountsDto

- *long* `ModuleId` - unique ID for the module

- *int* `deviceCount` - total number of devices in the module.
- *int* `driveCount` - number of drives in the module.
- *int* `cellCount` - total number of cells in the module.
- *int* `storageCellCount` - number of storage cells.
- *int* `capCellCount` - number of CAP cells.
- *int* `driveBayCount` - number of drive bays.
- *int* `cartridgeCount` - number of cartridges.
- *int* `failedDeviceCount` - number of devices in a failed operational state.

RailDto

- *long* `railId` - unique ID for the rail.
- *int* `railNumber` - number of the rail within the library.
- *RailCountsDto* `railCounts` - counts of various objects associated with the rail.
- *long* `AuditRequestId` - if an audit is in progress for the library, this is the request ID for that audit.
- *int* `sweptLengthMils` - The actual value measured by the robots for the `usableLengthMils`. This value will vary slightly from `usableLengthMils` due to manufacturing tolerances.
- *int* `usableLengthMils` - Nominal length of the rail in mils (thousandths of an inch). This is the distance from the left-most position a robot can occupy on the rail to the right-most position a robot can occupy.

RailCountsDto

- *int* `deviceCount` - number of devices associated with the rail. This is a total count that includes all nested devices.
- *int* `driveCount` - number of drives in the library complex. Same as the number of drive trays in the library complex.
- *int* `cellCount` - number of cells of all types.
- *int* `storageCellCount` - number of storage cells (client-accessible cells that can hold cartridges).
- *int* `capCellCount` - number of CAP cells.
- *int* `driveBayCount` - number of drive bays accessible on this rail, whether they contain a drive tray or not.
- *int* `cartridgeCount` - number of cartridges accessible on this rail.
- *int* `failedDeviceCount` - number of devices that are in a failed state.

CellDto

A cell object represents a location inside the library that can hold a cartridge.

- *long* `cellId` - unique ID for each cell.
- *CellType* `type` - the type of cell.
- *boolean* `allocated` - true if the cell is currently allocated to a job.

- *CellState* `state` - physical state of the cell. PRESENT means the cell is physically present and a robot can put or get a cartridge. NOT_PRESENT means it is not physically present, such as a CAP cell when the CAP is open. UNKNOWN means the state cannot be determined at this time.
- *CellContentsState* `contentsState` - contents state of the cartridge in this cell, if any.
- *CartridgeDto* `cartridge` - information about the cartridge in this cell, if any.
- *CellAddressDto* `address` - address for this cell.
- *long* `deviceId` - the unique ID of the device that contains this cell.
- *long* `partitionId` - the unique ID of the partition that owns this cell.
- *long* `libraryId` - the unique ID of the library that contains this cell.
- *int* `scsiElementId` - for cells that belong to partitions with SCSI enabled, this is the SCSI element ID assigned to the cell. These are unique within a partition, but duplicates will appear across multiple partitions.
- *string* `addressAsString` - text string of the form L,R,C,S,R (library, rail, column, side, row).

CellAddressDto

A cell address object is a physical address of a cell in the library. For more information on addressing, see the *SL4000 Library Guide*.

- *int* `libraryNumber` - library number for this cell. This is always 1 for SL4000 libraries.
- *int* `columnNumber` - column number for this cell.
- *int* `railNumber` - rail number for this cell. This is always 1 for SL4000 libraries.
- *int* `sideNumber` - side number for this cell. For SL4000 libraries, 1 = back wall and 2 = front wall.
- *int* `rowNumber` - row number for this cell.

SlotDto

A slot is a physical location inside a module or device that can hold a device. A slot might or might not actually contain a device.

- *long* `slotId` - ID for the slot. Unique for all slots in a library complex.
- *int* `slotNumber` - number for this slot. Unique for all slots within a module that can hold the same type of device.
- *long* `moduleId` - unique database identifier for the module that contains this slot.
- *int* `moduleNumber` - the number used to identify the module that contains this slot. The base module has a module number of 0. Modules to the left of the base (when viewed from the front of the library) have negative values, starting at -1 for the module immediately to the left of the base. Modules to the right of the base have positive numbers, starting with 1 for the module immediately to the right of the base.
- *long* `libraryId` - this is null for SL4000 libraries.
- *int* `libraryNumber` - this is null for SL4000 libraries.
- *long* `parentDeviceId` - for slots inside devices, this is the unique Device ID for the containing device. For slots located directly within a module, this is null.

- *ComponentLocationState* `controlState` - a state that controls whether or not a device inserted into this slot is automatically brought online.
- *DeviceType* `slotDeviceType` - type of device this slot can contain.
- *DeviceDto* `containedDevice` - DeviceDto for the device in the slot, if any.
- *string* `locationName` - name of the slot location.

DoorStateDto

The DoorState object provide the states of the doors.

- *boolean* `demDoorOpen` - TRUE indicates the DEM door is open. Only one value is available, even if there are multiple DEMs in the library.
- *boolean* `leftAemDoorOpen` - TRUE indicates the left AEM door is open.
- *boolean* `rightAemDoorOpen` - TRUE indicates the right AEM door is open.
- *boolean* `baseDoorOpen` - TRUE indicates the Base door is open.
- *boolean* `leftAemSafetyDoorOpen` - TRUE indicates left AEM safety door is open.
- *boolean* `rightAemSafetyDoorOpen` - TRUE indicates right AEM safety door is open
- *DoorState* `leftAemSafetyDoor` - indicates the left AEM safety door status.
- *DoorState* `rightAemSafetyDoor` - indicates the right AEM safety door status.

PartitionDto

- *long* `partitionId` - unique ID for this partition.
- *string* `name` - user-assigned name for the partition.
- *string* `group` - name of the user group to which the partition belongs.
- *PartitionStateType* `operationalState` - provides the current operational state of the partition. INOPERATIVE indicates a failure has left this partition unusable.
- *ControlState* `controlState` - current user-defined control state of the partition. When OFFLINE, a partition will reject all host commands that cause robotic actions.
- *long* `capPoolId` - unique ID of the CAP pool assigned to this partition. Use with getCapPool() to get details of the CAP pool. Use getPartitionCaps() to get the list of CAPs for the partition.
- *FastLoadType* `fastload` - controls when SCSI Move Medium commands return:
  - IMMEDIATE: the command returns as soon as it has been validated. Not currently supported.
  - FAST: the command returns as soon as the cartridge has been loaded into the drive. The drive may not be ready at that time.
  - NORMAL: the command waits for the drive to thread the tape and become ready before returning.
- *boolean* `driveSerialNumberSpoofing` - controls whether tape drive serial numbers are "spoofed". TRUE returns the first 10 digits of the drive tray serial number. FALSE returns the drive manufacturer's serial number. This only applies to LTO drives. T10000 drives do not support spoofing.

- *LabelWindowing* `labelWindowing` - controls which characters of the barcode are presented to clients.
- *boolean* `autoCleaning` - TRUE means the library will sense when drives in the partition need cleaning, and will automatically mount, run, and then dismount a cleaning cartridge (from the system cells) before the next mount. FALSE means the host software must manage drive cleaning.
- *boolean* `scsiAllowed` - TRUE indicates the partition can be accessed as a SCSI Medium changer device. If TRUE, SCI commands that move cartridges will be rejected. If FALSE the partition is not exposed as a SCSI medium changer LUN and only SCI commands can be used to move cartridges.
- *PartitionCountsDto* `partitionCounts` - counts of the various objects in the partition.
- *string* `scsiPartitionCode` - A two character code assigned to each partition. This is combined with the library serial number so that each partition has a unique serial number in the response to INQUIRY commands.
- *boolean* `PartitionMediaValidationDrivePool` - TRUE indicates that the partition is the media validation partition.

PartitionCountsDto

- *int* `cellCount` - total number of cells of all types.
- *int* `storageCellCount` - total number of storage cells (client-accessible cells that can hold cartridges).
- *int* `capCellCount` - total number of CAP cells.
- *int* `cartridgeCount` - total number of cartridges in the partition.
- *int* `driveCount` - total number of drives in the partition.
- *int* `driveBayCount` - total number of drive bays.

ScsiHostDto

- *long* `wwnn` - wwnn for the SCSI host.
- *long* `wwpn` - wwpn for the SCSI host.
- *long* `abortFlag` - the value of the abortFlag for the SCSI host.
- *string* `name` - a text name for the host, supplied by the user.
- *ScsiHostState* `scsiHostState` - the state of the SCSI host.
- *long* `lunIds` - list of IDs for the corresponding logical units.

ScsiLunDto

- *long* `scsiHostID` - ID of the SCSI host that participates in this nexus.
- *long* `partitionId` - ID of the partition that participates in this nexus.
- *int* `lunNumber` - Logical Unit Number for this nexus.
- *string* `source` - AUTOMATIC means this SCSI Logical Unit was added automatically by the library. USER means it was explicitly added by a user through the GUI.

- *boolean* `enabled` - if TRUE, commands are allowed using this SCSI logical unit. If FALSE, commands are blocked. It is set to FALSE by configuring access using the GUI.

TimeSettingsDto

Time settings define how the clocks are set on the library complex. You can explicitly set the date and time or use an NTP server (external to the library).

- *string* `ntpServers` - NTP servers, in string format.
- *boolean* `ntpEnabled` - true if NTP has been configured.
- *boolean* `forceEnabled` - reserved for future use. This parameter is not currently used.
- *date* `currentTime` - current time on the library when this DTO was created.

Tape Cartridge Objects

CartridgeDto

A cartridge object represents a tape cartridge. Because the library cannot definitively identify cartridges, a cartridge object does not have unique IDs common in other objects.

- *string* `volser` - if the contents state is readable, this is the volser derived from the rawLabel.
- *string* `rawLabel` - full data read from the barcode label.
- *CartridgeTypeDto* `detailedType` - contains information about the type of cartridge (make, model, size, and so on).
- *boolean* `diagnostic` - TRUE indicates this is a diagnostic cartridge. Volsers for diagnostic cartridges start with "DG".
- *long* `cellId` - unique ID of the cell that contains this cartridge.
- *long* `partitionId` - unique ID of the partition that contains this cartridge.
- *long* `lostCartridgeId` - a unique ID valid only for lost cartridges. This attribute will be populated only in the output from the getLostCartridges() method.

CartridgeTypeDto

- *long* `cartridgeTypeId` - unique ID for this cartridge type.
- *string* `family` - family for the cartridge: T10000 or LTO.
- *string* `generation` - generation for the cartridge. For LTO, this starts with 1 for the first generation. For T10000, this starts with A. The value of 0 is used for some cleaning cartridges.
- *int* `capacity` - native (uncompressed) capacity of the cartridge in GB.
- *string* `domainCode` - the domain code.
- *string* `typeCode` - the type code.

- *boolean* `cleaning` - TRUE indicates the cartridge is a cleaning cartridge.
- *boolean* `worm` - TRUE indicates the cartridge is a WORM (write once, read many) cartridge.
- *string* `descriptiveName` - name for the media type.
- *MediumType* `mediumType` - the type of cartridge: DATA or CLEANING.
- *int* `recommendedMaximumUsage` - manufacturer's recommendation for maximum number of uses.
- *int* `warningThreshold` - user-specified warning threshold. The cartridge is considered expired after this number of uses.

CleaningCartridgeDto

- *int* `cleanCount` - number of times the cartridge has been used for cleaning

Network Objects

- FcPortDto
- IpAddressDto
- NetworkAddressDto
- NetworkInterfaceSettingsDto
- NetworkPerformanceMeasurementDto
- NetworkSettingsDto
- TraceRouteResultsDto

FcPortDto

This object holds settings associated with FC ports on drives. These settings are only meaningful on an arbitrated loop configuration.

- *boolean* `hardAssignedPhysicalAddress` - TRUE if the port is set to use a specific physical address (an ArbitratedLoopAddress must be specified). If FALSE, you should set the SoftAssignedPhysicalAddress attribute.
- *int* `arbitratedLoopAddress` - a specific loop ID value from 0 to 125.
- *string* `softAssignedPhysicalAddress` - Used when hardAssignedPhysicalAddress is FALSE, the drive will seek a physical address. HI means addresses are searched in descending order. LO means the addresses are searched in ascending order.
- *FcPortState* `fcPortState` - state of the FC port.

IpAddressDto

- *string* `defaultGateway` - IP address of the first router connected to this interface.
- *string* `ipAddress` - IP Address value.
- *IpAddressType* `ipAddressType` - IPv4 or IPv6.
- *string* `netmask` - netmask for IPv4 IP addresses.
- *int* `prefixLength` - prefix length for IP v6 addresses.

NetworkAddressDto

Contains internal network information for devices inside the library. These addresses are internal to the library and are not visible through any of the external interfaces.

- *string* `name` - This field is not used.
- *string* `host` - IP address of the device.
- *string* `port` - port number used to communicate to the device.
- List of *string* `webServiceUri` - URI for the device's web service interface.

NetworkInterfaceSettingsDto

- *IpAddressDto* `ipv4Address` - assigned IPv4 addresses.
- List of *IpAddressDto* `ipv6Addresses` - list of IPv6 addresses.

NetworkPerformanceMeasurementDto

Network switch ports represent a port on a switch chip on a card connecting to another device. The names used for these measurement points identify the device on the other end of the connection. This is a subclass of MeasurementDto and adds the following attributes:

- *int* `portSpeed` - network speed in Mbps
- *int* `txOctets` - total number of good bytes of data transmitted by a port
- *int* `txDroppedPackets` - number of packet dropped by a port (incremented only if not counted by either the TxLateCollision or the TxExcessiveCollision counters)
- *int* `txCollisions` - number of collisions experienced by a port during packet transmissions
- *int* `txPausePackets` - number of pause events on a port
- *int* `rxOctets` - number of bytes of data received by a port (including bad packets).
- *int* `rxDroppedPackets` - number of good packets received by a port that were dropped due to lack of resources (incremented only if the receive error was not counted by the RxAlignmentErrors or the RxFCSErrors counters).
- *int* `rxPausePackets` - number of pause frames received by a port.
- *int* `rxAlignmentErrors` - number of packets received by a port that have a length between 64 and standard max frame size and a bad FCS with a non-integral number of bytes.
- *int* `rxFcsErrors` - The number of packets received by a port that have a length between 64 and standard max frame size and a bad FCS with an integral number of bytes.
- *int* `rxSymbolErrors` - The total number of times a valid length packet was received at a port and at least one invalid data symbol was detected. Counter increments only once per carrier event and does not increment on detection of collision during the carrier event.

NetworkSettingsDto

- *boolean* `ipv6Enabled` - true if IPv6 is enabled.
- *NetworkSettingsType* `type` - the interface to which these settings apply.

- *NetworkInterfaceSettingsDto* `networkInterfaceSettings` - the settings for this interface.

TraceRouteResultsDto

- list of *string* `traceHops` - output of the traceroute command for each router along the route to the target of the traceroute command.

Device Objects

Device objects relate to the hardware components of the library.

- DeviceDto
- LedDto
- PingDeviceResultsDto
- FruIdDto

DeviceDto

Devices are hardware components within the library. These are represented by a hierarchy of classes of objects. "Device" itself is the most generic, and contains information that applies to all devices. Various subclasses are used for device types that have additional data. The more specific classes extend the Device object and add additional attributes for the specific device type. Methods such as "getDevices(Library ID)" return a list of DeviceDto while methods such as "getRobots(Library ID)" will return a list of more specific RobotDtos. The getDevice(deviceId) method returns a DeviceDto while getRobot(deviceId) will return a RobotDto.

- *long* `deviceId` - unique integer ID for each device. Used on many methods to uniquely identify a device.
- *long* `parentDeviceId` - unique id for the parent device. The parent device is the device that physically contains this device.
- *DeviceType* `parentType` - enumeration specifying the type of the parent device.
- *string* `name` - text name for this device.
- *DeviceIdentityDto* `manufacturingCardIdentity` - DeviceIdentityDto data for the bare card. This data is assigned when the card is manufactured.
- *DeviceIdentityDto* `manufacturingFRUIdentity` - FRU level DeviceIdentityDto data for this device. This data is assigned when the card is manufactured into a FRU or other type of higher level assembly. In some cases, FRUs are only a single board and this data will be the same as the `manufacturingCardIdentity`.
- *DeviceIdentityDto* `marketingIdentity` - Marketing level DeviceIdentityDto data for this device. Used by service to identify the correct replacement part. Assigned when the FRU is manufactured. All parts with the same marketing part number are compatible even if the manufacturing part number is different. .
- *DeviceType* `type` - enumeration specifying the type of the device.
- *FruType* `fieldReplaceableUnitType` - the service category for this device.
- *long* `moduleId` - unique identifier for the module the device is in.
- *long* `moduleNumber` - module number of the module containing the device.

- *long* `libraryId` - unique identifier for the library in a library complex that contains the device.
- *long* `slotId` - Identifier for the slot in the module containing the device.
- *long* `slotNumber` - slot number of the slot containing the device.
- *TopLevelDeviceStateType* `topLevelDeviceOperationalState` - summary device state. This field summarizes the `operationalState` field into few higher level states.
- *DeviceStateType* `operationalState` - detailed device state. This field provides detailed, device type-specific operational state information for the device.
- *ControlState* `controlState` - current user-defined control state of the device.
- *boolean* `hotSwap` - TRUE indicates the device can be hot-swapped.
- *IpAddressDto* `ipAddress` - internal IP address for the device.
- *LedDto* `leds` - list of `LedDto` for the LEDs on this device. If this device has no LEDs, the list will be empty.

