Venus Documentation

Release 0.1

TChapman

Feb 23, 2024

CONTENTS

Venus is a coding language wrapper used in the coding of Hamilton Microlab STARs and similar devices. This is the consolidated unofficial documentation for Venus 4 as well as its libraries, compiled from official sources as well as personal troubleshooting.

This is not an official Hamilton resource

Note: This project is under active development. If any issues or mistakes are found, please comment or contact me directly at tarunchapman@hotmail.com. General place to download libraries: https://github.com/theonetruenerd/VenusPackages

---

CHAPTER

ONE

VENUS

This section aims to cover the *functions* that Venus can run. Documentation on more general principles of Venus, including things like the liquid editor and labware editor, will be found separately.

NB: This section is probably coming last as out of all of the Venus resources it has the best documentation.

The Venus functions are split into a few main categories:

- General Steps
- ML_STAR
- Custom Dialog Steps

1.1 General Steps

The general steps are steps which are not instrument-specific (and thus not hardware-related), and aren't involved in custom dialog creation. They are the following:

- `Comment()`
- `Assignment()`
- `Assignment with Calculation()`
- `Loop()`
- `Loop: Break()`
- `If, Else()`
- `Array: Declare / Set Size()`
- `Array: Set At()`
- `Array: Get At()`
- `Array: Get Size()`
- `Array: Copy()`
- `Sequence: Get Current Position()`
- `Sequence: Set Current Position()`
- `Sequence: Get End Position()`
- `Sequence: Set End Position()`
- `Adjust Sequences()`

- File: Open()
- File: Read()
- File: Write()
- File: Set Position()
- File: Close()
- Timer: Start()
- Timer: Wait For()
- Timer: Read Elapsed Time()
- Timer: Restart()
- User Input()
- User Output()
- Shell()
- Set Event()
- Wait for Event()
- Return()
- Abort()
- Error Handling by the User()
- Begin Parallel()
- End Parallel()

1.2 ML_STAR

The ML_STAR steps are steps which relate to the ML_STAR and interact with its hardware specifically, doing things such as pipetting, moving channels, etc. They are split into two categories; the four Smart Steps and the remaining Single Steps. They are as follows:

- 1000uL Channel Aspirate()
- 1000uL Channel Dispense()
- iSWAP Transport()
- 1000uL Channel CO-RE Grip Transport()
- 1000uL Channel Tip Pick Up (Single Step)()
- 1000uL Channel Aspirate (Single Step)()
- 1000uL Channel Dispense (Single Step)()
- 1000uL Channel Dispense on the Fly (Single Step)()
- 1000uL Channel Tip Eject (Single Step)()
- 1000uL Channel Get Last Liquid Level (Single Step)()
- 1000uL Channel Aspirate 2nd Phase (Single Step)()
- Initialize (Single Step)()

- Lock/Unlock Front Cover (Single Step)()
- iSWAP Get Plate (Single Step)()
- iSWAP Place Plate (Single Step)()
- iSWAP Move Plate (Single Step)()
- iSWAP Open Gripper (Single Step)()
- iSWAP Close Gripper (Single Step)()
- iSWAP Get First Plate Position (Single Step)()
- iSWAP Park (Single Step)()
- 1000uL Channel CO-RE Grip Get Plate (Single Step)()
- 1000uL Channel CO-RE Grip Place Plate (Single Step)()
- 1000uL Channel CO-RE Grip Move Plate (Single Step)()
- 1000uL Channel Move To Position (Single Step)()
- Wait for TADM Upload (Single Step)()

1.3 Custom Dialog Steps

The custom dialog steps only has a single step called custom dialog, which helps in the creation of more personalised versions of the standard User Input and User Output dialogs.

- Custom Dialog()

CHAPTER

TWO

ERROR CODES & DEBUGGING

I will try compile as many different error codes and their explanations here, as well as potential causes and solutions to deal with them as and when they pop up. If there are any you wish to add, please message me on my discord "theonetruenerd" or email me at "tarunchapman@hotmail.com". Notably, the error codes that show up in the trace look slightly different to the ones presented here. If the trace error code looks like this: (0x12 - 0x3 - 0x45) [which corresponds to (MinorID - MajorID - SpecificErrorID)], then the translated error code will look like this: 0xa3120045. The error code should always be 10 digits long, with zeroes padding the bit between the 2 and 45.

The Major ID is usually assigned in the library which is causing the error; for example in HSLUtilLib2 there is a function called "Error" in which there is a static const variable called MajorID. The minor ID corresponds to the general type of error; in HSLUtilLib2 "0x01" is used to refer to a general runtime error.

At least I think? I've found a bit of inconsistency so will keep digging.

- Runtime Errors
- Syntax Errors
- Unsorted Errors

2.1 Major IDs that have been identified

NB: The codes are in hexadecimal so (for example) 31 = 1F

- 0: None
- 2: HxCfgFil
- 3: HxTrcFil
- 4: HxParams
- 5: HxLabwr
- 6: HxPlumb
- 7: HxM4Comm
- 8: HxM4Inst
- 9: HxM4PPtr
- 10: HxPm4Met
- 11: HxPm4Cfg
- 12: HxPm4Run
- 13: HxPhnx

- 14: HxMovPrb
- 15: HxSsInst
- 16: HxDeckEd
- 17: HxRun
- 18: HxReg
- 19: HxM4MetsDat
- 20: HxM4MetsCfg
- 21: HxP96P4Wz
- 22: HxP96M4MetsRun
- 23: HxM4PPtrCfg
- 25: HxM4LoadCfg
- 26: HxM4PrbInLabwr
- 27: HxCfgEd
- 28: HxPm4Wz
- 29: HxWrkFil
- 30: HxM4PPtrCfg2
- 31: HxM4CComm
- 32: HxM4PPtrPars
- 33: HxM4Transfer
- 34: HxHslParser
- 35: HxHslExecutor
- 36: HxHslRctl
- 37: HxFdxProtocol
- 38: HxSerial
- 39: HxTextMet
- 40: HxGruCommand
- 41: HxGruInst
- 42: HxCommand
- 43: HxSwapCommand
- 44: HxProSim
- 45: HxCompCommand
- 46: HxMetEdCompCmd
- 47: HxPrbInLw
- 48: HxMetEd
- 49: HxCommunication
- 50: HxProtocol

- 51: HxUsbComm
- 52: HxGruCompCmd
- 53: HxSampleTracker
- 54: HxGruLiquid
- 55: HxSecurity
- 56: HxGruLiquidEditor
- 57: HxStarMaintAndVer
- 58: HxSoftMaxPro
- 59: HxCytomat
- 60: HxM4Command
- 61: HxCarousel
- 62: HxMetChaining
- 63: HxAuditTrail
- 64: HxTrace
- 65: HxTraceView
- 66: HxView
- 67: HxWatchView
- 68: HxPowerWaveHt
- 69: HxInstrumentData
- 70: HxMethodCopy
- 71: HxBtiElx405AutoWasher
- 72: HxSecurityCom
- 73: HxStarBvsCommand
- 74: HxStarBvsConfig
- 75: HxReportConfig
- 76: HxServices
- 77: HxScheduleView
- 78: HxStCompCmd
- 79: HxStarData
- 80: HxUserManager
- 81: HxSchedCompCmd
- 82: HxFan
- 83: HxSysDeck
- 84: HxVSpin
- 85: HxElementCounter
- 86: HxStarConfig

- 87: HxConfigEditor
- 88: HxStarDevices
- 89: HxVSpinAccess2
- 90: HxEditSequenceDlg
- 91: HxStarBiotechMaintMet
- 92: HxLabwrcat
- 93: HxLabwrCatComponents
- 94: HxStarDynDilLib
- 95: HxMosquito
- 96: HxDatabase
- 97: HxVectorDatabase
- 98: HxTranslationSupport
- 99: HxUtilLib
- 100: HxReagentDisp
- 101: HxSpeVacuum
- 102: HxM384
- 103: HxGen5
- 104: HxBigBearShaker
- 105: HxSys3DView
- 106: HxImpactCmd
- 107: HxXRPLiteMC
- 108: NimbusFourProbe
- 109: TrackGripper
- 110: HxCoreDevices
- 111: HxCoreLiquid
- 112: HxCoreLiquidEditor
- 113: HxXRPLiteConfigurator
- 114: HxTcpIpBdzComm
- 115: MiroIncubatorCmd
- 116: GuavaLinkCmd
- 117: PowerSocket
- 118: HxCustomDialog
- 119: GlasColMixerCmd
- 120: EntryExit
- 121: ForteOctetCmd
- 122: NexusXPeelCmd

- 123: PHERAstar_module
- 124: HXETRACKCMD

2.2 Minor IDs that have been identified

NB: Codes are once again in hexadecimal. I'm slightly unsure on some of these

HxVectorDB:

- 0: UnexpectedError
- 1: InterfaceNotInitialized
- 2: CannotInitializedInterface
- 3: CannotGetTableSchema
- 4: CannotGetElementCounter
- 5: ErrorInWorkerThread
- 6: CannotGetProperty
- 7: CannotSetProperty
- 8: CannotGetTranslationTable
- 9: CannotAddAdditionalData
- 10: CannotGetAdditionalData
- 11: CannotDeleteAdditionalData
- 16: Worklist_CannotAddJobs
- 17: Worklist_CannotGetJobs
- 18: Worklist_CannotGetJobAdditionalData
- 19: Worklist_CannotRemoveJob
- 20: Worklist_CannotGetJobState
- 21: Worklist_CannotAddJobAdditionalData
- 22: Worklist_CannotDeleteJobAdditionalData
- 23: Worklist_CannotGetJob
- 24: Worklist_CannotSetJobState
- 48: Tracking_CannotStartRun
- 49: Tracking_CannotPauseRun
- 50: Tracking_CannotResumeRun
- 51: Tracking_CannotEndRun
- 52: Tracking_CannotAbortRun
- 53: Tracking_CannotInterruptRun
- 54: Tracking_CannotCreateDeck
- 55: Tracking_CannotGetDeckID

- 56: Tracking_CannotGetAllLabwareOnDeck
- 57: Tracking_CannotGetLoadStateOfLabware
- 58: Tracking_CannotGetElementID
- 59: Tracking_CannotGetLabware
- 60: Tracking_CannotGetLabwareLoadingTime
- 61: Tracking_CannotAssignLabwareToJob
- 62: Tracking_CannotAssignLabwareToJobs
- 63: Tracking_CannotGetLabwareBarcode
- 64: Tracking_CannotGetLabwareVolume
- 65: Tracking_CannotGetLabwareState
- 66: Tracking_CannotUpdateTADMCurveIDForVolumeMove
- 67: Tracking_CannotGetLabwareLastSourceBarcode
- 68: Tracking_CannotGetLabwareSourceBarcodeList
- 69: Tracking_CannotGetLabwareInitialValues
- 70: Tracking_CannotGetRunID
- 71: Tracking_CannotGetRun
- 72: Tracking_CannotGetRunState
- 73: Tracking_CannotGetUserRunState
- 74: Tracking_CannotSetUserRunState
- 75: Tracking_CannotGetRunActions
- 76: Tracking_CannotGetRunAction
- 77: Tracking_CannotAddRunAction
- 78: Tracking_CannotGetElementIDs
- 79: Tracking_CannotGetLabwareHierarchy
- 80: Tracking_CannotTrackActionLoad
- 81: Tracking_CannotTrackActionUnload
- 82: Tracking_CannotTrackActionMoveVolume
- 83: Tracking_CannotTrackActionMove
- 84: Tracking_CannotTrackActionWash
- 85: Tracking_CannotTrackActionIncubate
- 86: Tracking_CannotTrackActionSetBarcode
- 87: Tracking_CannotTrackActionSetLabwareState
- 88: Tracking_CannotTrackActionCustomAction

HxDatabase:

- 00: Global
- 01: Singleton

- 02: ResourceManager
- 03: Error
- 05: IHxDbManagement
- 06: IHxDbConfiguration
- 07: Utilities
- 16: IHxDbCommand
- 17: IHxDbCommandCollection
- 18: IHxDbConnection
- 19: IHxDbCreateParameterCollection
- 20: IHxDbCreateProcedureCommand
- 21: IHxDbCreateTableCommand
- 22: IHxDbDataReader
- 23: IHxDbParameter
- 24: IHxDbParameterCollection
- 25: IHxDbTransaction

[Need sorting]

- 10: HxEmail
- 20: HxErrorEvent
- 21: SendAddress
- 22: SendFlag
- 23: ExecuteFlag
- 24: ExecuteName
- 26: startApplication
- 27: ExecuteArgument
- 40: HxGeneralSettings()
- 42: RequiredPasswordLength
- 43: SystemName
- 44: SimulationOn
- 45: GetSequenceRGB
- 46: AskForSequenceNameAfterDrop
- 47: GetSound
- 48: SetSound
- 49: GetTimeout
- 50: SetTimeout
- 60: HxInstallation
- 62: GetFeatureDescription

- 63: ActivateFeature
- 64: InstallFeature
- 65: GetFeatureExpiryISODate
- 66: GetFeatureStatusText
- 67: LegalizeInstallation
- 68: GetFeatureNameFromId
- 69: GetFeatureDescriptionById
- 70: GetFeatureExpiryISODateById
- 71: GetFeatureStatusTextById
- 72: UninstallFeature

2.3 Internal Error Codes which have been identified (presumably is the same as SpecificErrorId?)

NB: Once again, these are in Hexadecimal and four digits - error code 10 is 000A

HxVectorDatabase:

- 0: UnexpectedError
- 1: InterfaceAlreadyInitialized
- 2: CannotDetermineAdditionalValueType
- 3: UnknownEnumValue
- 4: UnsupportedEnumValue
- 5: UnsupportedValueFromDB
- 6: InconsistentDB
- 7: NotImplemented
- 8: CannotConnectToDb
- 9: CannotDetermineOS
- 10: ValueOutOfRange
- 11: ExternalDatabaseServerNotSupportedInStandardVersion
- 12: FunctionNotSupportedInStandardVersion
- 13: FunctionNotSupportedOnRemoteDatabaseServer
- 14: CannotFindConfigFile
- 15: CannotFindSqlScript
- 16: CannotConvertValueToByte
- 17: CannotConvertValueToShort
- 18: CannotConvertValueToInt
- 19: CannotConvertValueToLong

- 20: CannotConvertValueToBool
- 21: CannotConvertValueToDouble
- 22: CannotConvertValueToString
- 23: CannotConvertValueToDateTime
- 32: CannotConvertValueToHxVectorDbJobState
- 33: CannotConvertValueToHxVectorDbValueType
- 34: CannotConvertValueToHxVectorDbActionState
- 35: CannotConvertValueToHxPars
- 36: CannotConvertValueToHxVectorDbLabwareHandling
- 37: CannotConvertValueToHxVectorDbLabwareLevel
- 38: CannotConvertValueToHxVectorDbLabwareState
- 39: CannotConvertValueToHxVectorDbStepType
- 40: BadParameterSupplied
- 41: CannotConvertValueToHxVectorDbRunState
- 42: CannotSetValueInConfigFile
- 43: CannotGetValueInConfigFile
- 44: CannotFindStringInStringTable
- 45: CannotConvertValueToHxVectorDbActionType
- 46: CannotConvertValueToHxVectorDbLabwareUsageType
- 47: CannotConvertValueToHxVectorDbRunAction
- 48: CannotDetachDatabase
- 49: CannotConvertValueToHxVectorDbSortingAlgorithm
- 50: CannotRenameExistingDatabaseFiles
- 51: CannotConvertValueToHxVectorDbActionType
- 52: SqlScriptChecksumVerificationFailed
- 4097: CannotFindJob
- 4098: CannotDeterminePhoenixVersion
- 4099: CannotDetermineCurrentUsername
- 4100: RunAlreadyStarted
- 4101: RunNotRunning
- 4102: RunNotPaused
- 4103: MultipleRunsWithSameGUIDDetected
- 4104: CannotUpdateInternalRunID
- 4105: CannotDetermineInstrumentIDForConfiguration
- 4106: InstrumentDuplicatesInDatabse
- 4107: InstrumentConfigurationDuplicatesInDatabase

- 4108: DeckAlreadyExistsForInstrument
- 4109: LabwareAlreadyExists
- 4110: LabwareDoesNotExist
- 4111: OnlySingleLabwareAccessAllowed
- 4112: UnknownDeckID
- 4113: IllegalLabwareHandlingCombination
- 4114: CannotExtractLabwareName
- 4115: CannotMixActionsOfDifferentActionTypes
- 4116: IfNotExistsCreateNotAllowed
- 4117: IfExistsRemoveNotAllowed
- 4118: IfNotExists_ErrorNotAllowed
- 4119: CannotExtractBaseName
- 4120: CannotNormalizeLabwareAccessName
- 4121: CannotLinkLabware
- 4122: CannotExtractInstrumentName
- 4123: CannotSplitLabwareAccessName
- 4124: CannotDetermineLabwareStatePriority
- 4125: CannotConvertActionStateToLabwareState
- 4126: WrongNumberOfLabwareForAction
- 4127: CannotFindRun
- 4128: CannotFindAction
- 4129: CannotFinishPreWork
- 4130: CannotCreateAction
- 4131: CannotLinkLabware
- 4132: CannotAddDetail
- 4133: CannotAddAdditionalData
- 4134: CannotUpdateLabwareData
- 4135: IfNotExistsIgnoreActionNotAllowed
- 4136: DeckDoesNotExistForInstrument
- 4137: SequenceNotValidAtIndex
- 4138: CannotAddAdditionalData
- 4139: CannotGetAdditionalData
- 4140: CannotDeleteAdditionalData
- 4141: CannotFindRunAction
- 4142: CannotFindInstrument
- 4143: SequenceNotValid