DeviceIdentityDto

A DeviceIdentityDto contains information that identifies a device. This information is a combination of a serial number (defined when the device is manufactured) and additional information (part number, revision, and description) that is defined when the part is designed.

Many parts contain a FRUID Storage Container chip that holds this information. Some FRUID Storage Containers contain information about multiple devices. The LOD card and the LOID card in a drive tray, for example, contain the identity information for both the individual card and for the drive tray. Some third party components, such as tape drives, encryption cards, and web cameras, also have this identification information. Where possible, the library controller will retrieve this information and include it in this object. Some components do not have this information, including rails, power buses, PDUs, and power supplies.

- *string* `partSerialNumber` - serial number of the individual component.
- *string* `partNumber` - manufacturing part number for individual component.
- *string* `partRevision` - revision level for the individual component.
- *string* `partDescription` - description of the individual component on the bill of materials.
- *long* `deviceId` - unique integer ID for each device. Used on many methods to uniquely identify a device.
- *DeviceType* `deviceType` - the type of device.

LedDto

- *ServiceIndicatorName* `name` - the type of LED.
- *SeviceIndicatorState* `state` - current state of the LED.

PingDeviceResultsDto

- *long* `deviceId` - device ID of the device being pinged (from input to the pingDevice method).

- *DeviceType* `deviceType` - type of device that was pinged.
- *ControlState* `deviceControlState` - current control state of the device that was pinged.
- *DeviceStateType* `deviceOperationalState` - current operational state of the device that as pinged.
- *boolean* `devicePinged` - TRUE if the device was successfully pinged.
- *string* `errorMessage` - error message if the ping was unsuccessful.

FruIdDto

The FruIdDto contains data from the FRUID storage containers present on active cards. These are EEPROMs (Electrically Erasable Programmable Read Only Memory) which are programmed during manufacturing with serial numbers, part numbers and other data that uniquely identifies active boards and assemblies that contain active boards. An active board is a board that has active electronic components as opposed to a passive board which contains only non-active components, usually just connectors. The EEPROM is divided into "segments" which then contain one or more "records". Records contain individual fields. The segments used in the SL4000 are named "SD" and "FL", but these names have no particular meaning.

Each device which has a FRUID chip has five unique identity records. These five records can record unique data:

- *Base Part Identity* - identity data about an individual board. Located in the SD segment.
- *FRU Identity* - identity data about the assembly that contains the board. In some cases, the same as the base part identity data.
- *Configured Item Identity* - Contains "marketing" identity data about a FRU. Parts will the same marketing identity are fully compatible, even if the FRU or Base Part data differs.
- *System Identity Data* - data for the last library the part was inserted into.
- *Product Identity Data* - currently always identical to System Identity Data.

The attributes of FruIdDto are:

- *string* `rawFruIdData` - raw data from the FRUID EEPROM chip on the device, in base64 encoding.
- *SDSegmentDto* `SDSegment` - described below.
- *FLSegmentDto* `FLSegment` - described below.

SDSegmentDto

- *BasePartIdentityDto* `basePartIdentityRecord` - Base Part identity data for the device.
- *wwnRangeDto* `wwnRange` - WWN seed information.
- *string* `crc32` - CRC 32 bit error checking code for the SD segment.

FLSegmentDto

- *FrudIdentityDto* `fruIdentityRecord` - FRU identity data for the device. Contains manufacturing based identity information.

- *ConfiguredIdentityDto* `configuredIdentityRecord` - Marketing identity data for the assembly.
- *SystemIdentityDto* `systemIdentityRecord` - Library identity data for the last library where the part was installed.
- *ProductIdentityDto* `productIdentityRecord` - Library identity data for the last library where the part was installed.
- *string* `checksum` - checksum for the FL segment.

BasePartIdentityDto

- *date* `basePartTimeStamp` - Date and time of last update of this record.
- *string* `basePartDescription` - Card part description.
- *string* `basePartSerialNumber` - Card serial number.
- *string* `initialBasePartNumber` - Card part number.
- *string* `initialBasePartRevision` - Card part revision.
- *string* `specPartNumber` - Vendor part number
- *string* `supplierId` - Vendor ID.

wwnRangeDto

- *string* `wwn` - starting WWN for the library.
- *long* `range` - not used.

FrudIdentityDto

- *date* `fruTimeStamp` - Date and time of last update of this record.
- *string* `fruDescription` - FRU description
- *string* `fruSerialNumber` - FRU serial number.
- *string* `fruPartNumber` - FRU part number.
- *string* `fruRevision` - FRU revision level.

ConfiguredIdentityDto

- *date* `configuredPartTimeStamp` - Date and time of last update of this record.
- *string* `configuredPartDescription` - FRU part description, same as in *FrudIdentityDto*.
- *string* `configuredPartSerialNumber` - FRU serial number, same as in *FrudIdentityDto*.
- *string* `configuredPartNumber` - FRU marketing part number
- *string* `configuredPartRevisionLevel` - FRU revision level, same as data in *FrudIdentityDto*

SystemIdentityDto

- *date* `systemIdentityTimeStamp` - Date and time of last update of this record.
- *string* `systemIdentityModelName` - Library model name string.
- *string* `systemIdentitySerialNumber` - Library serial number.

- *string* `systemIdentityPartNumber` - Library part number.
- *string* `systemIdentityRevisionLevel` - Library revision level.
- *string* `hostId` - not used.
- *string* `macAddress` - not used.

ProductIdentityDto

- *date* `productIdentityTimeStamp` - Date and time of last update of this record.
- *string* `productIdentityModelName` - Library model name string.
- *string* `productIdentitySerialNumber` - Library serial number.
- *string* `productIdentityPartNumber` - Library part number.
- *string* `productIdentityRevisionLevel` - Library revision level.
- *string* `hostId` - not used.
- *string* `macAddress` - not used.

SensorDto

Sensors on devices collect and record measurements periodically. You can view device measurements by using getDeviceTelemetry(). You can view a list of sensors on a device by using getDeviceSensors().

- *long* `sensorId` - unique ID for the sensor.
- *string* `name` - name for the sensor.
- *SensorType* `type` - the sensor type.

TelemetryDto

- *long* `deviceId` - identifier of device associated with measurements
- *SensorDto* `sensor` - sensor that generated the measurements.
- List of *MeasurementDto* `measurements` - telemetry data for energy, hotswap, network, temperature or fan (see subclasses below).

MeasurementDto

- *date* `timeStamp` - date and time value for when the readings were captured.

EnergyMeasurementDto

This is a subclass of MeasurementDto and adds the following attributes:.

- *float* `powerKw` - Instantaneous power consumption at the time the reading was taken in kilowatts.
- *float* `energyKwh` - Energy consumption in kilowatt hours.

HotSwapMeasurementDto

Most cards have hot-swappable controllers with measurable power consumption. For example, the LOS card has a hot-swappable controller for the card and another for the rail. This is a subclass of MeasurementDto and adds the following attributes:

- *float* `inputVoltage` - Input voltage to the hot swap controller
- *float* `inputCurrent` - Input current to the hot swap controller.
- *float* `powerWatts` - An instantaneous reading of power being passed through the hot swap controller.

TemperatureMeasurementDto

This is a subclass of MeasurementDto and adds the following attributes:

- *float* `value` - Current temperature reading in degrees C.

FanMeasurementDto

This is a subclass of MeasurementDto and adds the following attributes:

- *float* `speed` - Current fan speed rpm.
- *FanHealth* `health` - The current health of the fan determined by the fan controller.

CAP Objects

- CapDto
- CapPoolDto
- CapMeasurementDto
- CapStatisticsDto

CapDto

CAP objects represent both rotational CAPs and AEMs. CapDto extends DeviceDto.

- *long* `capId` - unique identifier for the CAP.
- *long* `capPoolId` - unique identifier for the CAP pool which contains the CAP.
- *long* `owningPartitionId` - partition ID for partition that owns the CAP, if any.
- *boolean* `locked` - TRUE indicates the CAP is locked.
- *boolean* `open` - TRUE indicates the CAP is open.
- *string* `capPoolUsageState` - indicates the CAP is UNUSED, DEDICATED, or SHARED.
- *string* `modulePosition` - not used.

CapPoolDto

- *long* `capPoolId` - unique ID for this CAP pool.
- *string* `name` - string name for this CAP pool.
- *CapDto* `caps` - list of CAPs in this CAP pool.
- *PartitionDto* `partitions` - list of partitions that can use the CAPs in this CAP pool.

CapMeasurementDto

- *long* `totalOperations` - running total of open and close operations performed by CAP.

- *long* `retries` - running total of retries.
- *long* `unrecoverableErrors` - running total of unrecoverable errors for the CAP (typically zero or one because an unrecoverable error requires replacement).
- *long* `ipls` - running total of CAP restarts (typically just one at library startup, but this can be higher if you replace the CAP controller card while the library is running)

CapStatisticsDto

- *int* `totalOps` - running total of open and close operations performed by CAP
- *int* `retries` - running total of retries
- *int* `unrecoverableErrors` - running total of unrecoverable errors for the CAP (typically zero or one because an unrecoverable error requires replacement)
- *int* `ipls` - running total of CAP restarts (typically just one at library startup, but this can be higher if you replace the CAP controller card while the library is running)

Drive Objects

- DriveDto
- DriveTrayDto
- DriveOperationDto

DriveDto

Drive objects represent tape drives in the library. Drives have several attributes that can be retrieved with the getDrive() method and updated with the updateDrive() method. DriveDto extends DeviceDto.

- *string* `serialNumberFactoryAssigned` - the serial number assigned to the drive by the manufacturer.
- *string* `serialNumberSpoofed` - if serial number spoofing is enabled for the partition containing this drive, this is the serial number assigned to the drive by the library. An empty string if the drive is in a partition where spoofing is disabled.
- *DriveTypeDto* `detailedType` - drive type information.
- *DriveTrayDto* `driveTray` - drive tray information.
- *boolean* `ready` - TRUE indicates a loaded cartridge is ready.
- *CellDto* `cell` - information about the drive cell.
- *string* `firmwareLevel` - drive firmware level.
- *string* `portAWwn` - string representation of the full WWN for port A.
- *string* `portBWwn` - string representation of the full WWN for port B.
- *FcPortDto* `portAFcSettings` - arbitrated loop settings for port A. Not applicable to fabric configurations.
- *FcPortDto* `portBFcSettings` - arbitrated loop settings for port B. Not applicable to fabric configurations.
- *boolean* `fastload` - TRUE indicates fastload is enabled.

- *int* `tcpPortNumber` - TCP/IP port number used to connect to drive.
- *string* `driveIpAddress` - IP address for the drive.
- *string* `IodIpAddress` - IP address for the drive controller card.
- *string* `driveAlias` - user defined name for the drive.

DriveTypeDto

- *string* `brand` - brand name of the drive: STORAGETEK, HP, or IBM.
- *string* `family` - the drive series: T10000 or LTO.
- *string* `generation` - drive generation. LTO drives use numeric generations starting with 1. StorageTek drives use alphabetic generations starting with A.
- *DriveInterfaceType* `physicalInterfaceType` - the drive interface.
- *boolean* `encryptionCapable` - TRUE indicates the drive can encrypt.
- *int* `typeCode` - an integer value provided by Oracle StorageTek drives. This value encodes the family, generation, encryption capability, and emulation mode of the drive.
- *string* `descriptiveName` - a string value that combines the drive family and generation into a human-readable value.
- *string* `emulation` - Oracle StorageTek drives are capable of emulating IBM drives. A value of "3590" indicates the drive is set to emulate IBM drives. A null value means the drive is not emulating IBM drives.

DriveTrayDto

DriveTrayDto extends DeviceDto.

- *DriveDto* `drive` - DriveDto for the drive in the drive tray.
- *DeviceDto* `drivePowerSupply` - DeviceDto for the tape drive power supply in the drive tray.
- *DeviceDto* `lodCard` - DeviceDto for the LOD card in the drive tray.
- *DeviceDto* `encryptionCard` - DeviceDto for the encryption card, if installed in the drive tray.

DriveOperationDto

- List of *DriveActivityDataDto* `activityList` - A list of DriveActivityDataDto object containing information queried from the drive during a mount, dismount, or media verification operation. There will be one DriveActivityDataDto object for each individual command used to query data from the drive.
- *DriveOperationStatus* `operationStatus` - status of the mount or dismount operation.
- *CommandTiming* `commandTiming` - Timing of when the commands were issued to the drive, MOUNT, DISMOUNT or INTERMEDIATE.

Robot Objects

- RobotDto
- RobotCalibrationDto

- RobotCellDepthDto
- RobotGetStatisticsDto
- RobotMetricsDto
- RobotMetricDataDto
- RobotParametersDto
- RobotPositionHistoryDto
- RobotStatisticsDto
- MotionRangeDto

RobotDto

Robot objects represent the robots in the library. RobotDto extends DeviceDto.

- *int* `trackPosition` - track position of the robot.
- *long* `railNumber` - rail number associated with the robot.
- *RobotHomeEnd* `robotHomeEnd` - home end for the robot.
- *string* `ipAddresses` - Internal IP address of the robot.

RobotCalibrationDto

This object contains the results of a robot calibration for a specific cell array. This data is used by Oracle service and engineering to evaluate the robot's condition

- *boolean* `calEmptyFlag`
- *boolean* `calFullFlag`
- *string* `arrayAddress`
- *RobotMetricDataDto* `emptyBottomMaxMetrics`
- *RobotMetricDataDto* `emptyBottomMinMetrics`
- *RobotMetricDataDto* `emptyBottomMetrics`
- *RobotMetricDataDto* `emptyTopMaxMetrics`
- *RobotMetricDataDto* `emptyTopMinMetrics`
- *RobotMetricDataDto* `emptyTopMetrics`
- *RobotMetricDataDto* `fullBottomMaxMetrics`
- *RobotMetricDataDto* `fullBottomMinMetrics`
- *RobotMetricDataDto* `fullBottomMetrics`
- *RobotMetricDataDto* `fullTopMaxMetrics`
- *RobotMetricDataDto* `fullTopMinMetrics`
- *RobotMetricDataDto* `fullTopMetrics`

RobotCellDepthDto

- *double* `cellDepth` - Depth of cells is in mils (thousandths of an inch).

RobotGetStatisticsDto

Supplies statistical information about a robot.

- *int* `auditTotal` - total number of audits performed.
- *int* `auditFailures` - total number of audit failures.
- *int* `auditRetries` - total number of audit retries.
- *int* `fetchTotal` - total number of fetches performed.
- *int* `fetchFailures` - total number of fetch failures.
- *int* `fetchRetries` - total number of fetch retries.
- *int* `targetTotal` - total number of targets scanned.
- *int* `targetFailures` - total number of target scan failures.
- *int* `targetRetries` - total number of target scan retries.
- *int* `putTotal` - total number of puts performed.
- *int* `putFailures` - total number of put failures.
- *int* `putRetries` - total number of put retries.

RobotMetricsDto

- *string* `mechName` - Name of the robot mechanism that this data applies to. Can be TRACK (upper track motor), STRACK (lower track motor), ZMECH (Z or vertical motor), WRIST, REACH, or GRIP.
- *double* `distance`
- *double* `moveTime`
- *double* `maxPositionError`
- *double* `minPositionError`
- *double* `avgPositionError`
- *double* `maxCurrentCommand`
- *double* `minCurrentCommand`
- *double* `avgCurrentCommand`
- *boolean* `endMode`
- *double* `settlingTime`
- *double* `settlingAvgCurrent`
- *double* `settlingAvgPositionError`
- *double* `stallDistance`
- *double* `stallTime`
- *double* `stallCurrentMax`
- *double* `stallCurrentMin`
- *double* `stallCurrentAvg`
- *double* `stallStartPosErr`

RobotMetricDataDto

- *double* `t` - track position, in mils (thousandths of an inch)
- *double* `z` - Z (vertical) position, in mils (thousandths of an inch)
- *double* `w` - wrist position, in mils (thousandths of an inch)

RobotParametersDto

- *boolean* `retriesEnabled` - TRUE (default) indicates robot retries are enabled. FALSE indicates retries are disabled and the robot will return a fault if any action fails on the first attempt.
- *int* `trackMaxSpeedPercent` - maximum speed for track, as a percentage of maximum possible speed.
- *int* `zMaxSpeedPercent` - maximum speed for Z.
- *int* `wristMaxSpeedPercent` - maximum speed for the wrist mechanism.
- *int* `reachMaxSpeedPercent` - maximum speed for the reach mechanism.
- *int* `gripMaxSpeedPercent` - maximum speed for the grip mechanism.

RobotPositionHistoryDto

This DTO records a single robot move. A series of these DTOs will show the motion of the robot over the time period covered by the series.

- *long* `id` - unique id for this robot position history record.
- *long* `robotId` - device id of the robot performing this move.
- *RobotHomeEnd* `robotHomeEnd` - home end of the robot.
- *int* `trackPosition` - position of the robot after the move.
- *DeviceStateType* `currentState` - state of the robot during the move. This will be an active state such as PUTTING, FETCHING or MOVING.
- *DeviceStateType* `nextState` - state of the robot after the move. This will usually be INACTIVE, but could be FAILED_IMMOVEABLE or FAILED_MOVEABLE if a problem occurred during the move.
- *RobotStatusCode* `robotStatusCode` - status code from the robot for the move.
- *RobotHardwareStatusCode* `robotHardwareStatusCode` - a more detailed status code from the robot after the move.
- *boolean* `operationSuccessful` - TRUE if the move was successful.
- *string* `command` - the command being performed.
- *date* `timestamp` - time the move completed.
- *long* `jobId` - job ID for the job performing the move.

RobotStatisticsDto

Supplies statistical information about a robot.

- *int* `auditRetries` - total number of audit retries.
- *int* `auditFailures` - total number of audit failures.
- *int* `fetchTotal` - total number of fetches performed.

- *int* `fetchRetries` - total number of fetch retries.
- *int* `fetchFailures` - total number of fetch failures.
- *int* `putTotal` - total number of puts performed.
- *int* `putRetries` - total number of put retries.
- *int* `putFailures` - total number of put failures.
- *int* `targetTotal` - total number of targets scanned.
- *int* `targetRetries` - total number of target scan retries.
- *int* `targetFailures` - total number of target scan failures.

MotionRangeDto

Contains information on the range of travel for a physical mechanism. Methods that return this object typically return a list, one item for each mechanism for the device being queried. The operating min and max values are the limits of normal robot motion. The operating range is slightly smaller than the physical range as shown by the physical min and max values. The physical min and max values are the physical limit of motion.

- *string* `name` - name for the specific mechanism. Options are TRACK, ZMECH, WRIST, REACH, GRIP, STRACK, and CAP.
- *double* `operatingMax` - integer value in mils.
- *double* `operatingMin` - integer value in mils.
- *double* `physicalMax` - integer value in mils.
- *double* `physicalMin` - integer value in mils.

User Objects

- UserDto
- GroupDto
- RoleDto

UserDto

A user object represents a user ID that can connect to the library through the GUI or SCI interface. User authentication is performed by either the local LDAP server on the library or a customer-defined LDAP server. However, use of an external LDAP server is not allowed. The first time a user logs in to the library, a User will be created in the library controller software for that user. The User entry in the library controller software is used to track the group the user belongs to and user-specific preferences.

- *string* `name` - text userid for the user. This must match the userid in the LDAP server.
- *string* `source` - Text, either "local" or "enterprise". Local means the user is defined in the local (on library) LDAP server. "enterprise" means the user is defined in the enterprise LDAP server. Currently only local user are supported.
- *string* `group` - text name of the group to which the user belongs.
- *string* `libraryRole` - the library role for the user.

- *string* `enterpriseRole` - the enterprise role that maps to the library role (if these roles have been defined). Not currently used.

GroupDto

A group defines a set of users. Groups are intended for use in controlling access to partitions. However, this functionality is not currently implemented. Each partition is owned by only one group. Certain roles have access to all partitions. Other roles, however, have access only to partitions that belong to the same group as the user. When a new user logs into the library, that user will not belong to any group. An administrator must specify the user's group. This can be done before the user logs in or after. However, if the user's role limits access to partitions, that user will not be able to view or modify any partition-specific information.

- *string* `name` - text name for the group.
- *string* `description` - text description for the group.
- List of *UserDto* `users` - a list of user names for users that belong to the group.

RoleDto

The library uses role-based authentication. A role defines the functions a user may perform. The library software defines a list of library roles. This list and the permissions associated with each role cannot be changed. The role names can be used in a user-defined LDAP server. Alternately, you can set up a mapping between enterprise roles and library roles. Enterprise roles are the roles used in an LDAP server. When a new user first logs into the library, the user's list of roles will be retrieved from the LDAP server and compared to the library and enterprise role names. If a match is found, the user in the library will be assigned the matching library role.

- *string* `libraryRole` - text name of the library role.
- *string* `description` - text description for the role.
- *string* `enterpriseRole` - text name of a role defined in the customer's LDAP server. If not supplied, the LibraryRoles should be used in the LDAP server to control the role assigned to a user in the library software. Not currently used.

Hardware Activation Objects

- ActivatedFeatureDto
- HwafDto
- HwafActionDto

ActivatedFeatureDto

- *Feature* `activatedFeature` - the HWAF feature which is active.
- *int* `capacity` - for capacity HWAFs, the number of slots enabled by this HWAF

HwafDto

- *long* `hwafId` - unique ID for this HWAF.
- *Feature* `feature` - the feature controlled by the HWAF.
- *date* `expirationDate` - date that the HWAF expires

- *int* `capacity` - for capacity HWAFs, the number of slots enabled by this HWAF.

HwafActionDto

- *Feature* `feature` - the feature controlled by the HWAF.
- *int* `capacity` - for capacity HWAFs, the number of slots enabled by this HWAF
- *string* `action` - The action taken relating to the HWAF. For example, "ADD" and "DELETE".
- *date* `actionDate` - the time stamp when the action takes place.
- *string* `userId` - the user who performed the action.

Diagnostic Test Objects

A diagnostic test performs a series of library actions to evaluate the status of the library or to demonstrate a feature. The library provides a set of known diagnostics tests. A diagnostic test is a test the library can perform on itself. A user can initiate a diagnostic test, which will create a request. This request can be queried to see the status and results of the test.

The getDiagnosticTests() method can be used to query the library for a list of available tests. The runDiagnosticTest() method can then be used to initiate the tests.

Tests are typically executed in the background. The async parameter on the runDiagnosticTest() method controls when the runDiagnosticTest() call will return. Starting a test returns a RequestDto for the test run. You can retrieve the status of the test run and its final results using the Request ID from the RequestDto returned when you started the test.

DiagnosticTestDto

- *string* `name` - string name for the test.
- *string* `description` - string description of the test.
- *DiagnosticTestParameterDto* `testParameters` - the parameters used to define the test.

DiagnosticTestParameterDto

A DiagnosticTestParameterDto is used to describe the test specific parameter.

- *string* `name` - name for the parameter.
- *string* `description` - description for the test parameter.
- *string* `type` - the type of parameter: BOOLEAN, NUMBER, STRING
- *anyType* `value` - not used.

Notification Objects

- DestinationDto
- AsrDto
- ServiceContactDto

DestinationDto

This is a superclass of ASR Destination, SNMP Destination, Email Destination, and Outbound SCI Destination. The getDestinations() method will return a list of objects of this class.

- *long* `destinationId` - unique ID of the destination.
- *EventCategory* `eventCategories` - list of categories to which this destination is subscribed.
- *DestinationType* `type` - type of destination.

EmailDestinationDto

A subclass of DestinationDto.

- *string* `emailAddress` - alerts will be mailed to this address.
- *string* `locale` - the email will be localized for this locale. Use getSupportedLocales() to determine available values.

SciDestinationDto

A subclass of DestinationDto.

- *string* `host` - IP address for the destination for outbound SCI calls, reachable through the customer network.
- *string* `port` - port for the destination for outbound SCI calls.
- *string* `path` - URL for outbound SCI calls.
- *string* `userName` - user ID used to log into the outbound SCI interface.
- *string* `password` - password used to log into the outbound SCI interface.
- *int* `retentionTimeLimit` - a time limit for retaining notifications if the destination is unreachable. The library will attempt to retain events up to this time limit, and will periodically retry. It will send the queued events once the destination returns.

AsrDestinationDto

A subclass of DestinationDto.

- *NetworkSettingsType* `asrNetworkAdapter` - The network interface that is used for the connection to SDP2.
- *string* `address` - IP address of the SDP2 server.
- *int* `port` - Port number for the connection to SDP2.
- *string* `clientId` - Identity of the library, used by SDP2.
- *boolean* `enabled` - TRUE when the connection to SDP2 is enabled

SnmpDestinationDto

A subclass of DestinationDto. This object contains the parameters for an SNMP destination.

- *string* `host` - hostname or IP address of the destination host.
- *string* `protocolVersion` - SNMP protocol version (VTWOC, VTHREE). For V2, you must provide a community string. For V3, you must specify all other parameters.

- *string* `community` - for V2 only. A password or phrase. Cannot be "community".

> **Caution:** Configuring “public” or “private” as valid community strings is a major security risk. These are commonly used and easily guessed.

- *string* `userName` - for V3 only. Text name for the SNMP user. Limited to upper and lower case letters and !@#$%^*()-+\=\~.
- *string* `authenticationType` - for V3 only. Enumeration for the type of authentication for this user: MD5, SHA, or NONE.
- *string* `authenticationPassphrase` - for V3 only. String passphrase for this user.
- *string* `privacyType` - for V3 only. Enumeration for the type of privacy to be used on notifications (traps) that are sent to the user. This determines how traps sent to this destination are encrypted. DES, AES or NONE.
- *string* `privacyPassphrase` - for V3 only. String passphrase used for privacy.
- *string* `engineId` - for V3 only. String engine ID for this destination. If not specified, the library will use its own engine ID.

AsrDto

- *string* `assignedCaseNumber` - ASR identifier assigned by Oracle support.
- *long* `serviceBundleId` - ID of the service bundle generated when the fault was detected.
- *date* `submitTime` - date/time stamp when the ASR was submitted.
- *FaultDto* `fault` - fault object for the fault that triggered the ASR.

ServiceContactDto

- *string* `contactName` - string name of the contact person.
- *string* `phoneNumber` - string phone number for the contact person.
- *string* `streetAddr` - string street address where the library is installed.
- *string* `city` - string city where the library is installed.
- *string* `state` - string state or other region where the library is installed.
- *string* `country` - country where the library is installed.
- *string* `zipCode` - string postal code for the library.
- *string* `description` - string description for the contact person.

Logging and Fault Objects

The library tracks system reports and faults that occur in the library. Faults represent events requiring service intervention to correct. A fault can be a hardware failure requiring replacement of a part. It can also be a software problem that requires intervention to correct.

System reports are the input to faults, and record significant events that occur in the library. They can be created as the result of an error, but can also occur as the result of

successful library operations. These provide a history of actions in the library that allow the library to perform analysis to determine when a fault occurred.

- LoggingLevelDto
- SupportBundleDto
- SystemReportDto
- FaultDto
- SuspectFruDto

LoggingLevelDto

- *string* `loggerName` - Each log message has an associated logger name. The logger names are for different categories of messages and different devices. Examine the logging level settings in the GUI to see these names.
- *string* `loggerLevel` - The logging level represents the severity of the message. Only log messages at or above the specified logging level are captured in the library logs. Possible values (most severe to least) are SEVERE, WARNING, CONFIG, FINE, and FINER. A value of INHERITED means this logger uses its parent logger's level.

SupportBundleDto

A support bundle is a large file containing a data dump about the library. Support bundles can be created on demand and are also automatically captured when the library detects a fault. They can be downloaded and transferred to Oracle support for problem diagnosis.

- *long* `supportBundleId` - unique ID for the support bundle.
- *SupportBundleOriginator* `originator` - how the support bundle was generated.
- *SupportBundleState* `state` - the state of the support bundle.
- *date* `timeStamp` - Date and time this support bundle was generated.

SystemReportDto

- *long* `systemReportId` - unique ID for the system report.
- *date* `timestamp` - date/time stamp when the system report was created.
- *SystemReportType* `reportType` - type of system report.
- *ErrorCode* `statusCode` - code for the specific fault.
- *HardwareStatusCode* `hardwareStatusCode` - code for the hardware fault.
- *long* `createdRequestId` - unique ID of the request that was being processed when the system report was generated.
- *long* `createdJobId` - ID of the job created to process this system report, if any.
- *long* `sourceRequestId` - ID of the request being processed when this system report was created.
- *long* `sourceJobId` - unit ID of the job that was being processed when the system report was generated, if any.
- *long* `originatingDeviceId` - device ID for the device that created the system report.

- *long* `reportedDeviceId` - device ID for the device that was reported by this system report.
- *SensorDto* `reportedComponentIdentifier` - SensorDto for the reported component.

FaultDto

A FaultDto is created when the library detects a fault. This object includes a list of devices. If the library can identify a specific device that is the source of the fault, that fault will be the only device in the list. If not, multiple devices will appear in the list, ordered with the most likely cause first. "Device added" and "device removed" event types will have only a single device snapshot object.

- *long* `faultId` - unique integer ID for the event.
- *FaultSymptomCodeType* `faultSymptomCode` - a code for the specific fault.
- *date* `timestamp` - date/time stamp when the event was detected.
- List of *SuspectFruDto* `suspectFrus` - list of possible fault causing devices.
- List of *SystemReportDto* `systemReports` - list of SystemReportDto system reports.
- *long* `serviceBundleId` - ID of the service bundle generated when the fault was detected.
- *boolean* `reviewed` - whether someone has reviewed the fault. False when the fault is created. Set to true by clearFault() or within the GUI.
- *EventSeverity* `severity` - severity of this fault. Usually ERROR because most faults require intervention. A value of WARNING is used for faults that do not require immediate intervention.
- *CorrectiveActionsType* `correctiveAction` -indicates actions that need to be taken to eliminate the fault.

SuspectFruDto

- *long* `faultId` - unique ID of the fault for this suspect FRU.
- *DeviceType* `deviceType` - device type for this suspect FRU. The combination of device type, frame number and slot number identify the location of the slot where this suspect FURU is installed.
- *int* `frameNumber` - identifier for the module where this suspect FRU is located.
- *int* `slotNumber` - identifier for the slot where this suspect FRU is located.
- int `priority` - priority in the list of suspect FRUs. A value of 1 is for the most likely device. Higher values are for less likely devices.
- *DeviceDto* `device` - The device DTO for the suspect FRU

Firmware Related Objects

Firmware related objects represent versions of library and drive firmware on the library. The library can have two versions of library firmware. One is active, and the other is updated using the uploadLibraryFirmware() method and can later be activated with the activateLibraryFirmware() method.

The library will hold multiple versions of drive firmware. Drive firmware versions are first uploaded to the library and then applied to specific drives. Drive firmware versions that are uploaded to the library remain on the library until removed.

- LibraryFirmwareDto
- ComponentFirmwareDto
- DriveFirmwareDto
- FirmwareUpgradeEventDto

LibraryFirmwareDto

Provides information about a library firmware version. This is the complete package of all firmware for the entire library. This object does not include the firmware itself, only data about the firmware.

- *string* `version` - firmware version.
- *date* `buildDate` - date and time this firmware version was created.
- *ComponentFirmwareDto* `componentFirmwareList` - code version information for devices in the library.

ComponentFirmwareDto

Provides version information for the code running on devices inside the library.

- *DeviceType* `deviceType` - the type of device.
- *FirmwareType* `firmwareType` - the type of firmware.
- *string* `codeVersion` - the code version for the device.
- *string* `basePartNumber` - base part number for this device. See the BasePartIdentityDto for more information.
- *string* `basePartRevision` - base part revision of the device.
- *boolean* `activeVersion` - firmware version currently running on this device.

DriveFirmwareDto

- *string* `version` - drive firmware version.
- *string* `driveType` - the type of drive to which this firmware applies.
- *date* `buildDate` - date this firmware version was created.
- *date* `uploadDate` - date this firmware version was uploaded.

FirmwareUpgradeEventDto

Each time library or drive firmware is uploaded or activated, a firmware upgrade event is captured. This provides a history of upgrade activity.

- *FirmwareType* `firmwareType` - type of firmware.
- *string* `version` - version string
- *long* `driveId` - ID of drive, for drive firmware actions that are specific to a drive
- *date* `actionDate` - date/time stamp for the action.
- *string* `userName` - name of user who performed the action.

- *string* `result` - final status of the action

Outbound SCI Objects

These objects are specific to the outbound SCI interface.

- EventDataDto
- TestEventDataDto
- IntermediateMountDriveEventDataDto
- DriveActivityDataDto
- FaultEventDataDto
- LibraryComplexEventDataDto
- LibraryEventDataDto
- RailEventDataDto
- LostCartridgesEventDataDto
- DeviceEventDataDto
- CartridgeMoveEventDataDto
- RobotMoveDto
- CapMoveDto
- AuditEventDataDto
- AuditActivityDataDto

EventDataDto

- *long* `eventId` - The unique id for this event.
- *string* `comment` - A text comment, used for debugging.
- *date* `timeStamp` - The date and time when the event occurred.
- *EventCategory* `category` - The category for this event. Categories control which events are sent to which destination.
- *EventSeverity* `severity` - Severity of the event. ERROR indicates human intervention is required to correct the fault. WARNING indicates faults that do not require immediate attention.
- *RequestDto* `request` - The request for this event.
- *EventType* `type` - type of the event, tells the subclass for the specific event.
- *string* `libSerialNumber` - serial number of library that generated the event.

CapMoveDto

- *long* `capId` - device ID of the CAP that has moved.
- *long* `partitionId` - partition ID of the partition that owned the CAP when it was moved.
- *string* `moveDirection` - whether the CAP opened or closed.
- *CellDto* `cells` - list of cells in the CAP and their contents.

CapMoveEventDataDto

Sent when a CAP is opened or closed.

- *CapMoveDto* `capMove` - information about the cap and the operation performed

CapOwnerOverriddenEventDataDto

Extends EventDataDto.

- *CapDto* `cap` - the CAP who has its ownership overridden.

CapReadyToOpenEventDataDto

Extends EventDataDto.

- *long* `capId` - ID of the CAP that is ready to be opened.

CartridgeMoveEventDataDto

Extends EventDataDto.