- 4144: ConnectedContainerNotAllowed
- 4145: LabwareOrLabwareTypeNotValid
- 4146: LabwareTypeDoesNotExist
- 4147: LabwareMainTypeDoesNotExist
- 4148: ExperimentDoesNotExist
- 4149: LabwareOrExperimentDoesNotExist
- 4150: CannotAddErrorInfo
- 4151: CurrentLabwareDoesNotMatchPreviousLabware
- 4152: CurrentLabwareAlreadyUsed
- 4153: LabwareIsAlreadyPartOfExperiment
- 4154: ExperimentAlreadyExists
- 4155: DatabaseAlreadyExists
- 4156: DatabaseFileAlreadyAttached
- 4157: CannotFindAValueForAdditionalDataKey
- 4158: IfExistsErrorNotAllowed
- 4159: BadLabwareAccessName
- 4160: CannotFindDeck
- 4161: FileDoesNotExist
- 4162: CannotRemoveExperimentSourceLabware
- 4163: LabwareIsCurrentlyLoadedAndMustBeUnloadedBeforeReloading
- 4164: CannotLookupAdditionalData
- 4165: BarcodeAlreadyUsedAsUniqueBarcode
- 4166: BarcodeNotUnique
- 4167: CannotClearUniqueBarcodeList
- 8193: CannotFindReportDirectory
- 40961: AdditionalDataForeignNotExistsAction
- 40962: AdditionalDataForeignNotExistsInstrument
- 40963: AdditionalDataForeignNotExistsInstrumentConfiguration
- 40964: AdditionalDataForeignNotExistsJob
- 40965: AdditionalDataForeignNotExistsLabware
- 40966: AdditionalDataForeignNotExistsRunAction
- 40967: AdditionalDataForeignNotExistsRun
- 65535: CannotCreateErrorInfo

HxDatabase:

- 0: Unexpected
- 1: CreateTableNotSupported

- 2: CreateProcedureNotSupported
- 3: UnknownEnumValue
- 4: UnsupportedEnumValue
- 5: CannotSynchronizeInternalParameterCollection
- 6: InternalParameterCollectionOutOfSync
- 7: UnknownDBMSIdentifier
- 8: CannotLoadConfigFile
- 9: CannotConvertParameter
- 10: CannotCompareCultureAware
- 11: CannotInferValueFromValue
- 12: BadParameterSupplied
- 13: CannotUseSuppliedConfigFile
- 14: ConfigFileContainsErrors
- 15: ParametersMissing
- 16: SQLScriptCommandTextNotSet
- 17: SQLScriptDoesNotExist
- 18: ExecuteSQLScriptRequiresCommandTypeText
- 19: CannotReadSQLScript
- 20: CannotExecuteSQLScript
- 21: NotImplemented
- 22: CouldNotParseConnectionString
- 65535: CannotCreateErrorInfo

CHAPTER

THREE

ASWGLOBAL

https://github.com/theonetruenerd/VenusPackages/blob/main/ASWGlobal.pkg

The ASWGlobal library doesn't add any functions, but instead declares common HSL constants used in other libraries, SMTs, and methods. These are often used as the return values for functions. The constants it declares are:

- False, No, and Off are all assigned as being hslFalse
- True, Yes, and On are all assigned as being hslTrue
- **For dialogue buttons, each one is assigned a specific number:**
  - Ok = 1
  - Cancel = 2
  - Abort = 3
  - Retry = 4
  - Ignore = 5
  - Yes = 6
  - No = 7
- Clicking STOP on a timer is assigned as being 3

CHAPTER

FOUR

ALIQUOT FOR EASY STEPS V2

https://github.com/theonetruenerd/VenusPackages/blob/main/Aliquot_for_EasySteps_v2.pkg

This library adds two functions aimed at making aliquoting steps easier. The functions added are:

- *CalcAliquot_v2_1()*
- *CalcChannelPattern()*

CalcAliquot_v2_1(*sequence DispenseSequence*, *variable VolumePerWell*, *variable VolumePreAliquote*, *variable VolumePostAliquote*, *variable MaxVolumeTip*, *variable NumberOfDispense*, *variable NumberOfChannels*, *variable intNumberOfChannelsInstalled*, *array arrVolumeToAspirateByChannel*, *array arrVolumeToMixBeforeAspiration*, *array arrChannelPatternDispense*, *variable strChannelPatternAspirate*)

This function calculates the important variables for performing an aliquote pipetting with Easy- or Single Steps

Params DispenseSequence
The sequence for the aliquots to be dispensed into

Params VolumePerWell
The volume of the aliquotes per well

Params VolumePreAliquote
The volume of the pre-aliquote

Params VolumePostAliquote
The volume of the post-aliquote

Params MaxVolumeTip
The maximum volume of the selected CO-RE Tip

Params NumberOfDispense
The number of dispense steps per aspiration

Params NumberOfChannels
The number of the channels to use for the process

Params intNumberOfChannelsInstalled
The number of channels installed in the system

Params arrVolumeToAspirateByChannel
An output of the array of volumes to aspirate by channel

Params arrVolumeToMixBeforeAspiration
An output of the array of volumes to mix before aspiration (typically used during the first aspiration only)

Params arrChannelPatternDispense
The output array of channel patterns for the dispense

Params strChannelPatternAspirate
An output string of the channel pattern to aspirate

Returns
None

Return type
N/A

`CalcChannelPattern`(*variable i_int_NumberOfPositions*, *variable i_int_NumberOfInstalledChannels*, *variable o_str_ChannelPattern*)

This function creats a channel pattern based on the number of desired positions and the number of installed channels.

Params i_int_NumberOfPositions
The number of positions to use, which must be less than or equal to the Number of Installed Channels

Params i_int_NumberOfInstalledChannels
The number of channels installed on the instrument

Params o_str_ChannelPattern
The output channel pattern as a string

Returns
None

Return type
N/A

CHAPTER

FIVE

ALIQUOT WITH EASY STEPS

https://github.com/theonetruenerd/VenusPackages/blob/main/Aliquot_for_EasySteps.pkg

The Aliquot with Easy Steps library adds one function aimed at making aliquoting with Easy- or Single steps easier. This library is outdated, the Aliquot_for_EasySteps_v2 library should be used instead. The function it adds is:

- *CalcAliquote()*

CalcAliquote(*sequence DispenseSequence*, *variable VolumePerWell*, *variable VolumePreAliquote*, *variable VolumePostAliquote*, *variable MaxVolumeTip*, *variable VolumeToAspirate*, *variable NumberOfDispense*, *variable FullTrace*, *variable NumberOfChannels*)

This function calculates the important variables for performing aliquoting with Easy- or Single steps.

Params DispenseSequence
The sequence for the aliquots to be dispensed into

Params VolumePerWell
The volume of the aliquots per well

Params VolumePreAliquote
The volume of the pre-aliquot

Params VolumePostAliquote
The volume of the post-aliquot

Params MaxVolumeTip
The maximum volume of the selected CO-RE Tip

Params VolumeToAspirate
The volume which has to be aspirated

Params NumberOfDispense
The number of dispense steps per aspiration

Params FullTrace
The detail level of the trace (1 is full, 0 is normal)

Params NumberOfChannels
The number of channels installed on the system

Returns
None

Return type
N/A

CHAPTER

SIX

ARRAY (FROM HSLEXTENSIONS)

https://github.com/theonetruenerd/VenusPackages/blob/main/Array.pkg

The array library from HSLExtensions adds functions to help manipulate 1-D arrays. The following functions are added:

- *Append()*
- *CompareArrays()*
- *Concat()*
- *ContainsDuplicates()*
- *ContainsValue()*
- *ConvertToBooleanArray()*
- *ConvertToFloatArray()*
- ConvertToIntegerArray()
- *ConvertToStringArray()*
- Copy()
- *FindValue()*
- *InitializeAllValues()*
- *IsBooleanArray()*
- *IsEmpty()*
- *IsFloatArray()*
- *IsIntegerArray()*
- *IsStringArray()*
- *Sort()*

Append(*array io_arrValuesA*, *array i_arrValuesB*)

This function updates the array io_arrValuesA to add all the values from i_arrValuesB at the end of the array.

Params io_arrValuesA
The array to which the values will be added

Params i_arrValuesB
The array from which the values will be added

Returns
None

Return type
N/A

`CompareArrays`(*array i_arrExpectedValues*, *array i_arrActualValues*, *array o_arrMissingValues*, *array o_arrNotExpectedValues*)

This function compares two arrays and outputs arrays of values which are missing from the first array but present in the second, and values which are present in the second array but not in the first.

Params i_arrExpectedValues
The first array, which the second array will be checked against, usually is the array of expected values

Params i_arrActualValues
The second array, which will use the first array as a template when comparing against, usually is your "actual" array

Params o_arrMissingValues
An output array of values which are present in the first array but not the second

Params o_arrNotExpectedValues
An output array of values which are present in the second array but not the first (i.e. unexpected values in your actual data)

Returns
True if both arrays contain the same values (resulting in empty output arrays), false if arrays don't contain the same values (in which case the output arrays will have data in them)

Return type
Boolean

`Concat`(*array i_arrValuesA*, *array i_arrValuesB*)

This function appends one array to the other and then returns the concatenated array. The difference between this and the *Append()* function is that the Append function updates an existing array, whereas this function doesn't change the existing arrays and instead returns a new array.

Params i_arrValuesA
The array to which the values will be added

Params i_arrValuesB
The array from which the values will be added

Returns
A new array which is the concatenated version of the input arrays

Return type
Array

`ContainsDuplicates`(*array i_arrValues*)

This function checks whether the input array has multiple of the same value in it

Params i_arrValues
The array to be checked

Returns
An array with all values which appear more than once in the input array

Return type
Array

`ContainsValue`(*array i_arrValues*, *variable i_varValue*)

This function determines whether a value exists in an array without returning its index

Params i_arrValues
The array to be searched

Params i_varValue
The value to be searched for

Returns
True if the value is present, false otherwise

Return type
Boolean

`ConvertToBooleanArray`(*array i_arrValues*, *variable o_blnSuccessfullyConverted*)

This function converts the input array to an array with boolean values. If it is not possible to convert one or more values of the input array, the output will be false and the output array will be empty. Cannot interact with strings, will convert a non-zero int or float into a 1, and will turn a 0 float into a 0.

Params i_arrValues
The array to be converted

Params o_blnSuccessfullyConverted
A boolean which tells you whether the conversion was successful or not

Returns
The boolean version of the input array

Return type
Array

`ConvertToFloatArray`(*array i_arrValues*, *variable o_blnSuccessfullyConverted*)

This function converts the input array to an array with float values. If it is not possible to convert one or more values of the input array, the output will be false and the output array will be empty. Cannot interact with strings, will convert any int into a float.

Params i_arrValues
The array to be converted

Params o_blnSuccessfullyConverted
A boolean which tells you whether the conversion was successful or not

Returns
The float version of the input array

Return type
Array

`ConvertToIntArray`(*array i_arrValues*, *variable o_blnSuccessfullyConverted*)

This function converts the input array to one with integer values. If it is not possible to convert one or more values of the input array, the output will be false and the output array will be empty. Cannot interact with strings, will round any floats to the nearest integer.

Params i_arrValues
The array to be converted

Params o_blnSuccessfullyConverted
A boolean which tells you whether the conversion was successful or not

Returns
The integer version of the input array

Return type
Array

`ConvertToStringArray`(*array i_arrValues*)

This function converts the input array to one with string values.

Params i_arrValues
The array to be converted

Returns
The float version of the input array

Return type
Array

`Copy`(*array i_arrValues*)

This function will output an exact copy of the input array.

Params i_arrValues
The array to be copied

Returns
A copy of the input array

Return type
Array

`FindValue`(*array i_arrValues*, *variable i_varValue*)

This function will lookup the input variable within the input array and return a 1-based array of the indices of all positions that the input variable was found.

Params i_arrValues
The array to be searched

Params i_varValue
The variable to be searched for

Returns
An array of all the locations that the input variable was found

Return type
Array

`InitializeAllValues`(*array io_arrValues*, *variable i_varValue*)

This function sets all values within an array to the input variable. Does not work on empty arrays.

Params io_arrValues
The array in which all values will be converted.

Params i_varValue
The variable to which all values will be converted.

Returns
None

Return type
N/A

`IsBooleanArray`(*array i_arrValues*)

This function checks whether all values in the array are booleans.

Params i_arrValues
The array to be checked

Returns
A boolean of whether the input array is all booleans or not

Return type
Boolean

`IsEmpty`(*array i_arrValues*)

This function checks whether the input array is empty

Params i_arrValues
The array to be checked

Returns
A boolean of whether the input array is empty or not

Return type
Boolean

`IsFloatArray`(*array i_arrValues*)

This function checks whether all values in the array are floats.

Params i_arrValues
The array to be checked

Returns
A boolean of whether the input array is all floats or not

Return type
Boolean

`IsIntegerArray`(*array i_arrValues*)

This function checks whether all values in the array are integers.

Params i_arrValues
The array to be checked

Returns
A boolean of whether the input array is all integers or not

Return type
Boolean

`IsStringArray`(*array i_arrValues*)

This function checks whether all values in the array are strings.

Params i_arrValues
The array to be checked

Returns
A boolean of whether the input array is all strings or not

Return type
Boolean

`Sort`(*array i_arrValues*, *variable i_intSortMode*, *o_bSuccessfulSorted*)

This function outputs a sorted version of the array using the Shakersort sorting algorithm. All values in the array must share the same type for this function to work. Sort mode can either be 1 or 2, 1 is ascending and 2 is descending.

Params i_arrValues
The array containing the values to be sorted

Params i_intSortMode
Whether the array is to be sorted in ascending (1) or descending (2) order

Params o_bSuccesfulSorted
A boolean of whether the sort was successful or not

Returns
A sorted copy of the array

Return type
Array

CHAPTER

SEVEN

ARRAYTOOLS

https://github.com/theonetruenerd/VenusPackages/blob/main/ArrayTools.pkg

The ArrayTools library provides the following functions:

- *ArraySeqVLookup()*
- *ArrayVLookup()*
- *ConvertArrayOfNumericIntegersToString()*
- ConvertArrayOfNumericStringsToInteger()
- *Lookup()*
- *Sort3ArraysByNumericAscendingOrder()*
- *Update_Value_in_Array()*
- *get_distinct_from_array()*
- *mergeArrays()*
- *removeValueFromArray_basedOnIndex()*

ArraySeqVLookup(*array i_arraySequencesA*, *array i_arrayValuesB*, *variable i_varCondition*, *array o_arraySequencesC*)

Given 2 arrays A (Sequences) and B (Values) with same size Retrieve all the sequences in A where B = input value A(seq1, seq2, seq3,seq4,seq5, seq6, . . .) B(1, 2, 3, 1, 2, 3, . . .)

Get from A all elements where B = 1 C(seq1, seq4, . . .)

Parameters

- **i_arraySequencesA** (*Array of sequences*) – The input array of sequences to filter
- **i_arrayValuesB** (*Array of variables*) – The input array of variables to act as a filter
- **i_varCondition** (*Variable*) – The input value of the variable which is to be selected
- **o_arraySequencesC** (*Array of sequences*) – The output array of the sequences whose index matches the indices in which the the value of the variable is equal to the filter

Returns
None

Return type
N/A

ArrayVLookup(*array i_arrayValuesA*, *array i_arrayValuesB*, *variable i_varCondition*, *array o_arrayValuesC*)

Given 2 arrays of values A and B with same size Retrieve all the elements in B where A = input value

A(100, 20, 15, 45, 42, 1, ...) B(1, 2, 3, 1, 2, 3, ...)

Get from A all elements where B = 1 C(100, 15, ...)

Parameters

- **i_arrayValuesA** (*Array of variables*) – The input array of variables to filter
- **i_arrayValuesB** (*Array of variables*) – The input array of variables to act as a filter
- **i_varCondition** (*Variable*) – The input value of the variable which is to be selected
- **o_arrayValuesC** (*Array of variables*) – The output array of the variables whose index matches the indices in which the value of the variable is equal to the filter

Returns
None

Return type
N/A

ConvertArrayOfNumericIntegersToString(*array i_arr1_int*, *array o_arr1_str*)

Converts all the numeric integers within an array to strings.

Parameters

- **i_arr1_int** (*Array of variables*) – The input array containing the integers to be converted
- **o_arr1_str** (*Array of variables*) – The output array containing the strings

Returns
None

Return type
N/A

ConvertArrayOfNumericStringsToIntegers(*array i_arr1_str*, *array o_arr1_int*)

Converts all the numeric strings within an array to integers.

Parameters

- **i_arr1_str** (*Array of variables*) – The input array containing the strings to be converted
- **o_arr1_int** (*Array of variables*) – The output array containing the integers

Returns
None

Return type
N/A

Lookup(*array array*, *variable item*)

Looks up a value within an array, outputting a 1-based index of the value if found in the array, and a 0 if the value isn't found.