- *LibraryDto* `library` - The Library which created and sent the event.
- *CellDto* `sourceCell` - The source cell.
- *CellDto* `destinationCell` - The destination cell.
- *DriveDto* `sourceDrive` - The source drive.
- *DriveDto* `destinationDrive` - The destination drive.
- *CartridgeDto* `cartridge` - The contents of the source cell before the move.
- List of *RobotMoveDto* `robotMoves` - A list of the individual robotic moves performed to complete the cartridge movement.
- List of *DriveOperationDto* `srcDriveOperations` - A list of the source drive operations.
- List of *DriveOperationDto* `dstDriveOperations` - A list of the destination drive operations.
- *date* `mountStartTime` - time the mount operation was started.
- *date* `mountEndTime` - time the mount operation completed.
- *date* `dismountStartTime` - time the dismount operation started.
- *date* `dismountEndTime` - time the dismount operation completed.

DeviceEventDataDto

Extends EventDataDto.

- *long* `supportBundleId` - The unique id of the support bundle that was created for this event.
- *ErrorCode* `errorCode` -
- *string* `wrappedServiceUserId` - The encrypted userid for the service role user created to deal with this event.
- *string* `wrappedServicePassword` - The encrypted password for the service role user created to deal with this event.

- List of *SystemReportDto* `systemReports` - The system reports that contributed to this event.
- *DeviceDto* `device` - The device that generated the event.

DoorEventDataDto

Extends EventDataDto.

- *DoorStateDto* `doorState` - state information for the door that was opened or closed.

DriveActivityDataDto

Contains information retrieved from the drive during mounts and dismounts.

- *DriveProtocol* `protocol` - type of protocol used to communicate with the drive.
- *string* `protocolVersion` - drive protocol version string.
- *base64Binary* `command` - command issued to the drive to retrieve data.
- *base64Binary* `results` - results returned by the drive from the command.
- *boolean* `success` - TRUE indicates the command was successful in retrieving data from the drive. If FALSE, the "results" field will be null.
- *DriveActivityStatusCode* `statusCode` - status of the drive operation that was being performed when this data was retrieved.
- *CommandTiming* `commandTiming` - drive operation that was being performed when this data was retrieved.

DriveCleanNeededEventdataDto

Extends EventDataDto.

- *DriveDto* `drive` - the drive that needs cleaning

FaultEventDataDto

Extends EventDataDto.

- *FaultDto* `faultReport` -

IntermediateMountDriveEventDataDto

Extends EventDataDto.

- *DriveDto* `drive` - the drive performing the media verification.
- List *DriveActivityDataDto* `driveActivities` - List of DriveActivityDataDtos retrieved from the drive at the completion of the mount, but before acknowledging the move to the client that initiated the mount.
- *int* `validationPercent` - percentage complete of the media validation operation.

LibraryComplexEventDataDto

Extends EventDataDto.

- *LibraryComplexDto* `libraryComplex` - The library complex that experienced the event.

LicensedCapacityChangeEventDataDto

Extends LibraryComplexEventDataDto. Contains no additional attributes.

LibraryEventDataDto

Extends EventDataDto.

- *LibraryDto* `library` - The library that experienced the event.

AuditEventDataDto

Sent when the initial library audit after startup is completed. Extends LibraryEventDataDto.

- List *AuditActivityDataDto* `auditActivities` - contains the results of the audit, cell by cell.
- List *RobotMoveDto* `robotMoves` - contains a list of moves performed by the robots during the audit.

AuditActivityDataDto

- *CellDto* `cell` - cell and its contents after the audit.
- *date* `startTime` - start date and time of the audit.
- *date* `endTime` - end date and time of the audit.

LibraryStatisticsDto

LostCartridgesEventDataDto

Extends EventDataDto.

MediaValidationDrivePoolModifiedEventDataDto

Extends EventDataDto.

- List *DriveDto* - list of drives in the media validation pool.

RailEventDataDto

Extends EventDataDto.

- *RailDto* `rail` - The rail that experienced the event.

PartitionEventDataDto

Extends EventDataDto.

- *PartitionDto* `partition` - partition that has been modified.

RobotMoveDto

Contains data about an individual robotic action involving a tape.

- *CellDto* `sourceCell` - cell ID for the cartridge at the start of the move.
- *CellDto* `destinationCell` - cell ID for the cartridge at the end of the move.

- *string* `moveType` - type of the move. Options are:
  - HAND — For motion performed by the robot. This involves a specific robot moving to a location, fetching a cartridge, moving to another location, and putting the cartridge into the destination cell.
  - ELEVATOR — For motion performed by an elevator. This is the movement of the elevator from one rail to another. The four physical cells have a set of Cells on each rail. A robot will put a cartridge in an elevator cell when the elevator is positioned on one rail. The elevator will then move to another rail. A different robot will then fetch the cartridge from the same physical cell. Because the physical cell is on a different rail, the fetch operation is from a different cell than the source.
  - PTP — For motion performed by a pass-thru port (PTP). This is the same as for elevators, but the source and destination cells are in different libraries in a library complex.
- *date* `startTime` - date/time stamp when the move was started.
- *date* `endTime` - date/time stamp when the move ended.
- *RobotStatusCode* `moveStatus` - status of robot after attempting this move. SUCCESS indicates the operation was successful. Other values indicate an error.

TestEventDataDto

Extends EventDataDto.

4

Enumeration Types

- HardwareStatusCode
- IpAddressType
- JobType
- JobStateType
- LabelWindowing
- LibraryComplexStateType
- LibraryControllerError
- LibraryProductionState
- LibraryRole
- LibraryStateType
- LogLevel
- MediumType
- ModuleType
- NetworkSettingsType
- PartitionStateType
- RequestErrorType
- RequestSource
- RequestStatus
- ResourceName
- ResourceState
- ResourceType
- RobotHardwareStatusCode
- RobotHomeEnd
- RobotSelector
- RobotStatusCode
- ScanType
- ScsiHostState
- SensorType
- ServiceIndicatorName
- SeviceIndicatorState
- SupportBundleOriginator
- SupportBundleState
- SystemReportType
- TopLevelDeviceStateType

CellContentsState

- INVALID

- MAGAZINE-ABSENT
- EMPTY
- READABLE
- UPSIDE-DOWN
- UNREADABLE
- UNKNOWN
- MOVING-IN
- MOVING-OUT
- MEDIA-VALIDATE
- NOT-AUDITABLE

CellState

- PRESENT
- NOT_PRESENT
- UNKNOWN

CellType

- CAP
- STORAGE
- DRIVE
- DROPCELL
- ELEVATOR
- PTP
- ROBOT
- SWAPCELL
- SYSCELL
- TURNTABLE
- UNKNOWN
- EMPTY
- ENDRAIL_CELL
- INVALID
- MODULE_LABEL

CellTypeSelector

- ALL
- CAP
- STORAGE

- DRIVE
- DROPCELL
- ELEVATOR
- PTP
- ROBOT
- SWAPCELL
- SYSCELL
- TURNTABLE
- UNKNOWN
- EMPTY
- ENDRAIL_CELL
- INVALID
- MODULE_LABEL

CommandTiming

- MOUNT
- DISMOUNT
- INTERMEDIATE

ComponentLocationState

- ONLINE
- OFFLINE
- UNKNOWN

ControlState

- INITIALIZING
- ONLINE
- OFFLINE
- GOING_ONLINE
- GOING_OFFLINE
- UNCONTROLLED
- GOING_TO_POWER_OFF
- REBOOTING
- UNKNOWN

CorrectiveActionsType

- REPLACE_DEVICE

- REMOVE_CARTRIDGE_FROM_CELL_OR_DRIVE
- INSTALL_MISSING_MAGAZINE_OR_BEZEL
- INSTALL_DEVICE
- CHECK_FANS_REPLACE_DEVICE_IF_FANS_OK
- CLOSE_DOORS_CHECK_BREAKERS
- CORRECT_CONFIGURATION
- CORRECT_CONFIGURATION_NEGOTIATION
- VERIFY_MODULE_POWER_ON_CHECK_WIRING
- VERIFY_DRIVE_MODULE_POWERED_ON
- RESTORE_POWER_TO_PDU
- CORRECT_SERIAL_NUMBER_VIA_GUI
- REPLACE_INSTALL_REQUIRED_DEVICES
- REPLACE_ROBOT
- REPLACE_ROBOT_FAILED_MOVEABLE
- REPLACE_ROBOT_FAILED_IMMOVEABLE
- TEST_FAULT_NO_ACTION_REQUIRED
- CONTACT_SUPPORT
- INSPECT_CAP_FOR_OBSTRUCTION
- REPAIR_SAFETY_DOOR
- CHECK_BREAKER
- IMPORT_COMPATIBLE_CLEANING_CART
- CHECK_CONNECTIONS
- CHECK_DRIVE_ARRAY_CABLE_CONNECTIONS
- CHECK_BREAKER_REPLACE_DEVICE
- WRITE_BUG

DestinationType

- ASR
- EMAIL
- SNMP
- SCI
- GUI
- UNKNOWN

DeviceStateType

- PRESENCE_UNKNOWN
- PRESENCE_DETECTED

- UPDATING_FIRMWARE
- INITIALIZING
- USABLE
- NOT_COMMUNICATING
- FAILED
- SUSPECT
- UPDATING_FW_COMPLETE
- SELF_INIT_NEEDED
- SELF_INITIALIZING
- SENSOR_INIT_NEEDED
- SENSOR_INITIALIZING
- HAND_INIT_NEEDED
- AUDIT_INIT
- HAND_INITIALIZING
- READING_FRAME_LABELS
- INACTIVE
- MOVING
- ACTIVE_MOVING_TO_STALL
- INIT_MOVING_TO_STALL
- FETCHING
- PUSHING
- PUTTING
- AUDITING
- NEEDS_RESET
- FAILED_MOVABLE
- FAILED_IMMOVABLE
- CALIBRATING
- EMPTY
- CART_PRESENT
- MOUNTED
- BUSY_UNLOADING
- BUSY_LOADING
- BUSY_CLEANING
- BUSY_VALIDATING
- FAIL_UNSUP_DRIVE_TYPE
- FAIL_NOT_UNLOADABLE
- FAIL_NOT_LOADABLE

- OPERATIVE
- INOPERATIVE
- DEGRADED
- NOT_LICENSED
- UNKNOWN
- OPEN
- OPENING
- CLOSING
- FREE
- UNLOCKED
- LOCKED
- AUTOLOCKED
- REBUILDING
- NORMAL
- ABSENT

DeviceType

- AEM
- AEM_DOOR
- AEM_SERVICE_PANEL
- AEM_STATUS_CABLE
- BASE
- BASE_SERVICE_PANEL
- BASE_CARD_CAGE
- BASE_MAIN_HARNESS
- CAP
- ROTARY_CAP
- AEM_CAP
- CAP_CABLE
- CEM
- PEM
- DEM
- LIBRARY
- LIBRARY_COMPLEX
- DOOR_SWITCH
- DRIVE
- DRIVE_ARRAY_ASSEM

- DRIVE_BACKPLANE
- DRIVE_AC_POWER
- DRIVE_DC_POWER
- DRIVE_NETWORK_CABLE
- DRIVE_POWER_SUPPLY
- DRIVE_TRAY
- ENCRYPTION_CARD
- ENCRYPTION_CARD_CONFIGURATOR
- ETHER_SWITCH_ASSEM
- ETHERNET_CABLE
- HBQ
- LOCATION_ID_CABLE
- LOCATION_ID_TERMINATOR
- FCPORTDEV
- FEATURE
- LOB
- LOC
- LOD
- LOEB
- LOER
- LOES
- LOF
- LOH
- LOID
- LON
- LOS
- LOV
- LOX
- LOY
- MAGAZINE
- OPERATOR_PANEL
- PDU
- POWER_SUPPLY
- PUW
- PUZ
- RAIL
- RAIL_CONTROLLER_CABLE

- RAIL_AC_POWER
- RAIL_DC_POWER
- ROBOT
- SAFETY_DOOR
- STORAGE_ACCESS
- WEB_CAMERA
- RDA_WRITER
- MODULE_MAGAZINE
- UNKNOWN
- NO_DEVICE
- BASEMOD_DOOR
- DRIVEMOD_DOOR
- STORAGE_SERIALIZER
- LEG_LOD_SERIALIZER

DeviceTypeSelector

- ALL
- AEM
- AEM_DOOR
- AEM_SERVICE_PANEL
- AEM_STATUS_CABLE
- BASE
- BASE_SERVICE_PANEL
- BASE_CARD_CAGE
- BASE_MAIN_HARNESS
- CAP
- ROTARY_CAP
- AEM_CAP
- CAP_CABLE
- CEM
- DEM
- DOOR_SWITCH
- DRIVE
- DRIVE_ARRAY_ASSEM
- DRIVE_BACKPLANE
- DRIVE_AC_POWER
- DRIVE_DC_POWER

- DRIVE_NETWORK_CABLE
- DRIVE_POWER_SUPPLY
- DRIVE_TRAY
- ENCRYPTION_CARD
- ENCRYPTION_CARD_CONFIGURATOR
- ETHER_SWITCH_ASSEM
- ETHERNET_CABLE
- HBQ
- LOCATION_ID_CABLE
- LOCATION_ID_TERMINATOR
- FCPORTDEV
- FEATURE
- LOB
- LOC
- LOD
- LOEB
- LOER
- LOES
- LOF
- LOH
- LOID
- LON
- LOS
- LOV
- LOX
- LOY
- MAGAZINE
- MODULE_MAGAZINE
- OPERATOR_PANEL
- PDU
- POWER_SUPPLY
- PUW
- PUZ
- RAIL
- RAIL_CONTROLLER_CABLE
- RAIL_AC_POWER
- RAIL_DC_POWER

- ROBOT
- SAFETY_DOOR
- WEB_CAMERA
- RDA_WRITER
- UNKNOWN
- NO_DEVICE
- STORAGE_SERIALIZER

DoorState

- OPENED
- CLOSED
- SEPERATED
- UNKNOWN

DriveActivityStatusCode

- SUCCESS
- FAILED
- SKIPPED

DriveInterfaceType

- UNKNOWN
- SCSI
- ESCON
- FICON
- FICON_NATIVE
- FC
- SAS
- FCOE

DriveOperationStatus

- SUCCESS
- MOUNT_FAILED
- DISMOUNT_FAILED

DriveProtocol

- UNKNOWN
- ADI
- TTI

ErrorCode

- UNKNOWN
- UNRESPONSIVE
- UNRESPONSIVE_UNRECOVERABLE
- CODE_DOWNLOAD_FAILS
- DEVICE_ERROR
- SUCCESS
- COMM_FAILURE
- LOD_COMM_FAILURE
- INVALID_REQUEST
- NEEDS_RESET
- FAILED_MOVABLE
- FAILED_IMMOVABLE
- NOT_COMMUNICATING
- CANT_MOVE_ON_RAIL
- CANT_MOVE_WRIST
- VISION_INOP
- CANT_FIND_TARGET
- LOC_UNUSABLE
- NO_CART_IN_HAND
- CART_IN_HAND
- CART_STUCK
- CART_DROPPED
- CELL_EMPTY
- CELL_FULL
- LABEL_MISCOMPARE
- MISBUCKLE
- DRIVE_STATE_CHANGE
- LINK_UP
- LINK_DOWN
- FAN_FAILURE
- FAN_RECOVERED
- OVER_TEMPERATURE
- FPGA_FAULT
- BUS_CONTROL_LOST
- POWER_ON
- POWER_OFF

- OVER_VOLTAGE_SHUTDOWN
- DOOR_OPEN
- DOOR_CLOSED
- SAFETY_CARD_BATTERY_LOW
- ROBOTICS_DISABLED
- DOOR_SWITCH_FAULT
- LOCATE_BUTTON_PRESS
- TAPE_DRIVE_POWER_SWITCH_STATE_CHANGE
- TAPE_DRIVE_POWER_FAILED
- POWER_SUPPLY_AC_OK_STATE_CHANGE
- POWER_SUPPLY_DC_OK_STATE_CHANGE
- AC_POWER_SUPPLY_FAILED
- DC_POWER_SUPPLY_FAILED
- RAIL_DC_OK_STATE_CHANGE
- PDU_BREAKER_OPEN
- PDU_BREAKER_CLOSED
- PDU_AC_OK_STATE_CHANGE
- PDU_DC_OK_STATE_CHANGE
- UNEXPECTED_ANNOUNCE
- FET_SHORT
- POWER_BAD
- OVER_CURRENT
- OVER_VOLTAGE
- UNDER_VOLTAGE
- DRIVE_FAN_FULL_ON
- DEVICE_ADDED
- DEVICE_REMOVED
- MISSING_DEVICE
- MISSING_DEVICE_ANNOUNCE_LEFT
- MISSING_DEVICE_DETECTED_LEFT
- MISSING_DEVICE_ANNOUNCE_RIGHT
- MISSING_DEVICE_DETECTED_RIGHT
- INVALID_CONFIGURATION
- EXCEPTION
- TEST_EVENT
- INVALID_FRU_BASE_PART
- DRIVE_TRAY_SERIAL_NUMBER_MISMTACH

- DRIVE_EMPTY_BUT_HAS_TAPE
- DEGRADED_TRANSITION
- INITIALIZING_TRANSITION
- INOPERATIVE_TRANSITION
- OPERATIVE_TRANSITION
- UNKNOWN_TRANSITION
- STARTUP_TRANSITION
- POWER_OFF_TRANSITION
- LIBRARY_INIT_TIMEOUT
- SOAP_ERROR
- HARDWARE_MALFUNCTION
- SOFTWARE_EXCEPTION
- MISSING_PARAM
- INVALID_PARAM
- INVALID_HEX_VALUE
- NEW_THREAD_ERROR
- UNSPECIFIED_THREAD_ERROR
- PRIVATE_THREAD_DATA_NOT_ALLOCATED
- ADD_LEAF_ERROR
- MEM_ALLOC_ERROR
- BUFFER_FULL
- RING_BUFFER_FULL
- INVALID_RUN_MODE
- PIC_CODELOAD_FAILURE
- FPGA_CODELOAD_FAILURE
- TARGET_IMAGE_CRC_MISMATCH
- FPGA_CHKSUM_MISMATCH
- FPGA_INVALID_BIN_RECORD
- FPGA_REPEAT_FIRST_FLAG
- FPGA_REPEAT_LAST_FLAG
- FPGA_MISSING_FIRST_FLAG
- FPGA_FLASH_ERASE_FAILED
- FPGA_FLASH_WRITE_FAILED
- CHKSUM_MISMATCH
- INVALID_HEX_RECORD
- REPEAT_FIRST_FLAG
- REPEAT_LAST_FLAG

- MISSING_FIRST_FLAG
- MISSING_LAST_FLAG
- FLASH_ERASE_FAILED
- FLASH_WRITE_FAILED
- HEX_RECORD_SET_CRC_MISMATCH
- UART_NOT_CONFIGURED
- IIC_TRANSACTION_FAILED
- SPI_TRANSACTION_FAILED
- DRIVE_POWER_UNABLE_TO_TURN_ON
- DRIVE_POWER_UNABLE_TO_TURN_OFF
- TIMEOUT
- BAD_ARGUMENT
- UNEXPECTED_VALUE
- NETWORK_DOWN
- MOTOR_BUSY
- MOTOR_OPERATION_TIMEOUT
- CAP_HARDWARE_FAULT
- CAP_BUTTON_PRESS
- INTERNAL_SOFTWARE_ERROR
- NO_COMMUNICATION
- NO_ERROR_DATA
- PDU_PHASE_C_BREAKER_OPEN
- PDU_PHASE_B_BREAKER_OPEN
- PDU_PHASE_A_BREAKER_OPEN
- PDU_TWENTY_FOUR_VOLT_OK
- PDU_PHASE_C_PRESENT
- PDU_PHASE_B_PRESENT
- PDU_PHASE_A_PRESENT
- PDU_PHASE_C_BREAKER_CLOSED
- PDU_PHASE_B_BREAKER_CLOSED
- PDU_PHASE_A_BREAKER_CLOSED
- PDU_TWENTY_FOUR_VOLT_NOT_OK
- PDU_PHASE_C_NOT_PRESENT
- PDU_PHASE_B_NOT_PRESENT
- PDU_PHASE_A_NOT_PRESENT
- RAIL_POWER_SUPPLY_FAILED
- RAIL_POWER_SUPPLY_AC_INPUT_OK

- RAIL_POWER_SUPPLY_DC_OUTPUT_OK
- RAIL_POWER_SUPPLY_OK
- RAIL_POWER_SUPPLY_AC_INPUT_NOT_OK
- RAIL_POWER_SUPPLY_DC_OUTPUT_NOT_OK
- LOS_X_FORTY_EIGHT_VOLT_OK
- LOS_X_FORTY_EIGHT_VOLT_NOT_OK
- BREAKER_OPEN
- BREAKER_CLOSED
- CANT_JUMP_TO_APP
- FAILED
- READ_FAILURE
- WRITE_FAILURE
- MEDIA_WEAROUT
- RAID_FAILURE
- DEVICE_OVERHEATING
- TEST_ERROR_1_SUPPORT_BUNDLE
- TEST_ERROR_2_NO_SUPPORT_BUNDLE
- TWELVE_VOLTORING_OK
- TWELVE_VOLTORING_FAILURE
- POWER_OK
- POWER_FAILURE
- DRIVE_POWER_SUPPLY_FAILED
- DRIVE_POWER_SUPPLY_AC_INPUT_OK
- DRIVE_POWER_SUPPLY_DC_OUTPUT_OK
- DRIVE_POWER_SUPPLY_OK
- DRIVE_POWER_SUPPLY_AC_INPUT_NOT_OK
- DRIVE_POWER_SUPPLY_DC_OUTPUT_NOT_OK
- DRIVE_UNLOAD_FAILED
- DRIVE_TYPE_NOT_SUPPORTED
- DRIVE_FAILURE_NOT_UNLOADABLE
- DRIVE_FAILED
- MOVE_OUT_OF_WAY_FAILURE
- AEM_SVCK_ON
- AEM_SVCK_OFF
- FRU_UPDATE_FAILED
- OFFLINE
- CONFIGURATION_NEGOTATION_FAILED

- RAIL_OBSTRUCTED
- NO_ANNOUNCEMENT
- MISSED_HEARTBEAT
- ASR_SUPPORT_BUNDLE_REQUEST
- MISSING_ROBOT
- MISSING_ACCESS_CONTROLLER_MODULE_LEFT
- MISSING_ACCESS_CONTROLLER_MODULE_RIGHT
- MISSING_ROBOT_CONTROLLER
- MISSING_RAIL_CONTROLLER
- TEST_FAULT
- NON_ANNOUNCING_DEVICE_RIGHT
- NON_ANNOUNCING_DEVICE_LEFT
- FAILED_TO_CONFIGURE_ENCRYPTION_CARD
- FAILED_DRIVE_DISCOVERY
- CELL_CONTENTS_MISMATCH
- DEADBOLT_HW_FAULT
- MECH_NOT_BUSY_TIMEOUT
- MECH_ERROR
- MECH_MOVE_TIMEOUT
- SAFETY_DOOR_SEPARATION
- CLEAN_CART_DOES_NOT_EXIST
- NO_RULE_FOUND
- DROPPED_OFF_CART_AT_INIT
- DEVICE_WAS_RESET
- MISSING_REQUEST_MESSAGE
- UNABLE_TO_INIT_ROTARY_CAPS
- STORAGE_IDENTIFICATION_MISSING
- SYSTEM_CELLS_FULL
- STORAGE_FAILURE
- DRIVE_ARRAY_CONNECTION_FAULT
- CODELOAD_IN_PROGRESS
- FPGA_REG_READ_FAILED
- FPGA_REG_WRITE_FAILED
- FRU_READ_FAILED
- LINK_DOWN_FAILED
- LINK_UP_FAILED
- RUN_MODE_CHANGE_IN_PROGRESS

- UNDER_VOLTAGE_SHUTDOWN
- NO_LONGER_USED
- FC_PORT_FAILURE

EventCategory

- ASR
- FAULT
- CARTRIDGE_MOVEMENT
- MEDIA_VALIDATION
- DEVICE
- DOOR
- CAP
- PARTITION
- CLEANING_REQUIRED
- LIBRARY
- HEARTBEAT
- TEST
- UNKNOWN

EventSeverity

- SEVERE
- ERROR
- WARNING
- INFO
- NONE
- UNKNOWN

EventType

- MOVE
- TEST
- DEV_INSTALLED
- DEV_FAILED
- DEV_REMOVED
- DEV_CNTRL_STATE_CHNG
- LIBRARY_STATE_CHANGE
- RAIL_STATE_CHANGE
- LIBRARY_COMPLEX_STATE_CHANGE

- LICENSED_CAPACITY_CHANGE
- BOOT_COMPLETE
- AUDIT_COMPLETE
- LIBRARY_STATISTICS
- LOST_CARTRIDGES
- PARTITION
- DOOR_OPEN
- DOOR_CLOSE
- INTERMEDIATE_MOUNT_DRIVE
- MEDIA_VAL_DRV_POOL_MODIFIED
- CAP_MOVE
- CAP_READY_OPEN
- CAP_OWN_OVER_RIDDEN
- DRV_CLEAN_NEEDED
- HEARTBEAT
- FAULT_EVENT
- ASR_CONFIG_REQUEST
- ASR_TIME_REPORT_REQUEST
- ASR_LIBRARY_VERSION_REQUEST
- ASR_SUPPORT_BUNDLE_REQUEST
- UNKNOWN

FanHealth

- GOOD
- MARGINAL
- POOR
- UNSTABLE
- NO_READING
- GREEN
- YELLOW
- ORANGE
- RED
- UNKNOWN

FastLoadType

- IMMEDIATE
- FAST

- NORMAL
- UNKNOWN

FaultSymptomCodeType

- UNRESPONSIVE
- CODE_DOWNLOAD_FAILED
- COMMUNICATION_FAILURE
- INOP
- FPGA_FAULT
- SAFETY_CARD_BATTERY_LOW
- FAN_FAILURE
- AN_RECOVERED
- POWER_SUPPLY_FAILED
- FAILED_MOVEABLE
- FAILED_IMMOVEABLE
- ROBOT_UNRECOVERABLE
- CANT_FIND_TARGET
- CARTRIDGE_STUCK
- MISBUCKLE
- ROBOTICS_DISABLED
- PDU_FAILED
- AC_POWER_LOST
- DEVICE_MISSING
- UNEXPECTED_MODULE
- CONFIGURATION_NEGOTIATION_FAILED
- DRIVE_TRAY_SERIAL_NUMBER_FAULT
- COMPATIBLE_CLEAN_CART_DOES_NOT_EXIST
- LIBRARY_INIT_TIMEOUT
- SOFWARE_FAULT
- INVALID_CONFIGURATION
- OVER_TEMPERATURE
- DEVICE_FAULT
- TEST_FAULT
- UNEXPECTED_ANNOUNCE
- INTERNAL_SOFTWARE_ERROR
- SAFETY_DOOR_SEPARATION
- OPERATION_TIMEOUT

- ROTARY_CAPS_INIT_FAILURE
- DRIVE_ARRAY_CABLE_FAULT
- UNEXPECTED_DEVICE_REMOVED

FcPortState

- FC_UP
- FC_DOWN
- FC_UNKNOWN

Feature

- CAPACITY
- DUAL_PORT
- PARTITIONING
- SHELL_ACCESS
- UNKNOWN

FirmwareType

- PIC_APP
- PIC_BOOT
- PIC_BOOTS
- PIC_APPH
- PIC_APPL
- FPGA
- SUB_FPGA
- OMAP_APPS
- OMAP_DSP
- OMAP_LINUX
- OMAP_FILESYS
- OMAP_UBOOT
- OMAP_UBL_LOADER
- OMAP_MANIFEST_FORMAT
- DRIVE
- ENDR_CARD

FruType

- FRU — Field Replaceable Unit. Meets the rules to be a field replaceable device. Can be easily replaced in the field.

- FRP — Field Replaceable Part. A device that does not fully meet the rules for FRUs, but that can be replaced, with some difficulty and tools, in the field.
- NONE — Not field replaceable.
- UNKNOWN

HardwareStatusCode

- NONE
- DRIVE_CONNECTION_BUSY
- DRIVE_NOT_RESPONDING
- LOD_CONNECT_FAILURE
- LOD_WRITE_FAILURE
- LOD_READ_FAILURE
- LOGIN_FAILURE
- LOAD_FAILURE
- LOAD_FAILURE_DRIVE_EMPTY
- UNLOAD_FAILURE_NO_FORCE
- UNLOAD_FAILURE_STILL_SEATED
- TAPE_MOUNTED_DURING_ENCR_CARD_CONFIG
- CHECK_CONDITION
- INVALID_PACKET_RECEIVED
- COMMAND_NOT_SUPPORTED
- DRIVE_RESET
- INVALID_STX_BYTE
- INVALID_ETX_BYTE
- INVALID_SEQ_NUM
- CRC_ERROR
- NAK_RECEIVED
- INCOMPLETE_SCSI_RESPONSE
- BAD_SCSI_STATUS
- UNEXPECTED_SCSI_RESPONSE
- INVALID_FIELD
- BAD_CHECKSUM
- UNEXPECTED_PORT_LOGIN
- UNEXPECTED_PORT_LOGOUT
- MEDIUM_ERROR
- INTERNAL_SOFTWARE_ERROR
- DRIVE_VERIFY_STATE_FAILED

- ENCR_CARD_CONFIG_FAILED
- ENCR_CARD_CONFIG_FAILED_STILL_ON_DEFAULT_IP
- ENCR_CARD_CONFIG_FAILED_CANNOT_CONNECT
- ENCR_CARD_FAILED_TO_RESET
- TELNET_CONNECT_FAILURE
- TELNET_UNEXPECTED_RESPONSE
- TELNET_READ_TIMEOUT
- TELNET_SOCKET_FAILURE
- TELNET_CONNECTION_CLOSED
- TELNET_SOCKET_READ_FAILURE
- TELNET_SOCKET_WRITE_FAILURE
- NOTDEFINED
- PHCNOERROR
- PHCNOTINITIALIZED
- PHCSPIFAILURE
- PHCUSBFAILURE
- PHCIICFAILURE
- PHCPHYFAILURE
- PHCFPGAFAILURE
- PHCEEPROMFAILURE
- PHCHOTSWAPFAILURE
- PHCNETSWITCHFAILURE
- PHCCOMMINITFAILURE
- PHCCOMMNACKRECVD
- PHCMECHNOTPRESENT
- PHCLOWVOLTAGE
- PHCOVERVOLTAGE
- PHCOVERCURRENT
- PHCPROTOCOLFAILURE
- LEFTOFBASE
- RIGHTOFBASE
- PHCFANONEFAILURE
- PHCFANTWOFAILURE
- PHCFANTHREEFAILURE
- PHCFANFOURFAILURE
- PHCFANONERECOVERED
- PHCFANTWORECOVERED

- PHCFANTHREERECOVERED
- PHCFANFOURRECOVERED
- SUCCESS
- BAD-USAGE
- APPL-NOT-READY
- APPL-TASK-NOT-READY
- APPL-TASK-FAILURE
- TARGET-EXCEEDED-MAX-ALLOWABLE-BARS
- TARGET-NOT-RECOGNIZED
- TARGET-OFFSET-INVALID"
- TARGET-LAST-RESULT-CODE
- TARGET-FAILED-TO-CALIBRATE-SCANNER
- SCAN-FAULT-OPEN-FAILURE
- SCAN-FAULT-POWER-UP-FAILURE
- SCAN-FAULT-INITIALIZATION-FAILURE
- SCAN-APP-RECEIVED-NO-MESSAGES
- SCAN-APP-TO-DRIVER-READ-TIMEOUT
- SCAN-APP-TO-DRIVER-WRITE-TIMEOUT
- SCAN-APP-RECEIVED-LLF-NAK-FN
- SCAN-APP-RECEIVED-LLF-NAK-CHKSUM
- SCAN-APP-RECEIVED-LLF-BUSY
- SCAN-APP-RECEIVED-PACKET-WITH-BAD-CHKSUM
- SCAN-APP-RECEIVED-NR
- SCAN-APP-BAD-STATUS
- SCAN-APP-PACKET-SIZE-TOO-LARGE
- SCAN-APP-ASCII-TO-INT-PARSE-FAILURE
- SCAN-APP-LOOKING-FOR-TARGET-GOT-BARCODE
- SCAN-APP-RECEIVED-LINE-STATUS-ERROR-INDICATION
- SCAN-APP-TARGETING-DATA-TOO-SHORT
- SCAN-FIRMWARE-DOWNLOAD-FAILURE
- SCAN-LABEL-CHARACTER-TRANSLATED
- SCAN-LAST-RESULT-CODE
- SRV-MECH-STALLED
- SRV-MECH-STALLED-ON-INIT
- SRV-MECH-OUTSIDE-STOPLOCK
- SRV-ISR-LOGICAL-FAILURE
- ERR-SRV-UNKNOWN-REQUEST-TYPE

- dont-use-ERR-SRV-UNEXPECTED-SYS-ERROR-RET
- ERR-SRV-BAD-CHK-MOVE-CALC
- ERR-SRV-DEST-OUTSIDE-OPER-RANGE
- ERR-SRV-ILLEGAL-PROFILE-TYPE
- ERR-SRV-OVERCURRENT
- ERR-SRV-EXCESSIVE-POSITION-ERROR
- ERR-SRV-TACH-PHASE-ERROR
- ERR-SRV-CANT-START-NOT-IN-STOPLOCK
- ERR-SRV-ISR-REENTERED
- ERR-SRV-SATURATION-CURRENT-REQUESTED-TOO-LONG
- ERR-SRV-MECH-DROPPED-OUT-OF-STOPLOCK
- ERR-SRV-MECH-FAILED-TO-SETTLE-INTO-STOPLOCK
- ERR-SRV-OPERATING-RANGE-OUT-OF-SPEC
- ERR-SRV-INVALID-THETA-Z-RANGE-COMBO
- ERR-SRV-REDEFINED-LIB-CONFIG
- ERR-SRV-BAD-MECH-ID-IN-ISR
- ERR-SRV-ILLEGAL-REQUEST-OPTION
- ERR-SRV-FAILED-TO-ENCOUNTER-CARTRIDGE
- ERR-SRV-FAILED-TO-DISENGAGE-CARTRIDGE
- ERR-SRV-FAILED-TO-SEAT-CARTRIDGE
- ERR-SRV-FAILED-TO-UNSEAT-CARTRIDGE
- ERR-SRV-REQUEST-ALREADY-ACTIVE-AGAINST-MECHANISM
- ERR-SRV-CANT-MOVE-ARM-HAND-IS-ACTIVE
- ERR-SRV-CANT-MOVE-HAND-ARM-IS-ACTIVE
- ERR-SRV-UNEXPECTED-RESPONSE
- ERR-SRV-CANT-GET-WITH-HAND-FULL
- ERR-SRV-CANT-PUT-WITH-HAND-EMPTY
- ERR-SRV-MOVE-ABORTED
- ERR-SRV-HAND-NOT-SAFE-HAND-IS-INOPERATIVE
- ERR-SRV-HAND-NOT-SAFE-REACH-NOT-RETRACTED
- ERR-SRV-HAND-NOT-SAFE-CARTRIDGE-IS-UNSEATED-IN-GRIP
- ERR-SRV-MECHANISM-NOT-INITIALIZED
- ERR-SRV-MECHANISM-SHUTDOWN
- ERR-SRV-MECHANISM-NOT-OPERATIONAL
- ERR-SRV-USER-REQ-THETA-MOVE-FOR-SCAN
- ERR-SRV-CANT-CLEAR-AMP-ENABLE
- ERR-SRV-SATURATION-CURRENT-READ-TOO-LONG