Parameters

- **array** (*Array*) – The input array to be searched
- **item** (*Variable*) – The variable to be searched for

Returns
None

Return type
N/A

`Sort3ArraysByNumericAscendingOrder`(*array io_array1*, *array io_array2*, *array io_array3*)

Sorts 3 arrays by numeric ascending order. io_array1 must contain only numeric values; this one will be sorted and then the other arrays will update to match the new order of io_array1.

Parameters

- **io_array1** (*Array*) – The first of the arrays to be sorted, which must contain only numeric values.
- **io_array2** (*Array*) – The second of the arrays to be sorted, which can contain any values.
- **io_array3** (*Array*) – The third of the arrays to be sorted, which can contain any values.

Returns
None

Return type
N/A

`Update_Value_in_Array`(*array i_array*, *variable i_value*, *variable i_index*)

Overwrites a value in the array at a specified index.

Parameters

- **i_array** (*Array*) – The array in which the value will be changed.
- **i_value** (*Variable*) – The new value to be inserted into the array.
- **i_index** (*Variable*) – The 1-based index of the position in the array to be overwritten.

Returns
None

Return type
N/A

`get_distinct_from_array`(*array i_arr*, *array o_arr*)

Gets all the values in an array that only appear once.

Parameters

- **i_arr** (*Array*) – The input array to be searched.
- **o_arr** – The new output array containing the values which only appear once.

Returns
None

Return type
N/A

`mergeArrays`(*array array1*, *array array2*, *array array3*)

Concatenates two arrays and outputs the result into a third array.

Parameters

- **array1** (*Array*) – The first array of interest.
- **array2** (*Array*) – The second array of interest.

- **array3** (*Array*) – The resulting array of values.

Returns
None

Return type
N/A

removeValueFromArray_basedOnIndex(*array i_array_elements*, *variable i_index_to_remove*)

Removes a value from an array at the specified index.

Parameters

- **i_array_elements** (*Array*) – The array from which an item is to be removed.
- **i_index_to_remove** (*Variable*) – The 1-based index of the array from which the item is to be removed.

Returns
None

Return type
N/A

CHAPTER

EIGHT

COLOURSCHEMELIBRARY

https://github.com/theonetruenerd/VenusPackages/blob/main/ColourSchemeLibrary.pkg

The ColourSchemeLibrary allows the user to change the colour scheme of the Hamilton standard dialogues. It adds the following function

- *Assign()*

Assign(*variable i_str_ColorSchemeCode*)

This function updates the colour scheme of the Standard Dialogues that will be used in the method where the library is executed. Requires HSLExtensions(File), HSLExtensions(Directory), and ASW Global.

Params i_str_ColorSchemeCode
The input color scheme code which is the int KitProviderCode. Hamilton = 0.

Returns
None

Return type
N/A

CHAPTER

NINE

CONVERT FILE TO ASCII

https://github.com/theonetruenerd/VenusPackages/blob/main/ConvertFileToASCII.pkg

The Convert File to ASCII library adds the ability to convert Hamilton files to ASCII files. The function it adds is:

- *ConvertFileToASCII()*

ConvertFileToASCII(*variable i_Str_SourceFile*)

This function converts files from binary to ASCII. The file must be a Hamilton file.

Params i_Str_SourceFile
The path to the file which is being converted.

Returns
None

Return type
N/A

CHAPTER

TEN

DEBUGASSIST

https://github.com/theonetruenerd/VenusPackages/blob/main/DebugAssist.pkg

The debug assist library adds one function aimed at helping the user identify, understand, and fix runtime errors. The function it adds is:

- *FullErrorAnalysis()*

This function can use the ASWStandardDialog library which makes the dialog nicer, but is not a requirement.

FullErrorAnalysis(*variable i_PathForErrorDataAnalysis*)

This function is designed to be called in the "OnAbort" submethod. It will look at the error that caused the abort to be triggered, convert the trace file error code into the more standard form, identify what that error corresponds with and hopefully suggest some initial things to check. In order for the dialogue to pop up, this function requires ASWStandardDialogues to be initialised.

Params i_PathForErrorDataAnalysis
A file path for where the error data will be exported. Currently not very important as the main bonus of the library is the dialogue that pops up, although the intention is to add more detail to this exported file.

Returns
None

Return type
N/A

CHAPTER

ELEVEN

EDITFILEATTRIBUTES

https://github.com/theonetruenerd/VenusPackages/blob/main/EditFileAttributes.pkg

The Edit File Attributes library adds the following functions:

- *Make_Hidden()*
- *Make_ReadOnly()*
- *Remove_Hidden()*
- *Remove_ReadOnly()*

Make_Hidden(*variable Filename*)

Applies the hidden attribute to the file found at the path specified.

Params filename
The path to the file that is going to be modified.

Returns
None

Return type
N/A

Make_ReadOnly(*variable Filename*)

Applies the read only attribute to the file found at the path specified.

Params filename
The path to the file that is going to be modified.

Returns
None

Return type
N/A

Remove_Hidden(*variable Filename*)

Removes the hidden attribute from the file found at the path specified.

Params filename
The path to the file that is going to be modified.

Returns
None

Return type
N/A

Remove_ReadOnly(*variable Filename*)

Removes the read only attribute from the file found at the path specified.

Params filename
The path to the file that is going to be modified.

Returns
None

Return type
N/A

CHAPTER

TWELVE

ERROR REPORT LIBRARY

https://github.com/theonetruenerd/VenusPackages/blob/main/ErrorReportLibrary.pkg

The error report library generates concise reports on both general errors and pipetting errors that occur during a run, extracting the information from the trace file. It adds the following functions:

- *CopyActiveTraceFile()*
- *GenerateGeneralErrorsReport()*
- *GeneratePipettingErrorsReport()*

CopyActiveTraceFile(*variable o_strTraceCopyFilePath*)

Finds the trace file that is currently active (i.e. associated with the current run) and makes a copy of it, which it saves in the location specified.

Params o_strTraceCopyFilePath
The location that the copy of the trace file will be created in

Returns
None

Return type
N/A

GenerateGeneralErrorsReport(*variable i_strTraceFilePath*, *variable o_strPDF_FilePath*)

Reads the active trace file, locates errors found within it and generates an error report on errors found in non-pipetting steps

Params i_strTraceFilePath
The path to the current trace file

Params o_strPDF_FilePath
The file path for the exported PDF of the error report

Returns
None

Return type
N/A

GeneratePipettingErrorsReport(*variable i_strTraceFilePath*, *variable i_strTemplateFilePath*, *variable i_intWriteRowStart*, *o_strPDF_FilePath*)

Reads the active trace file, locates errors in the pipetting steps within it and generates an error report for the pipetting steps only

Params i_strTraceFilePath
The path to the current trace file

Params i_strTemplateFilePath
The path to the excel file used as a template for generating the report; should be installed when the library is unpacked

Params i_intWriteRowStart
Which row of the excel file to begin the report on; usually two, can be more than two

Params o_strPDF_FilePath
The file path for the exported PDF of the error report

Returns
None

Return type
N/A

CHAPTER

THIRTEEN

ERROR SIMULATOR

https://github.com/theonetruenerd/VenusPackages/blob/main/ErrorSimulator.pkg

This library adds the ability to simulate a variety of errors, to assist with debugging or custom error handling. The functions it adds are:

- *AA_Abstract()*
- *STEP1_PrepareRegistryAndCfgFile()*
- *STEP2a_SimulateError_Channels()*
- *STEP2b_SimulateError_COREGripper()*
- *STEP2c_SimulateError_iSWAP()*
- *STEP2d_SimulateError_96Head()*
- STEP2e_SimulateError_384Head()
- *STEP2f_SimulateError_BarcodeReading()*
- *STEP2g_SimulateError_Autoload()*
- *STEP2h_SimulateError_CRWashstation()*
- *STEP3_Restore_BackupCfgFile()*
- *STEP4_Optional_SwitchChecksum_ON()*

AA_Abstract()

This function does nothing. In the submethod library itself it simply lists the changelog.

STEP1_PrepareRegistryAndCfgFile(*device ML_STAR*)

This function prepares the system files and registry to perform the error simulation. You can put this step at the beginning of your method, and you can disable it after the 1st run, once all registry and configuration settings are done. It switches off the checksum in the registry, asking for confirmation to modify the registry during run. It converts the ConfigML_STAR_Simulator.cfg file to ASCII. It creates a copy of ML_STAR_Simulator.cfg named ML_STAR_Simulator.cfg.bak for future restoration if needed. It replaces the string reload "0" with reload "1".

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

STEP2a_SimulateError_Channels(*variable ChannelNumber*, *variable WhenSimulateError*, *variable ErrorToSimulate*, *device ML_STAR*)

This function simulates an error in the channels. It can simulate an error of your choice (between errors 1 and 12) in a desired channel at any step of the pipetting process. The error codes are: - 1..... Liquid Level not found (error 06/70) - 2..... Not enough liquid (error 06/71) - 3..... Liquid level not found with Dual LLD (error 06/73) - 4..... Clot error (error 04/81) - 5..... Liquid not correctly aspirated (error 06/80) - 6.....Tip already present (error 07/00) - 7..... No Tip (error 08/00) - 8..... Wrong tip type detected (error 08/78) - 9..... Syntax / parameter out of range (error 01/00) - 10..... Hardware (error 02/00) - 11..... Not Executed (error 03/00) - 12..... Not Completed (error 10/00)

Params ChannelNumber
A string of the channel(s) in which the error is to be simulated, e.g. "1,2,4,7"

Params WhenSimulateError
The step of the process in which the error is desired to be simulated. 1 = Aspiration, 2 = Dispense, 3 = Tip pickup, 4 = Tip eject

Params ErrorToSimulate
Integer corresponding to the desired simulated error, key listed in function description.

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

STEP2b_SimulateError_COREGripper(*variable ChannelNumber*, *variable WhenSimulateError*, *variable ErrorToSimulate*, *device ML_STAR*)

This function simulates an error in the CO-RE gripper. It can simulate an error of your choice (between errors 1 and 4) in a desired channel at any step of the process. The error codes are: - 1..... Can´t get the CORE grippers (error 08/75) - 2..... Step lost in Z drive, crash against something (error 02/62) - 3..... Hardware (error 02/00) - 4..... Read Barcode Error (05/00)

Params ChannelNumber
A string of the channel(s) in which the error is to be simulated, e.g. "1,2,4,7"

Params WhenSimulateError
The step of the process in which the error is desired to be simulated. 1 = Pickup CO-RE, 2 = Get Plate, 3 = Put Plate, 4 = Eject CO-RE gripper

Params ErrorToSimulate
Integer corresponding to the desired simulated error, key listed in function description.

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

STEP2c_SimulateError_iSWAP(*variable WhenSimulateError*, *variable ErrorToSimulate*, *device ML_STAR*)

This function simulates an error in the iSWAP. It can simulate an error of your choice (between errors 1 and 4) at any step of the process. The error codes are: - 1..... Expected object not found (error 21/94) - 2..... Step lost in Z drive, crash against something (error 02/62) - 3..... Hardware (error 02/00) - 4..... Object lost (error 23/96)

Params WhenSimulateError
The step of the process in which the error is desired to be simulated. 1 = Get Plate, 2 = Put Plate

Params ErrorToSimulate
Integer corresponding to the desired simulated error, key listed in function description.

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

`STEP2d_SimulateError_96Head`(*variable WhenSimulateError*, *variable ErrorToSimulate*, *device ML_STAR*)

This function simulates an error in the 96-head. It can simulate an error of your choice (between errors 1 and 12 - not all available) at any step of the process. The error codes are: - 1..... Liquid Level not found (error 06/70) - 2..... Not enough liquid (error 06/71) - 6..... Tip already present (error 07/00) - 7..... No Tip (error 08/00) - 8..... Wrong tip type detected (error 08/78) - 9..... Syntax / parameter out of range (error 01/00) - 10..... Hardware (error 02/00) - 11..... Not Executed (error 03/00) - 12..... Not Completed (error 10/00)

Params WhenSimulateError
The step of the process in which the error is desired to be simulated. 1 = Aspiration, 2 = Dispense, 3 = Tip Pickup, 4 = Tip eject, 5 = Wash

Params ErrorToSimulate
Integer corresponding to the desired simulated error, key listed in function description.

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

`STEP2e_SimulateError_384head`(*variable WhenSimulateError*, *variable ErrorToSimulate*, *device ML_STAR*)

This function simulates an error in the 384-head. It can simulate an error of your choice (between errors 1 and 12 - not all available) at any step of the process. The error codes are: - 1..... Liquid Level not found (error 06/70) - 2..... Not enough liquid (error 06/71) - 6..... Tip already present (error 07/00) - 7..... No Tip (error 08/00) - 8..... Wrong tip type detected (error 08/78) - 9..... Syntax / parameter out of range (error 01/00) - 10..... Hardware (error 02/00) - 11..... Not Executed (error 03/00) - 12..... Not Completed (error 10/00)

Params WhenSimulateError
The step of the process in which desired error is to be simulated. 1 = Aspiration, 2 = Dispense, 3 = Tip Pickup, 4 = Tip eject, 5 = Wash.

Params ErrorToSimulate
Integer corresponding to the desired simulated error, key listed in function definition.

Params ML_STAR
The instrument being used. Should be ML_STAR

Returns
None

Return type
N/A

STEP2f_SimulateError_BarcodeReading(*variable WhenSimulateError*, *device ML_STAR*)

This function simulates an error during the barcode reading step. It can simulate the error as though it was either the CO-RE grips or the iSWAP involved.

Params WhenSimulateError
Poorly labelled variable, likely copied and pasted. This is not when the error is simulated; it is instead whether the barcode is being read with CO-RE grips (1) or the iSWAP (2)

Parameters
ML_STAR (*Device*) – The instrument being used. Should be ML_STAR

Returns
None

Return type
N/A

STEP2g_SimulateError_Autoload(*variable notReadPosition_Str*, *variable notPresentPosition_Str*, *device ML_STAR*)

This submethod simulates the "Barcode not read" and "Labware not Present Errors".you can enter positions with errors separated by a comma

Params notReadPosition_Str
A string of comma separated values with the positions in which you want to simulate barcode errors

Params notPresentPosition_Str
A string of comma separated values with the positions in which you want to simulate labware not present errors.

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

STEP2h_SimulateError_CRWashstation(*variable errorToSimulate*, *device ML_STAR*)

This function simulates an error when starting the washing with the CR washstation

Params errorToSimulate
The error being simulated. 1 = Not washing liquid error, 2 = hardware error.

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

STEP3_Restore_BackupCfgFile(*device ML_STAR*)

This function restores the backup configuration file created in function STEP1_PrepareRegistryAndCfgFile

Params ML_STAR
The instrument being used. Should be ML_STAR.

Returns
None

Return type
N/A

STEP4_Optional_SwitchChecksum_ON()

This function is optional. Use it at the end of your method if you want to Switch the File Checksum ON in the registry

Returns
None

Return type
N/A

CHAPTER

FOURTEEN

FRAMEWORK (FROM HSLEXTENSIONS)

https://github.com/theonetruenerd/VenusPackages/blob/main/Framework.pkg

The framework library from HSLExtensions provides two visible functions, but is also required for a few of the other HSLExtension libraries. It uses the functionality of the ASW Standard TraceLevel library. The functions it adds are:

- *GetVersion()*
- *SetTraceLevel()*

GetVersion()

This function gets the framework version

Returns
The framework version

Return type
String

SetTraceLevel(*variable i_intTraceLevel*)

This function sets the trace level of the framework.

Params i_intTraceLevel
The trace level, for the ASWStandard::TraceLevel library. 0 = TRACE_LEVEL_NONE, 1 = TRACE_LEVEL_RELEASE, 2 = TRACE_LEVEL_DEBUG.

Returns
None

Return type
N/A

CHAPTER

FIFTEEN

HSLDECKVISUALIZE

https://github.com/theonetruenerd/VenusPackages/blob/main/HSLDeckVisualize.pkg

The HSLDeckVisualize library adds functions which will update the layout shown in the run control to account for manually performed actions or cause specific effects to occur. The functions it adds are:

- *UpdateUsedPositions()*
- *UpdateUsedLabware()*
- *UpdateLoadedLabware()*

UpdateUsedPositions(*device dev*, *sequence positions*, *variable action*, *variable description*)

This function updates the list of used positions and updates the view.

Params dev
The instrument for which the associated deck view will be updated. ML_STAR should be the only option visible in the dropdown.

Params positions
The sequence of labware and positions IDs to be updated.

Params action
The action to be performed on the specified positions. The same action will be performed for all positions. Action can be an integer from 0-6, with the values each representing a different event. 0 = Position is selected, 1 = Processing, 2 = Reserved, 3 = Error, 4 = Processed, 5 = Reset action state to none, 6 = Position selected and flashing.

Params description
The description displayed in the first section of the status bar in the deck view.

Returns
None

Return type
N/A

UpdateUsedLabware(*device dev*, *array labware*, *array action*, *variable description*)

This function updates the list of used labware and updates the view accordingly. The labware and action arrays must be the same size.

Params dev
The instrument for which the associated deck view will be updated. ML_STAR should be the only option visible in the dropdown.

Params labware
The array of labware IDs for updating

Params action
The array of actions which will be applied to the associated labware at the same index in the labware array. Action can be an integer from 0-6, with the values each representing a different event. 0 = Position is seleccted, 1 = Processing, 2 = Reserved, 3 = Error, 4 = Processed, 5 = Reset action state to none, 6 = Position selected and flashing.