- ERR-SRV-MINIMUM-INIT-MOVE-NOT-DETECTED
- ERR-SRV-REACH-SAFE-SENSOR-FAIL
- ERR-SRV-REACH-GRIP-OVERCURRENT
- ERR-SRV-AMP-ENABLE-FAIL
- ERR-SRV-FAILED-STALL
- ERR-SRV-FAILED-STALL-OBSTRUCTED
- ERR-SRV-DEST-OUTSIDE-OPER-RANGE-ADJUSTED
- ERR-SRV-NOT-RESPONSE-DISCARDED
- ERR-SRV-CANT-FIND-REACH-DEPTH
- ERR-SRV-POWER-LOW-ERROR
- ERR-SRV-REQUEST-QUEUED-TIMEOUT
- ERR-SRV-REQUEST-ACTIVE-TIMEOUT
- ERR-SRV-BAD-MECH-ID-IN-COORD
- ERR-SRV-SYS-MSG-ALLOC-FAIL
- ERR-SRV-SYS-MSG-SEND-FAIL
- ERR-SRV-SYS-MSG-RECV-FAIL
- ERR-SRV-SYS-MSG-BAD-SIZE
- ERR-SRV-SYS-MSG-GET-CONTENT-FAIL
- ERR-SRV-SYS-MSG-SET-CONTENT-FAIL
- ERR-SRV-SYS-MSG-RELEASE-FAIL
- ERR-SRV-HAND-NOT-SAFE
- ERR-SRV-HAND-INIT-FAIL-NOT-EMPTY
- ERR-SRV-COORD-SEND-MECH-REQUEST-FAILED
- ERR-SRV-HALL-ERROR
- ERR-SRV-HDW-OVER-CURRENT-ERROR
- ERR-SRV-HDW-UNKNOWN-ERROR
- ERR-SRV-EXCESSIVE-MOTOR-HEATING
- ERR-SRV-SAT-CURRENT-REQ-TOO-LONG-STALL-MIN-NOT-REACHED
- ERR-SRV-EXCESSIVE-TRACK-STRACK-RELATIVE-ERROR
- ERR-SRV-DOOR-OPEN-ERROR
- ERR-SRV-ISR-STARTUP-FAILED
- ERR-SRV-BAD-MECH-ID-AT-MECH-LAYER
- ERR-SRV-COACTIVE-QUEUED-FAILURE
- ERR-SRV-COACTIVE-QUEUED-TIMEOUT
- ERR-SRV-FAILED-TO-REACH-STALL-POSITION
- ERR-SRV-BAD-MECH-ID-AT-USER
- ERR-SRV-UNDETECTED-AMP-DISABLE

- ERR-SRV-COORD-SEQUENCING-FAILED
- ERR-SRV-FAILED-ZERO-TACH-ON-STALL
- ERR-SRV-FAILED-MECH-LIMIT
- CMO-FAILED-CARTESIAN-LOOKUP-AUDIT
- CMO-FAILED-CARTESIAN-LOOKUP-FETCH
- CMO-FAILED-CARTESIAN-LOOKUP-PUT
- CMO-FAILED-CARTESIAN-LOOKUP-TARGET
- CMO-FAILED-CARTESIAN-LOOKUP-MOVE
- CMO-COULD-NOT-STORE-TARGET-CALIBRATION
- CMO-REACH-NOT-SAFE-DETECTED
- CMO-HAND-EMPTY-DETECTED
- CMO-HAND-FULL-DETECTED
- CMO-FAILED-TARGET-CALIBRATION
- CMO-FETCH-RETRY-PERFORMED
- CMO-PUT-RETRY-PERFORMED
- CMO-CART-LABEL-MISCOMPARE
- CMO-CELL-FULL-DETECTED
- CMO-CELL-EMPTY-DETECTED
- CMO-END-OF-RAIL-ID-FAILURE
- CMO-INIT-FAILURE
- CMO-FAILED-CARTESIAN-LOOKUP-PROX
- CMO-FAILED-PROX-READ
- CMO-MOVE-RETRY-PERFORMED
- CMO-INCONSISTENT-SUCCESS-ON-FETCH
- CMO-INCONSISTENT-SUCCESS-ON-PUT
- CMO-CELL-SCAN-USED-FOR-AUDIT
- CMO-DEPRECATED-POSITION-USED-TO-TARGET
- CMO-USED-INITIAL-TARGETED-LOCATION
- CMO-AUDIT-LABEL-MIN-LENGTH-NOT-MET
- CMO-FAILED-UNSET-TARGET-CALIBRATION
- CMO-RECOVER-FOR-FETCH-PUTBACK-NOT-ATTEMPTED
- CMO-RECOVER-FOR-FETCH-SRV-RECOVERED-CART
- CMO-FAILED-HANDBOT-RANGE-GET
- CMO-CALIBRATION-RETRY-PERFORMED
- CMO-ROBOT-Z-RANGE-IS-SHORT
- CMO-ROBOT-TRACK-RANGE-IS-SHORT
- CMO-ROBOT-WRIST-RANGE-IS-SHORT

- CMO-INVALID-ADDRESS
- CMO-SCANNER-HW-NOT-SUPPORTED
- CMO-SCANNER-UNRECOGNIZED-HW-VERSION
- CMO-CART-NO-LABEL-FOUND
- CMO-CRIMSON-FRAMELABEL-UNRECOGNIZED-LABEL
- CMO-CRIMSON-EMPTY-FRAMELABEL-CELL
- CMO-CRIMSON-FRAMELABEL-NO-LABEL
- CMO-CRIMSON-FRAMELABEL-PROBLEM
- CMO-END-OF-RAIL-ID-FAILURE-CRIMSON
- CMO-CRIMSON-FRAMELABEL-WARNING
- CMO-HAND-NOT-AT-LOCATION
- CMO-CAP-MAGAZINE-NO-INSTALLED
- CMO-ARM-NOT-OPERATIONAL-DETECTED
- CMO-END-OF-RAIL-SET-REV-FAILURE-CRIMSON
- CMO-CARTRIDGE-DROPPED-OFF-AT-INIT
- ERROR-OPENING-CALIBDATA-FILE
- ERROR-OPENING-CFGLOCATIONS-FILE
- ERROR-OPENING-MULTI-ROW-SCAN-DISABLE-FILE
- ERROR-FAILED-MECH-LIMIT-OPERATION
- DIRCT-FAULT-BAD-REQU
- DIRCT-FAULT-ZERO-REFERENCE-FAILURE
- DIRCT-FAULT-BAD-END-OF-RAIL-INIT
- DIRCT-FAULT-ROBOT-SET-DIRECTION-FAILURE
- DIRCT-FAULT-SET-CONFIG-MAP-FAILURE
- DIRCT-FAULT-SEND-MOVE-MAP-FAILURE
- DIRCT-FAULT-INVALID-Z-HEIGHT
- DIRCT-FAULT-REBUILD-CONFIG-MAP-FAILURE
- DIRCT-FAULT-GET-CONFIG-FAILURE
- DIRCT-FAULT-READ-ALL-CRIMSON-LABELS
- DIRCT-FAULT-SET-CRIMSON-CONFIG-MAP-FAILURE
- DIRCT-FAULT-BAD-END-OF-RAIL-INIT-CRIMSON
- DIRCT-FAULT-MODULE-DEFINITION-MAP-FAILURE
- DIRCT-FAULT-GET-LOCATIONS-FAILURE
- DIRCT-FAULT-INIT-HAND-CRIMSON
- DIRCT-WARNING-TRACK-STRACK-RANGE-MISMATCH-CRIMSON
- DIRCT-FAULT-TRACK-STRACK-RANGE-MISMATCH-CRIMSON
- DIRCT-FAULT-INVALID-ADDRESS

IpAddressType

- VFOUR
- VSIX
- UNKNOWN

JobType

- POSITION_ROBOT
- MOVE_TO_SERVICE_BAY
- MOVE_UNTIL_STALL
- FETCH
- PUT
- REQUEST_TO_PUT
- LOAD
- UNLOAD
- DISMOUNT_METRICS
- FETCH_FROM_DRIVE
- CHANGE_DEVICE_STATE
- TOP_LEVEL_AUDIT
- ROBOT_AUDIT_COLUMN
- TOP_LEVEL_MOVE
- DIAGNOSTIC_TOP_LEVEL_MOVE
- DIAGNOSTIC_EXCHANGE
- TOP_LEVEL_DRIVE_CLEAN
- DRIVE_CLEAN
- DRIVE_MEDIA_VALIDATION
- CONTIGUOUS_MOVE
- AEM_CAP_BUTTON_HANDLER
- CAP_BUTTON_HANDLER
- SAFETY_DOOR_HANDLER
- MOVE_ELEVATOR
- MOVE_PTP
- LOAD_CAP
- UNLOAD_CAP
- ENABLE_LOC
- PIC_CODE_LOAD
- LOB_CODE_LOAD
- LOS_CODE_LOAD

- FRU_UPDATE
- DRIVE_TRAY_FRU_UPDATE
- DRIVE_DISCOVERY
- DRIVE_CHECK_ENCRYPTION_CARD
- DRIVE_CONFIGURE_ENCRYPTION_CARD
- DRIVE_STATE_CHANGED
- DRIVE_POWER_STATE_CHANGED
- ROBOT_SENSOR_INIT
- ROBOT_SELF_INIT
- ROBOT_HAND_INIT
- SCAN_LIBRARY
- INIT_FRAME_LABELS
- INIT_RAIL
- REINIT_RAIL
- DETERMINE_ROBOT_LOC
- RAIL_SWEEP
- UPDATE_LOB_PRESENCE
- RAIL_POWER_CYLCLE
- RAIL_POWER_DOWN
- HNDL_DRPOFF_CELL
- LOG_DIAG_ACTION
- DEFAULT_DEV_ONLINE
- FAULTED_TOP_LEVEL_PAPERWORK
- ASR_REQUEST_SUPPORT_BUNDLE
- REQUEST_SUPPORT_BUNDLE
- AEM_OVER_TEMP
- DOOR_SWITCH_FAULT
- DRIVE_TRAY_FAN_FAILURE
- DRIVE_TRAY_OVER_TEMP
- ETHERNET_SWITCH_ASSEMBLY_OVER_TEMP
- FAN_ASSEMBLY_FAILURE
- FAULT_DEVICE
- FAULT_EVENT
- FAULT_ROBOT
- HANDLE_DOOR_STATE_CHANGE
- HANDLE_LOW_SAFETY_BATTERY
- LOCATE_LIBRARY

- MAIN_CARD_CAGE_OVER_TEMP
- MODULE_POWER_DIAGNOSTIC
- NETWORK_DIAGNOSTIC
- REBOOT_DEVICE
- TOGGLE_TAPE_DRIVE_POWER
- LOES_DEVICE_ADD_DETECTED
- LOER_DEVICE_ADD_DETECTED
- LOC_DEVICE_ADD_DETECTED
- DRIVE_AUDIT
- ONLINE_FC_PORT_DEV
- OMAP_CODE_LOAD
- FC_PORT_DEV_LICENSING
- OMAP_CODE_LOAD
- RECOVER_DEVICE
- PING_DEVICE
- RESET_DEVICE
- MISSING_DEVICE
- UNKNOWN
- DOWNLOAD_ROBOT_LOG
- ROBOT_CALIBRATION
- TOP_LEVEL_CALIBRATE
- ROBOT_SWEEP
- ROBOT_MOVE_TACH_COUNT
- ROBOT_MOVE_LOCATION
- ROBOT_MOVE_TO_CELL
- RECOVER_ROBOT
- ROBOT_FRU_DIAGNOSIS
- RE_INITIALIZE_ROBOT
- GET_ROBOT_TO_HOME_END
- PUSH_ROBOT_TO_HOME_END
- HANDLE_ROBOT_ADDITION
- LOER_DEVICE_REMOVAL_DETECTED
- LOES_DEVICE_REMOVAL_DETECTED
- LOC_DEVICE_REMOVAL_DETECTED
- ROBOT_CELL_TO_CELL
- OFFLINE_DRIVE_TRAY
- OFFLINE_DRIVE

- OFFLINE_STORAGE
- OFFLINE_ROBOT
- OFFLINE_ROOT_SWITCH
- OFFLINE_RAIL_CONTROLLER
- OFFLINE_CONTROLLER
- OFFLINE_CAP
- ONLINE_ROOT_SWITCH
- ONLINE_ROBOT
- ONLINE_RAIL_CONTROLLER
- ONLINE_CONTROLLER
- ONLINE_STORAGE
- ONLINE_ROTARY_CAP
- ONLINE_AEM
- DEFAULT_DEV_OFFLINE
- INIT_CAPS
- CAP_MOVE
- TOP_LEVEL_CAP_MOVE
- RESET_ROBOT
- HANDLE_STORAGE_FAILURE
- ENTER
- EJECT
- DEFAULT_CNTL_DEV_ONLINE
- INIT_AEMS
- INIT_AEM
- EVAC_AEM
- AEM_CAP_MOVE
- HNDL_INTERPT_MOVE
- LOX_ONLINE
- FC_PORT_DEVICE_RECOVERY
- AEM_SVCKEY_HANDLER
- SWEEP_AEM
- ALL_CAP_DIAGNOSTIC
- SINGLE_CAP_DIAGNOSTIC
- DRIVE_DIAGNOSTIC
- DEVICE_DIAGNOSTIC
- FEATURE_DIAGNOSTIC
- MOVE_TO_CORNERS

- MOVE_TO_MAGAZINES
- MOVE_IN_RANGE
- SINGLE_LED_DIAGNOSTIC
- ALL_LED_DIAGNOSTIC
- VERSION_DIAGNOSTIC
- MOVE_ALL_CELLS
- MNT_DISMNT_DRIVES_DIAG
- ALL_CAP_MAGAZINE_DIAG
- CUSTOMER_ACCEPTANCE
- CONFIG_BACKUP
- ADD_NODE
- REMOVE_NODE
- AUDIT_MAG
- FAULT_CAP
- FAULT_CAP_CONTROLLER
- FAULT_DRIVE
- FAULT_DRIVE_POWER_SUPPLY
- FAULT_ENCRYPTION_CARD
- FAULT_LOD
- FAULT_LOID
- FAULT_DRIVE_FAN
- LEG_LOD_CODE_LOAD
- FC_PORT_DEVICE_FAULT
- LOER_PRESENCE_CHANGE

JobStateType

- CANCELLED
- COMPLETED_SUCCESS
- COMPLETED_ERROR
- NEEDS_RESOURCES
- RUNNABLE
- DEV_FAILED
- WAITING_FOR_DEVICE_RESPONSE
- WAITING_FOR_SUBJOB
- UNKNOWN

LabelWindowing

- PREPEND_LAST_2_CHARS
- FULL_LABEL
- TRIM_LAST_CHAR
- TRIM_LAST_2_CHARS
- TRIM_FIRST_CHAR
- TRIM_FIRST_2_CHARS
- UNKNOWN

LibraryComplexStateType

- INITIALIZING
- OPERATIVE
- INOPERATIVE
- DEGRADED
- STARTUP
- UNKNOWN

LibraryControllerError

- SOURCE_EMPTY
- DESTINATION_FULL
- NONEXISTANT_SOURCE
- NONEXISTENT_DESTINATION
- NONEXISTENT_CLEANING_CARTRIDGE
- CAP_NOT_FOUND
- CAP_OPEN
- CAP_CLOSED
- CAP_UNLOCKED
- CAP_RESERVED
- CAP_IN_USE
- CAP_IN_USE_BY_OTHER_REQ
- CAP_NOT_ONLINE
- CAP_NOT_OWNED
- CAP_OWNED_BY_ANOTHER
- ELEMENT_IN_USE
- CAP_UNAVAILABLE
- CAP_FAILURE
- CAP_HANDLE_NOT_FOUND

- CAP_POOL_ALREADY_EXISTS
- CAP_POOL_NOT_FOUND
- CAP_POOL_HAS_CAP
- CAP_POOL_HAS_PARTITION
- NO_CAP_MAGAZINE
- DRIVE_NOT_ONLINE
- REQ_NOT_CANCELABLE
- NO_READONLY
- DRIVE_MAINTENANCE
- DRIVE_CONTROLLER_NOT_RESPONDING
- DRIVE_NOT_PRESENT
- MEDIA_MAINTENANCE
- DRIVE_CANT_LOAD_CART
- DRIVE_EMPTY
- MISBUCKLE_ERROR
- EXPIRED_CLEAN_CARTRIDGE
- MOUNT_FAILURE_MEDIA_ERROR
- MEDIA_ERROR
- RESERVE_REQ_TIMED_OUT
- DRIVE_LOADED
- CARTRIDGE_TYPE_INVALID
- ROBOT_FAILURE
- NO_OPERATIONAL_HARDWARE_PATH
- DRIVE_FAILURE
- LIBRARY_NOT_ONLINE
- LIBRARY_ID_INVALID
- MODULE_ID_INVALID
- DEVICE_ID_INVALID
- RAIL_ID_INVALID
- DRIVE_ID_INVALID
- CAP_ID_INVALID
- SUPPORT_BUNDLE_ID_INVALID
- INVALID_PARAMETER
- MV_DRIVE_POOL_PARTITION
- PARTITION_NOT_ONLINE
- LABEL_MISCOMPARE
- INTERFACE_ERROR

- INTERNAL_SOFTWARE_ERROR
- MOUNT_DESTINATION_NOT_A_DRIVE
- MOUNT_SOURCE_IS_NOT_A_CELL
- MOVE_DESTINATION_IS_A_DRIVE
- MOVE_SOURCE_IS_A_DRIVE
- DISMOUNT_SOURCE_IS_NOT_A_DRIVE
- DISMOUNT_DESTINATION_IS_NOT_A_CELL
- LIBRARY_NUMBER_OUT_OF_RANGE
- RAIL_NUMBER_OUT_OF_RANGE
- COLUMN_NUMBER_OUT_OF_RANGE
- SIDE_NUMBER_OUT_OF_RANGE
- ROW_NUMBER_OUT_OF_RANGE
- CELL_TYPE_INVALID
- DEVICE_TYPE_INVALID
- COUNT_IS_NEGATIVE
- STARTING_VALUE_IS_NEGATIVE
- IDENTITY_INVALID
- COMPLEX_INVALID
- LIBRARY_INVALID
- OLD_SEED_INVALID
- WWN_SEED_INVALID
- DELETE_PARTITION_FAILED_CELLS_PRESENT
- LIST_EMPTY
- SET_EMPTY
- MEDIA_TYPE_EMPTY
- DOMAIN_CODE_EMPTY
- BASE_PART_NUMBER_EMPTY
- BASE_PART_REV_EMPTY
- CODE_VERSION_EMPTY
- FIRMWARE_TYPE_INVALID
- DEVICE_STATE_INVALID
- CONTROL_STATE_INVALID
- ANNOUNCE_MESSAGE_INVALID
- STATE_INVALID
- NAME_INVALID
- LED_LOCATE_NOT_VALID
- DEVICE_NOT_RESETTABLE

- DEVICE_NULL
- DESTINATION_ID_INVALID
- NEGATIVE_VALUE
- ZERO_VALUE
- REQUEST_NOT_FOUND
- REQUEST_COMMAND_INVALID
- PARTITIONS_DO_NOT_MATCH
- PARTITION_DOES_NOT_EXIST
- SOURCE_PARTITION_DOES_NOT_EXIST
- DESTINATION_PARTITION_DOES_NOT_EXIST
- FILENAME_MISSING
- INVALID_ROBOT_ID
- VOLSER_MISSING
- PARTITION_CELLS_PRESENT
- PROTOCOL_INVALID
- IPADDRESS_EMPTY
- EMAIL_ADDRESS_EMPTY
- LOCALE_EMPTY
- PORT_EMPTY
- USER_ID_EMPTY
- PASSWORD_EMPTY
- PATH_EMPTY
- DESTINATION_NULL
- COMMUNITY_STRING_EMPTY
- USER_NAME_EMPTY
- TRAP_LEVELS_EMPTY
- AUTHENTICATION_PHRASE_EMPTY
- PRIVACY_PHRASE_EMPTY
- ENGINE_ID_EMPTY
- PROTOCOL_VERSION_INVALID
- AUTHENTICATION_TYPE_INVALID
- PRIVACY_TYPE_INVALID
- USER_EXPIRED_OR_NOT_KNOWN
- NOT_AUTHORIZED
- LIST_NULL
- INCOMPATIBLE_MEDIA
- DRIVE_TYPE_MISSING

- MEDIA_TYPE_MISSING
- NON_MEDIA_VALIDATION_DRIVE
- NO_CARTRIDGE_FOUND_FOR_VOLSER
- INCORRECT_NUMBER_OF_PARAMETERS
- PARAMETER_NOT_FOUND
- INVALID_DIAGNOSTIC_TEST
- ISCRUPTIVE_DIAGNOSTIC_ON_ONLINE_LIBRARY
- INVALID_DIAGNOSTIC_PARAMETER
- FAN_ASSEMBLY_OFFLINE_BLOCKED
- CONTROLLER_OFFLINE_BLOCKED
- ROBOT_CONTROLLER_OFFLINE_BLOCKED
- DRIVE_TRAY_CONTROLLER_OFFLINE_BLOCKED
- ROOT_SWITCH_OFFLINE_BLOCKED
- DRIVE_SWITCH_OFFLINE_BLOCKED
- STORAGE_OFFLINE_BLOCKED
- RAIL_CONTROLLER_OFFLINE_BLOCKED
- VIDEO_OFFLINE_BLOCKED
- ACCESS_CONTROLLER_MODULE_OFFLINE_BLOCKED
- DC_CONVERTER_OFFLINE_BLOCKED
- PUZ_OFFLINE_BLOCKED
- ROBOT_OFFLINE_BLOCKED
- PDU_OFFLINE_BLOCKED
- POWER_SUPPLY_OFFLINE_BLOCKED
- DEVICE_OUT_OF_SERVICE
- LIBRARY_RANGE_INCORRECT
- RAIL_RANGE_INCORRECT
- COLUMN_RANGE_INCORRECT
- SIDE_RANGE_INCORRECT
- ROW_RANGE_INCORRECT
- UPSIDE_DOWN_CARTRIDGE
- DRIVE_OFFLINE_BLOCKED
- PARTITION_ALREADY_EXISTS
- INPUT_STRING_TOO_LONG
- INPUT_SCI_DEST_INVALID
- MOUNT_CARTRIDGE_IS_UNREADABLE
- NO_AEMS_PRESENT
- PARTITION_ACTIVE

- NOT_A_PDU
- DRIVE_NOT_MOUNTED
- MAX_PARTITIONS_EXCEEDED
- LIBRARY_INOPERATIVE
- LIBRARY_OFFLINE
- UNKNOWN
- PARTITIONING_NOT_ENABLED
- MAXIMUM_ALLOWED_REQUESTS_EXCEEDED
- LIBRARY_BUSY
- REQ_TIME_OUT
- TIME_OUT_NEGATIVE
- LIBRARY_ROLE_INVALID
- SCSI_ALLOWED_PARTITION_NOT_SCI_ACCESSIBLE
- NO_OPERATIONAL_ROBOTS
- JOB_TIMEOUT_ERROR
- MULTIPLE_CARTS_FOUND_FOR_VOLSER

LibraryProductionState

- MANUFACTURE
- PW_CHANGE_NEEDED
- INSTALLING
- HANDOFF
- PRODUCTION

LibraryRole

- ACSOperator
- ACSUser
- ACSAdmin
- ACSService
- ACSAdvSvc
- ACSExcalation
- ACSViewer
- ACSInstall

LibraryStateType

- INITIALIZING
- OPERATIVE

- INOPERATIVE
- DEGRADED
- POWERED_OFF
- UNKNOWN

LogLevel

- OFF
- SEVERE
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST
- INHERITED

MediumType

- UNKNOWN
- DATA
- UNSUPPORTED
- CLEANING

ModuleType

- BASE
- DRIVE
- CARTRIDGE
- ACCESS
- PARKING
- UNKNOWN

NetworkSettingsType

- CUSTOMER
- OKM
- SERVICE
- UNKNOWN

PartitionStateType

- INITIALIZING

- OPERATIVE
- INOPERATIVE
- DEGRADED
- POWERED_OFF
- UNKNOWN

RequestErrorType

- GENERAL
- UNKNOWN

RequestSource

- INTERNAL
- GUI
- SCI
- SCSI
- ASR
- UNKNOWN

RequestStatus

- SUBMITTED
- ACTIVE
- COMPLETE
- CANCELLED
- FAILED
- TIMEDOUT
- UNKNOWN

ResourceName

- UNKNOWN
- SOURCE_CELL
- DESTINATION_CELL
- SWAP_CELL
- FETCH_PUT_CELL
- ROBOT
- ROBOT_1
- ROBOT_2
- ROBOT_CELL

- DRIVE
- SOURCE_DRIVE
- DEST_DRIVE
- RAILSEGMENT
- LOER_ONE
- LOER_TWO
- LOS_ONE
- LOS_TWO
- LOB_RB_ONE
- LOB_RB_TWO
- CONTROL_LOC
- REDUN_LOC
- CARD_CAGE
- LOD_CARD
- LOID_CARD
- LOB_CARD
- DEVICE
- SECONDARY_ROBOT
- MEDIA_VAL_CELL
- MEDIA_VAL_DRIVE
- LOES_CARD

ResourceState

- UNKNOWN
- NEEDED
- ALLOCATED
- COMPLETE

ResourceType

- UNKNOWN
- DEVICE
- CELL
- RAIL_SEGMENT

RobotHardwareStatusCode

- SUCCESS
- BAD-USAGE

- APPL-NOT-READY
- APPL-TASK-NOT-READY
- APPL-TASK-FAILURE
- TARGET-EXCEEDED-MAX-ALLOWABLE-BARS
- TARGET-NOT-RECOGNIZED
- TARGET-OFFSET-INVALID
- TARGET-LAST-RESULT-CODE
- TARGET-FAILED-TO-CALIBRATE-SCANNER
- SCAN-FAULT-OPEN-FAILURE
- SCAN-FAULT-POWER-UP-FAILURE
- SCAN-FAULT-INITIALIZATION-FAILURE
- SCAN-APP-RECEIVED-NO-MESSAGES
- SCAN-APP-TO-DRIVER-READ-TIMEOUT
- SCAN-APP-TO-DRIVER-WRITE-TIMEOUT
- SCAN-APP-RECEIVED-LLF-NAK-FN
- SCAN-APP-RECEIVED-LLF-NAK-CHKSUM
- SCAN-APP-RECEIVED-LLF-BUSY
- SCAN-APP-RECEIVED-PACKET-WITH-BAD-CHKSUM
- SCAN-APP-RECEIVED-NR
- SCAN-APP-BAD-STATUS
- SCAN-APP-PACKET-SIZE-TOO-LARGE
- SCAN-APP-ASCII-TO-INT-PARSE-FAILURE
- SCAN-APP-LOOKING-FOR-TARGET-GOT-BARCODE
- SCAN-APP-RECEIVED-LINE-STATUS-ERROR-INDICATION
- SCAN-APP-TARGETING-DATA-TOO-SHORT
- SCAN-FIRMWARE-DOWNLOAD-FAILURE
- SCAN-LABEL-CHARACTER-TRANSLATED
- SCAN-LAST-RESULT-CODE
- SRV-MECH-STALLED
- SRV-MECH-STALLED-ON-INIT
- SRV-MECH-OUTSIDE-STOPLOCK
- SRV-ISR-LOGICAL-FAILURE
- ERR-SRV-UNKNOWN-REQUEST-TYPE
- dont-use-ERR-SRV-UNEXPECTED-SYS-ERROR-RET
- ERR-SRV-BAD-CHK-MOVE-CALC
- ERR-SRV-DEST-OUTSIDE-OPER-RANGE
- ERR-SRV-ILLEGAL-PROFILE-TYPE

- ERR-SRV-OVERCURRENT
- ERR-SRV-EXCESSIVE-POSITION-ERROR
- ERR-SRV-TACH-PHASE-ERROR
- ERR-SRV-CANT-START-NOT-IN-STOPLOCK
- ERR-SRV-ISR-REENTERED
- ERR-SRV-SATURATION-CURRENT-REQUESTED-TOO-LONG
- ERR-SRV-MECH-DROPPED-OUT-OF-STOPLOCK
- ERR-SRV-MECH-FAILED-TO-SETTLE-INTO-STOPLOCK
- ERR-SRV-OPERATING-RANGE-OUT-OF-SPEC
- ERR-SRV-INVALID-THETA-Z-RANGE-COMBO
- ERR-SRV-REDEFINED-LIB-CONFIG
- ERR-SRV-BAD-MECH-ID-IN-ISR
- ERR-SRV-ILLEGAL-REQUEST-OPTION
- ERR-SRV-FAILED-TO-ENCOUNTER-CARTRIDGE
- ERR-SRV-FAILED-TO-DISENGAGE-CARTRIDGE
- ERR-SRV-FAILED-TO-SEAT-CARTRIDGE
- ERR-SRV-FAILED-TO-UNSEAT-CARTRIDGE
- ERR-SRV-REQUEST-ALREADY-ACTIVE-AGAINST-MECHANISM
- ERR-SRV-CANT-MOVE-ARM-HAND-IS-ACTIVE
- ERR-SRV-CANT-MOVE-HAND-ARM-IS-ACTIVE
- ERR-SRV-UNEXPECTED-RESPONSE
- ERR-SRV-CANT-GET-WITH-HAND-FULL
- ERR-SRV-CANT-PUT-WITH-HAND-EMPTY
- ERR-SRV-MOVE-ABORTED
- ERR-SRV-HAND-NOT-SAFE-HAND-IS-INOPERATIVE
- ERR-SRV-HAND-NOT-SAFE-REACH-NOT-RETRACTED
- ERR-SRV-HAND-NOT-SAFE-CARTRIDGE-IS-UNSEATED-IN-GRIP
- ERR-SRV-MECHANISM-NOT-INITIALIZED
- ERR-SRV-MECHANISM-SHUTDOWN
- ERR-SRV-MECHANISM-NOT-OPERATIONAL
- ERR-SRV-USER-REQ-THETA-MOVE-FOR-SCAN
- ERR-SRV-CANT-CLEAR-AMP-ENABLE
- ERR-SRV-SATURATION-CURRENT-READ-TOO-LONG
- ERR-SRV-MINIMUM-INIT-MOVE-NOT-DETECTED
- ERR-SRV-REACH-SAFE-SENSOR-FAIL
- ERR-SRV-REACH-GRIP-OVERCURRENT
- ERR-SRV-AMP-ENABLE-FAIL

- ERR-SRV-FAILED-STALL
- ERR-SRV-FAILED-STALL-OBSTRUCTED
- ERR-SRV-DEST-OUTSIDE-OPER-RANGE-ADJUSTED
- ERR-SRV-NOT-RESPONSE-DISCARDED
- ERR-SRV-CANT-FIND-REACH-DEPTH
- ERR-SRV-POWER-LOW-ERROR
- ERR-SRV-REQUEST-QUEUED-TIMEOUT
- ERR-SRV-REQUEST-ACTIVE-TIMEOUT
- ERR-SRV-BAD-MECH-ID-IN-COORD
- ERR-SRV-SYS-MSG-ALLOC-FAIL
- ERR-SRV-SYS-MSG-SEND-FAIL
- ERR-SRV-SYS-MSG-RECV-FAIL
- ERR-SRV-SYS-MSG-BAD-SIZE
- ERR-SRV-SYS-MSG-GET-CONTENT-FAIL
- ERR-SRV-SYS-MSG-SET-CONTENT-FAIL
- ERR-SRV-SYS-MSG-RELEASE-FAIL
- ERR-SRV-HAND-NOT-SAFE
- ERR-SRV-HAND-INIT-FAIL-NOT-EMPTY
- ERR-SRV-COORD-SEND-MECH-REQUEST-FAILED
- ERR-SRV-HALL-ERROR
- ERR-SRV-HDW-OVER-CURRENT-ERROR
- ERR-SRV-HDW-UNKNOWN-ERROR
- ERR-SRV-EXCESSIVE-MOTOR-HEATING
- ERR-SRV-SAT-CURRENT-REQ-TOO-LONG-STALL-MIN-NOT-REACHED
- ERR-SRV-EXCESSIVE-TRACK-STRACK-RELATIVE-ERROR
- ERR-SRV-DOOR-OPEN-ERROR
- ERR-SRV-ISR-STARTUP-FAILED
- ERR-SRV-BAD-MECH-ID-AT-MECH-LAYER
- ERR-SRV-COACTIVE-QUEUED-FAILURE
- ERR-SRV-COACTIVE-QUEUED-TIMEOUT
- ERR-SRV-FAILED-TO-REACH-STALL-POSITION
- ERR-SRV-BAD-MECH-ID-AT-USER
- ERR-SRV-UNDETECTED-AMP-DISABLE
- ERR-SRV-COORD-SEQUENCING-FAILED
- ERR-SRV-FAILED-ZERO-TACH-ON-STALL
- ERR-SRV-FAILED-MECH-LIMIT
- CMO-FAILED-CARTESIAN-LOOKUP-AUDIT