Params description
The description displayed in the first section of the status bar in the deck view.

Returns
None

Return type
N/A

`UpdateLoadedLabware`(*device dev*, *array labware*, *array state*, *variable description*)

This function updates the list of loaded labware and updates the view accordingly. The labware and action arrays must be the same size.

Params dev
The instrument for which the associated deck view will be updated. ML_STAR should be the only option visible in the dropdown.

Params labware
The array of labware IDs for updating

Params state
The array of load state which will be applied to the associated labware at the same index in the labware array. State can be an integer from 0-5, with the values each representing a different state. 0 = unload the labware and make not visible, 1 = load the labware, 2 = preparing to unload, 3 = preparing to load, 4 = labware will flash, 5 = labware will flash

Params description
The description displayed in the first section of the status bar in the deck view.

Returns
None

Return type
N/A

CHAPTER

SIXTEEN

HSLFILLIB

https://github.com/theonetruenerd/VenusPackages/blob/main/HSLFilLib.pkg

This library allows interaction with and manipulation of files present on the host computer. It adds the following functions:

- *FilEof()*
- *FilFindFile()*
- *FilFindNextFile()*
- *FilFormatBarcodeFile()*
- *FilGetBinPath()*
- *FilGetCommState()*
- FilGetCommTimeouts()
- *FilGetConfigPath()*
- *FilGetLabwarePath()*
- *FilGetLibraryPath()*
- *FilGetLogFilesPath()*
- *FilGetMethodsPath()*
- *FilGetSystemPath()*
- *FilIsNull()*
- *FilReadString()*
- *FilRemoveFields()*
- *FilSearchPath()*
- *FilSetCommState()*
- *FilSetCommTimeouts()*
- *FilUpdateRecord()*
- *FilWriteString()*

FilEof(*variable filObj*)

This function checks whether the current position in the specified file is the final line

Params filObj
The opened file to be checked

Returns
Boolean as to whether the position is the end of the file or not

Return type
Boolean

`FilFindFile`(*variable filName*)

This function starts searching the specified path for a file. This function is obsolete, and *FilSearchPath()* should be used instead.

Params filName
The directory or path and file name to be searched for. Can contain wildcard characters such as * or ?

Returns
If successful, the path name of the first file found that matches the input

Return type
Variable

`FilFindNextFile`()

Continues the search from *FilFindFile()* to the next file.

Returns
If successful, the path name of the next file found that matches the input from the most recent FilFindFile command

Return type
Variable

`FilFormatBarcodeFile`(*variable dataSource*, *variable dataTarget*)

This function takes the barcode ASCII text file written during LoadCarrier and converts it into a strongly formatted barcode file. This strongly formatted file can be an ASCII text file, a Microsoft Excel file, or a Microsoft Access file. Will contain the following columns:

- ID (record ID, integer)
- Specifier (string, P = position, C = carrier)
- Position (position, integer)
- Barcode (barcode, string)
- Timestamp (timestamp, YYYY-MM-DD hh:mm:ss, string)

Params dataSource
The name of the barcode ASCII file generated during the load carrier step

Params dataTarget
The name of the barcode ASCII text file, Microsoft Excel file or Microsoft Access file generated by the function. The name must include a table name for a Microsoft Excel file or a Microsoft Access Database.

Returns
Boolean showing if the function was successful or not

Return type
Boolean

`FilGetBinPath`()

This function retrieves the vector binary path

Returns
The vectory binary path (usually C:Program Files (x86)HamiltonBin)

Return type
Variable

`FilGetCommState`(*file port*)

This function retrieves the configuration information for the specified communication resource. The entries of the structure that retrieves the configuration information must be accessible in the global scope.

Params port
The communication resource opened during the file-Open operation

Type
Port

Returns
Boolean showing if the function was successful or not

Return type
Boolean

`GetCommTimeouts`(*file port*)

This function retrieves the time-out parameters for all read and write operations for the specified communication resource. The entries of the structure that contains the configuration information must be accessible in the global scope.

Params port
The communication resource opened during the file-Open operation

Returns
Boolean showing if the function was successful or not

Return type
Boolean

`FilGetConfigPath`()

This function retrieves the vector configuration path

Returns
The vector configuration path (usually C:Program Files (x86)HamiltonConfig)

Return type
Variable

`FilGetLabwarePath`()

This function retrieves the vector labware path

Returns
The vector labware path (usually C:Program Files (x86)HamiltonLabware)

Return type
Variable

`FilGetLibraryPath`()

This function retrieves the vector library path

Returns
The vector library path (usually C:Program Files (x86)HamiltonLibrary)

Return type
Variable

FilGetLogFilesPath()

This function retrieves the vector log files path

Returns
The vector log files path (usually C:Program Files (x86)HamiltonLogFiles)

Return type
Variable

FilGetMethodsPath()

This function retrieves the vector methods path

Returns
The vector methods path (usually C:Program Files (x86)HamiltonMethods)

Return type
Variable

FilGetSystemPath()

This function retrieves the vector system path

Returns
The vector system path (usually C:Program Files (x86)HamiltonSystem)

Return type
Variable

FilIsNull(*variable value*)

This function returns a non-zero if the variable is a null value (SQL style Null).

Params value
The variable being checked

Returns
A boolean determining if the variable is SQL style Null or not

Return type
Boolean

FilReadString(*file fileObj*)

This function reads the next record from the input file as string-valued data. Row data, but no schema data, is saved to the string. After you call FilReadString, the next unread record becomes the current record, or Eof is set to hslTrue if there are no more records.

Params fileObj
The file being looked at

Returns
The contents of the line being looked at as string-valued data, or the specific run-time error

Return type
String

FilRemoveFields(*file fileObj*)

This function removes all fields from a record definition

Params fileObj
The file containing the record which is having its fields removed

Returns
None

Return type
N/A

`FilSearchPath`(*variable fileName*)

This function searches for the specified file, and outputs the path and filename of the file if found, or an empty string if not found. Will search the current directory, the methods directory, the library directory, and any directories in the PATH environment variable.

Params fileName
The file name to be searched for

Returns
The path name of the first file found, or an empty string if no files were found

Return type
Variable

`FilSetCommState`(*file port*, *variable cfgFile*)

This function configures a communication resource according to the specifications in a structure that contains the configuration information. The structure that contains the configuration information must be structured as shown below. Each entry in the structure is optional and overwrites the default value in parentheses.

Params port
The communication resource opened during the file-Open operation

Params cfgFile
The name of the file containing the configuration information. If this parameter is empty, the entries in the structure that contains the configuration information must be accessible in the global scope.

Returns
Boolean showing whether the function succeeded or not

Return type
Boolean

`FilSetCommTimeouts`(*file port*, *variable cfgFile*)

This function sets the time-out parameters for all read and write operations on a specified communication resource. The structure that contains the time-out information is as shown below. Each entry in the structure is optional and overwrites the default value in parentheses.

Params port
The communication resource opened during the file-Open operation

Params cfgFile
The name of the file that contains the time-out information. If this parameter is empty, the entries in the structure that contains the time-out information must be accessible in the global scope.

Returns
Boolean showing whether the function succeeded or not

Return type
Boolean

`FilUpdateRecord`(*file fileObj*)

Updates the current record of the file object with the values of the variable objects specified in the record definition. The current record remains current after you call the FilUpdateRecord function. The provider must support UPDATE.

Params fileObj
The file object being updated

Returns
Boolean showing whether the function succeeded or not

Return type
Boolean

`FilWriteString`(*file fileObj*, *variable stringObj*)

Writes a string to the end of the file data source. After you call the FilWriteString function, the new record becomes the current record.

Params fileObj
The file which is being written in

Params stringObj
The string to be written

Returns
Boolean showing whether the function succeeded or not

Return type
Boolean

CHAPTER

SEVENTEEN

HSLFILELIBEX

https://github.com/theonetruenerd/VenusPackages/blob/main/HSLFilLibEx.pkg

HSLFilLibEx allows you to manipulate files. It adds the following functions:

- *FilCopyFileEx()*
- *FilDeleteFileEx()*
- *FilFormatReportFileEx()*

FilCopyFileEx(*variable SourceFilePathName*, *variable DestinationFilePathName*)

This function creates a copy of the specified file, which will be placed in the specified new location

Params SourceFilePathName
The path to the file which you wish to copy, including the file name and extension

Params DestinationFilePathName
The path to which you would like to copy the file, including the file name and extension

Returns
None

Return type
N/A

FilDeleteFileEx(*variable FilePathName*)

This function deletes the file at the specified location

Params FilePathName
The path to the file which you wish to delete, including file name and extension

Returns
None

Return type
N/A

FilFormatReportFileEx(*variable SourceFilePathName*, *variable DestinationFilePathName*)

This function creates a formatted/filtered copy of the specified report file. It searches the file for the string "Element Name" and copies any subsequent lines to the new file.