- CMO-FAILED-CARTESIAN-LOOKUP-FETCH
- CMO-FAILED-CARTESIAN-LOOKUP-PUT
- CMO-FAILED-CARTESIAN-LOOKUP-TARGET
- CMO-FAILED-CARTESIAN-LOOKUP-MOVE
- CMO-COULD-NOT-STORE-TARGET-CALIBRATION
- CMO-REACH-NOT-SAFE-DETECTED
- CMO-HAND-EMPTY-DETECTED
- CMO-HAND-FULL-DETECTED
- CMO-FAILED-TARGET-CALIBRATION
- CMO-FETCH-RETRY-PERFORMED
- CMO-PUT-RETRY-PERFORMED
- CMO-CART-LABEL-MISCOMPARE
- CMO-CELL-FULL-DETECTED
- CMO-CELL-EMPTY-DETECTED
- CMO-END-OF-RAIL-ID-FAILURE
- CMO-INIT-FAILURE
- CMO-FAILED-CARTESIAN-LOOKUP-PROX
- CMO-FAILED-PROX-READ
- CMO-MOVE-RETRY-PERFORMED
- CMO-INCONSISTENT-SUCCESS-ON-FETCH
- CMO-INCONSISTENT-SUCCESS-ON-PUT
- CMO-CELL-SCAN-USED-FOR-AUDIT
- CMO-DEPRECATED-POSITION-USED-TO-TARGET
- CMO-USED-INITIAL-TARGETED-LOCATION
- CMO-AUDIT-LABEL-MIN-LENGTH-NOT-MET
- CMO-FAILED-UNSET-TARGET-CALIBRATION
- CMO-RECOVER-FOR-FETCH-PUTBACK-NOT-ATTEMPTED
- CMO-RECOVER-FOR-FETCH-SRV-RECOVERED-CART
- CMO-FAILED-HANDBOT-RANGE-GET
- CMO-CALIBRATION-RETRY-PERFORMED
- CMO-ROBOT-Z-RANGE-IS-SHORT
- CMO-ROBOT-TRACK-RANGE-IS-SHORT
- CMO-ROBOT-WRIST-RANGE-IS-SHORT
- CMO-INVALID-ADDRESS
- CMO-SCANNER-HW-NOT-SUPPORTED
- CMO-SCANNER-UNRECOGNIZED-HW-VERSION
- CMO-CART-NO-LABEL-FOUND

- CMO-CRIMSON-FRAMELABEL-UNRECOGNIZED-LABEL
- CMO-CRIMSON-EMPTY-FRAMELABEL-CELL
- CMO-CRIMSON-FRAMELABEL-NO-LABEL
- CMO-CRIMSON-FRAMELABEL-PROBLEM
- CMO-END-OF-RAIL-ID-FAILURE-CRIMSON
- CMO-CRIMSON-FRAMELABEL-WARNING
- CMO-HAND-NOT-AT-LOCATION
- CMO-CAP-MAGAZINE-NO-INSTALLED
- CMO-ARM-NOT-OPERATIONAL-DETECTED
- CMO-END-OF-RAIL-SET-REV-FAILURE-CRIMSON
- CMO-CARTRIDGE-DROPPED-OFF-AT-INIT
- ERROR-OPENING-CALIBDATA-FILE
- ERROR-OPENING-CFGLOCATIONS-FILE
- ERROR-OPENING-MULTI-ROW-SCAN-DISABLE-FILE
- ERROR-FAILED-MECH-LIMIT-OPERATION
- DIRCT-FAULT-BAD-REQU
- DIRCT-FAULT-ZERO-REFERENCE-FAILURE
- DIRCT-FAULT-BAD-END-OF-RAIL-INIT
- DIRCT-FAULT-ROBOT-SET-DIRECTION-FAILURE
- DIRCT-FAULT-SET-CONFIG-MAP-FAILURE
- DIRCT-FAULT-SEND-MOVE-MAP-FAILURE
- DIRCT-FAULT-INVALID-Z-HEIGHT
- DIRCT-FAULT-REBUILD-CONFIG-MAP-FAILURE
- DIRCT-FAULT-GET-CONFIG-FAILURE
- DIRCT-FAULT-READ-ALL-CRIMSON-LABELS
- DIRCT-FAULT-SET-CRIMSON-CONFIG-MAP-FAILURE
- DIRCT-FAULT-BAD-END-OF-RAIL-INIT-CRIMSON
- DIRCT-FAULT-MODULE-DEFINITION-MAP-FAILURE
- DIRCT-FAULT-GET-LOCATIONS-FAILURE
- DIRCT-FAULT-INIT-HAND-CRIMSON
- DIRCT-WARNING-TRACK-STRACK-RANGE-MISMATCH-CRIMSON
- DIRCT-FAULT-TRACK-STRACK-RANGE-MISMATCH-CRIMSON
- DIRCT-FAULT-INVALID-ADDRESS

RobotHomeEnd

- LEFT-ROBOT
- RIGHT-ROBOT

RobotSelector

- ALL
- LEFT_ROBOT
- RIGHT_ROBOT

RobotStatusCode

- SUCCESS
- SOAP-ERROR
- COMM-FAILURE
- INVALID-REQUEST
- TEST-EVENT
- CANT-MOVE-ON-RAIL
- CANT-FIND-TARGET
- NEED-TO-BE-INOP
- CART-STUCK
- SOURCE-LOCATION-EMPTY
- CELL-EMPTY
- REACH-NOT-SAFE
- LABEL-MISCOMPARE
- LOC-UNUSABLE
- CANT-MOVE-WRIST
- DESTINATION-FULL-OBSTRUCTED
- VISION-INOP
- CANT-BE-OPERATIVE
- HIT-OBSTRUCTION
- NEEDS-RESET
- NO-CART-IN-HAND
- CART-IN-HAND
- CELL-FULL
- INVALID-CONFIGURATION-LABELS
- DROPPED-OFF-CART-AT-INIT
- UNRESPONSIVE
- DEVICE-WAS-RESET
- MOVE_OUT_OF_WAY_FAILURE

ScanType

- NO_VALIDATION

- BASIC_VERIFY
- COMPLETE_VERIFY_BOT
- COMPLETE_VERIFY_RESUME
- COMPLETE_VERIFY_PLUS_DIV_BOT
- COMPLETE_VERIFY_PLUS_DIV_RESUME
- STANDARD_VERIFY
- REBUILD_MIR
- STOP_VALIDATION

ScsiHostState

- LOGGED_IN
- LOGGED_OUT
- UNKNOWN

SensorType

- FAN
- HOT_SWAP_CONTROLLER
- NETWORK_SWITCH_PORT
- PDU_ENERGY_MONITOR
- TEMPERATURE
- CAP
- ROBOT
- NO_SENSOR
- UNKNOWN

ServiceIndicatorName

- OKTOREMOVE
- SERVICEACTIONREQUIRE
- OK
- CANACTIVE
- LANAACTIVE
- LANBACTIVE
- WAIT
- LIBRARYACTIVE
- SERVICEREQUIRED
- LOCATE
- ENTER

- UNLOCKED

SeviceIndicatorState

- UNLIT
- LIT
- SLOWBLINK
- FASTBLINK

SupportBundleOriginator

- FAULT_SUBSYSTEM — Support bundle automatically generated when fault was detected.
- SCI — An explicit user action generated the support bundle.
- GUI — An explicit user action generated the support bundle.
- ASR — A service initiated support bundle from the Service Delivery Platform (SDP2)
- UNKNOWN

SupportBundleState

- IN_PROGRESS
- COMPLETE
- FAIL
- CANCELLED
- DELETED
- UNKNOWN

SystemReportType

- ASR
- GENERAL
- ROBOT
- DRIVE
- PIC
- STORAGE
- FEATURE
- UNKNOWN
- FC_PORT

TopLevelDeviceStateType

- PRESENCE_UNKNOWN

- DETECTED
- OPERATIVE
- INOPERATIVE
- DEGRADED
- REMOVED
- UNKNOWN
- NOT_LICENSED
- INITIALIZING

A

Implementation Examples

This chapter provides examples of how the SCI interface can be implemented in various coding languages.

- Python
- C/C++

Python

This requires a "pip install suds" to get the python suds 0.4 package.

```
#!/bin/env python

from suds.client import Client
from suds.bindings import binding
from suds.wsse import Security, UsernameToken

library = "https://library.company.com/WebService/1.0.0"

# This could be cached to a local file for efficiency
wsdl = "http://library.company.com/WebService/1.0.0?WSDL"

# Use SOAP 1.2
binding.envns = ('SOAP-ENV', 'http://www.w3.org/2003/05/soap-envelope')
proxy = Client(url=wsdl, location=library,
               headers={'Content-Type': 'application/soap+xml'})
security = Security()
token = UsernameToken('admin', 'password1')
security.tokens.append(token)
proxy.set_options(wsse=security)

response = proxy.service.ping()
print "Ping responded with: "
print response, "\n"

response = proxy.service.getRobots(libraryId = 1)
print "Robots: "
print response, "\n"

response = proxy.service.getDevice(deviceId = response[0].parentDeviceId)
print "Rail: "
print response, "\n"
```

C/C++

This is compiled with Genivia gSOAP. The recommended version is 2.8.17r. gSOAP is copyrighted by Robert A. van Engelen, Genivia, Inc.

```
#include <iostream>
#include "soapWebServicePortBindingProxy.h"
#include "WebServicePortBinding.nsmap"
#include "plugin/wsseapi.h"

using namespace std;
static const char *library = "https://library.company.com/WebService/1.0.0";

int main(int argc, char **argv)
{
   WebServicePortBindingProxy proxy;

   ns1__getLibraryComplex getLibraryComplex;
   ns1__getLibraryComplexResponse getLibraryComplexResp;

   if (soap_ssl_client_context(proxy.soap, SOAP_SSL_NO_AUTHENTICATION, NULL, NULL,
   NULL, NULL, NULL)) {
      soap_print_fault(proxy.soap, stderr);
   }

   soap_wsse_add_Security(proxy.soap);
   soap_wsse_add_UsernameTokenText(proxy.soap, "Id", "admin", "password1");

   if(proxy.getLibraryComplex(library, NULL, &getLibraryComplex,
   &getLibraryComplexResp)) {
      proxy.soap_stream_fault(cerr);
      exit(-1);
   }
   cout << "Name: " << *(getLibraryComplexResp.libraryComplex->name) << endl;
   cout << "Ready: " << (getLibraryComplexResp.libraryComplex->ready ? "TRUE" : "FALSE") << endl;
   cout << "Libraries: " << getLibraryComplexResp.libraryComplex->counts->libraryCount << endl;
   cout << "Partitions: " << getLibraryComplexResp.libraryComplex->counts->partitionCount << endl;
   cout << "Devices: " << getLibraryComplexResp.libraryComplex->counts->deviceCount << endl;
   cout << "Drives: " << getLibraryComplexResp.libraryComplex->counts->driveCount << endl;
   cout << "Drive Bays: " << getLibraryComplexResp.libraryComplex->counts->driveBayCount << endl;
   cout << "Cells: " << getLibraryComplexResp.libraryComplex->counts->cellCount << endl;
   cout << "Robots: " << getLibraryComplexResp.libraryComplex->counts->robotCount << endl;
   soap_end(proxy.soap);
}
```

B

Secure Development Guide

This appendix provides an overview of common security risks for developers using the SL4000 web services API called StorageTek Library Control Interface (SCI), and information on how to address those risks.

SCI is a Web Services Definition Language (WSDL) based API that uses XML for data transmission and HTTPS for transport. SCI is bidirectional. For inbound SCI, the library is a server that responds to requests from a client program. Inbound SCI defines about 300 methods used to operate, configure, or monitor the library. Outbound SCI defines a set of about 25 methods that the library uses to send notifications. For outbound SCI, the library is the client for an external server.

Both the inbound and outbound interfaces provide similar security functionality:

- Transport layer security with HTTPS using TLSv1.1 or TLSv1.2 protocols
- Authentication with a username password token
- Authorization for role based access control on inbound methods

Generating Code From WSDL Specifications

Processing tools such as wsgen (Java code), gsoap (C and C++) or WSDL.exe (.net) can generate both client-side and server-side code from the WSDL file and its associated type files (XSD files). While you can manually construct the XML documents sent by a client of the inbound SCI interface or manually parse the XML documents sent by outbound SCI, Oracle recommends using a processing tool.

For the client-side code, the processing tool outputs a set of language-specific classes and callable methods that can directly be invoked by code that uses the interface. The terms “class” and “method” are used generically here. For server-side code, the processing tool outputs code that can be called by a client, in this case the library. Unlike the client-side code, the server-side code is incomplete and will typically have a line saying “add your code here” in each generated method. The developer must add bodies to the generated methods.

Transport Layer Security

Transport layer security provides privacy during data transmission by encrypting the message when the server sends it and then decrypting it when the client receives it. Both the client and the server have the contents of the message in clear text. The library uses the HTTPS protocol to provide transport layer security.

During initial installation, the library uses a default certificate (a pre-defined, self-signed x509 certificate) for HTTPS. During the library "hand off" process, you can

choose to replace the default certificate with a library-specific, self-signed certificate or provide a third-party signed certificate. See the following for more information:

- "Manage the Library's SSL/TLS Certificate for HTTPS" in the *SL4000 Library Guide* for instructions on how to update the library certificate.
- "Certificates for HTTPS Interfaces" and "Handing-Off the Library to the Customer" in the *SL4000 Security Guide* for more information about the certificates.

For the inbound SCI interface, HTTPS is required. The library implements authentication using username password tokens. The user id and password appear in clear text, therefore HTTPS is required to avoid an eavesdropper on the network from reading the id and password from the messages in flight.

For the outbound SCI interface, HTTPS is optional. Oracle recommends using authentication and HTTPS on the outbound interface, however not all environments may require authentication. Creating an outbound SCI server (remember, the library is the client) without authentication does open the server up to numerous attacks.

Supported cipher suites are:

- tls1_1:ECDHE-RSA-AES128-SHA
- tls1_1:DHE-RSA-AES128-SHA
- tls1_1:ECDHE-RSA-DES-CBC3-SHA
- tls1_1:EDH-RSA-DES-CBC3-SHA
- tls1_1:AES128-SHA
- tls1_1:DES-CBC3-SHA
- tls1_2:ECDHE-RSA-AES128-GCM-SHA256
- tls1_2:ECDHE-RSA-AES128-SHA256
- tls1_2:ECDHE-RSA-AES128-SHA
- tls1_2:DHE-RSA-AES128-GCM-SHA256
- tls1_2:DHE-RSA-AES128-SHA256
- tls1_2:DHE-RSA-AES128-SHA
- tls1_2:ECDHE-RSA-DES-CBC3-SHA
- tls1_2:EDH-RSA-DES-CBC3-SHA
- tls1_2:AES128-GCM-SHA256
- tls1_2:AES128-SHA256
- tls1_2:AES128-SHA
- tls1_2:DES-CBC3-SHA

Inbound and Outbound Authentication

Both the inbound and outbound SCI interfaces use a username password token for authentication. Authentication is required for inbound and optional for outbound.

For inbound commands, the client must add a SOAP header to every message sent to the library to provide the username password token in clear text. Therefore, the client program must have access to and securely manage these credentials. The most secure method is to not store the credentials, but to have the client program prompt the user when necessary. A client program should avoid taking these values as command line

arguments because system monitoring tools may display command line arguments. If the client program must store the credentials, do so in a secure manner, such as using a java wallet.

If you choose to implement authentication for the outbound interface, the server must extract the username and password from the SOAP header and use them to perform authentication.

The details of how to insert these values into the message or extract them from the message are specific to the programming language used on the client-side and the WSDL processor used to generate the stubs. The following is a sample inbound request:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:v1="http://v1_0_
0.webservice.librarycontroller.summit.acs.tape.oracle/">
  <soap:Header>
    <wsse:Security
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <wsse:UsernameToken wsu:Id="UsernameToken-98F0D229E2F29CEF1514779315276651">
        <wsse:Username>username</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
         PasswordText">password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <v1:ping/>
  </soap:Body>
</soap:Envelope>
```

Authorization by Role

All inbound SCI methods require authorization. Only certain roles can execute some methods. The roles are abbreviated as:

- S1 (service), S2 (advanced service), S3 (escalation)
- C1 (operator), C2 (user), C3 (admin)
- I (installer)
- All (viewer, all service roles, all customer roles, and installer)

The services (S) and customer (C) roles have increasing privileges. The method descriptions in Chapter 1, "Library Inbound Methods" list the lowest role that can execute the method. Higher privileged roles can also execute the command. For example, a method labeled "C2,S2" can be executed by C2 (user), C3 (admin), S2 (advanced service), or S3 (escalation).

A customer-created SCI program can only use the customer roles of Viewer, Operator, User, or Administrator. Developers should examine the method they intend to use and choose the lowest of the four available roles. Then library Administrator can create a library user with that role for the client program and then provide the id and password to the client program. For information on creating a library user through the GUI, see “Add, Modify, or Delete a User” in the *SL4000 Library Guide*.

Outbound SCI is one of several forms of “notification” provided by the library. Use the GUI to configure the outbound SCI clients (see "Configure the Library to Send Outbound SCI" in the *SL4000 Library Guide*). If the outbound SCI server will perform authentication, you must select HTTPS and provide an id and password. You cannot use authentication credentials if you select HTTP.