Params SourceFilePathName
The path to the file which you wish to format, including file name and extension

Params DestinationFilePathName
The path to the file you would like to create the formatted report in, including file name and extension.

Returns
None

Return type
N/A

CHAPTER

EIGHTEEN

HSLSTATISTICS

https://github.com/theonetruenerd/VenusPackages/blob/main/HSLStatistics.pkg

The HSLStatistics library is designed to easily allow the user to perform basic statistical functions easily. It adds the following functions:

- *Stat_Average()*
- *Stat_StdDeviation()*
- Stat_CorrelationCoefficent()
- *Stat_RSQ()*
- *Stat_Slope()*
- *Stat_Intercept()*
- *Stat_Min()*
- Stat_Max()

Stat_Average(*array DataArray*)

This function returns the average of a data set input as an array

Params DataArray
The input array of data from which the average is calculated

Returns
The calculated average of the array

Return type
Variable

Stat_StdDeviation(*array DataArray*)

This function returns the standard deviation of a data set input as an array

Params DataArray
The input array of data from which the standard deviation is calculated

Returns
The calculated standard deviation of the array

Return type
Variable

Stat_CorrelationCoefficient(*array XArray*, *array YArray*)

This function returns the correlation coefficient (r-value) of a paired data set

Params XArray
The array of X data values in the paired set

Params YArray
The array of Y data values in the paired set

Returns
The r-value of the paired data set

Return type
Variable

`Stat_RSQ`(*array XArray*, *array YArray*)

This function returns the pearson coefficient (r^2) of a paired data set

Params XArray
The array of X data values in the paired set

Params YArray
The array of Y data values in the paired set

Returns
The pearson coefficient of the paired data set

Return type
Variable

`Stat_Slope`(*array XArray*, *array YArray*)

This function returns the slope of the best fit line of a paired data set

Params XArray
The array of X data values in the paired set

Params YArray
The array of Y data values in the paired set

Returns
The slope of the best fit line

Return type
Variable

`Stat_Intercept`(*array XArray*, *array YArray*)

This function returns the intercept of the best fit line of a paired data set

Params XArray
The array of X data values in the paired set

Params YArray
The array of Y data values in the paired set

Returns
The intercept of the line of best fit

Return type
Variable

`Stat_Min`(*array DataArray*)

This function returns the lowest value of a dataset

Params DataArray
The array for the minimum to be searched in

Returns
The lowest value of the dataset

Return type
Variable

CHAPTER

NINETEEN

HSLSTRLIB

https://github.com/theonetruenerd/VenusPackages/blob/main/HSLStrLib.pkg

HSL String Library provides the following functions:

- *StrAsciiToStr()*
- *StrConcat2()*
- *StrConcat4()*
- *StrConcat8()*
- *StrConcat12()*
- *StrEvaluateExpr()*
- *StrFillLeft()*
- *StrFillRight()*
- *StrFind()*
- *StrFStr()*
- *StrFStrEx()*
- *StrFVal()*
- *StrGetLength()*
- *StrGetType()*
- *StrHexIStr()*
- *StrIsDigit()*
- StrIStr()
- *StrIVal()*
- *StrLeft()*
- *StrMakeLower()*
- *StrMakeLowerCopy()*
- *StrMakeUpper()*
- *StrMakeUpperCopy()*
- *StrMid()*
- *StrReplace()*

- *StrReverseFind()*
- *StrRight()*
- *StrSpanExcluding()*
- *StrStrToAscii()*
- *StrTrimLeft()*
- *StrTrimRight()*

StrAsciiToStr(*variable asciiCode*)

Converts the given ASCII Code (an integer) to a character (string).

Parameters
asciiCode (*Integer*) – The ASCII code to convert

Returns
The ASCII code as a string

Return type
String

StrConcat2(*variable Var1*, *variable Var2*)

Combines two strings into a new string

Parameters

- **Var1** (*String*) – The first string to be combined
- **Var2** (*String*) – The second string to be combined

Returns
The combined string of Var1 + Var2

Return type
String

StrConcat4(*variable Var1*, *variable Var2*, *variable Var3*, *variable Var4*)

Combines four strings into a new string

Parameters

- **Var1** (*String*) – The first string to be combined
- **Var2** (*String*) – The second string to be combined
- **Var3** (*String*) – The third string to be combined
- **Var4** (*String*) – The fourth string to be combined

Returns
The combined string of Var1 + Var2 + Var3 + Var4

Return type
String

StrConcat8(*variable Var1*, *variable Var2*, *variable Var3*, *variable Var4*, *variable Var5*, *variable Var6*, *variable Var7*, *variable Var8*)

Combines eight strings into a new string

Parameters

- **Var1** (*Variable*) – The first string to be combined

- **Var2** (*Variable*) – The second string to be combined
- **Var3** (*Variable*) – The third string to be combined
- **Var4** (*Variable*) – The fourth string to be combined
- **Var5** (*Variable*) – The fifth string to be combined
- **Var6** (*Variable*) – The sixth string to be combined
- **Var7** (*Variable*) – The seventh string to be combined
- **Var8** (*Variable*) – The eighth string to be combined

Returns
The combined string of Var1 + Var2 + Var3 + Var4 + Var5 + Var6 + Var7 + Var8

Return type
String

StrConcat12(*variable Var1*, *variable Var2*, *variable Var3*, *variable Var4*, *variable Var5*, *variable Var6*, *variable Var7*, *variable Var8*, *variable Var9*, *variable var10*, *variable var11*, *variable var12*)

Combines twelve strings into a new string

Parameters

- **Var1** (*Variable*) – The first string to be combined
- **Var2** (*Variable*) – The second string to be combined
- **Var3** (*Variable*) – The third string to be combined
- **Var4** (*Variable*) – The fourth string to be combined
- **Var5** (*Variable*) – The fifth string to be combined
- **Var6** (*Variable*) – The sixth string to be combined
- **Var7** (*Variable*) – The seventh string to be combined
- **Var8** (*Variable*) – The eighth string to be combined
- **Var9** (*Variable*) – The ninth string to be combined
- **Var10** (*Variable*) – The tenth string to be combined
- **Var11** (*Variable*) – The eleventh string to be combined
- **Var12** (*Variable*) – The twelfth string to be combined

Returns
The combined string of Var1 + Var2 + Var3 + Var4 + Var5 + Var6 + Var7 + Var8 + Var9 + Var10 + Var11 + Var12

Return type
Variable

StrEvaluateExpr(*variable expression*)

This function evaluates an expression within a string. All variables involved must have global scope.

Params expression
The expression to evaluate as a string

Returns
The value of the expression if the function succeeds, otherwise a runtime error

Return type
Variable

`StrFillLeft`(*variable str*, *variable character*, *variable width*)

This function fills leading characters to the string

Params str
The string to be modified

Params character
The user-defined character to be filled

Params width
The width to be filled

Returns
The modified string

Return type
Variable

`StrFillRight`(*variable str*, *variable character*, *variable width*)

This function fills trailing characters to the string

Params str
The string to be modified

Params character
The user-defined character to be filled

Params width
The width to be filled

Returns
The modified string

Return type
Variable

`StrFind`(*variable str*, *variable subStr*)

This function searches the string for the first match of the sub-string.

Params str
The string to be searched

Params subStr
The substring to be searched for

Returns
The zero-based index of the first character in this string object that matches the requested sub-string or characters. -1 if the sub-string is not found.

Return type
Integer

`StrFStr`(*variable number*)

This function converts the floating point number input into the corresponding character string.

Params number
The float to be converted into a string

Returns
The string form of the float

Return type
Variable

`StrFStrEx`(*variable number*, *variable languageSpecific*, *variable precision*)

This function converts the floating point number input into the corresponding character string

Params number
The float to be converted into a string.

Params languageSpecific
Boolean which specifies whether the decimal symbol in the Regional Settings should be used to write the string representation of the floating point number.

Params precision
The total number of significant digits to be used for the floating-point display

Returns
The string representation of the floating point number

Return type
Variable

`StrFVal`(*variable str*)

Converts the sequence of digits, contained in the character string str, into the corresponding floating point number. Conversion aborts at the first character in str, which is not a digit or not one of the characters +, -, e, E.

Params str
The string to be converted

Returns
THe float representation of the input string. Null if the string cannot be converted. DBL_MAX if the conversion results in an overflow. DBL_MIN if the conversion results in an underflow.

Return type
Float or variable

`StrGetLength`(*variable str*)

Returns the number of characters in a string object (without '0').

Params str
The string being read

Returns
The length of the string

Return type
Integer

`StrGetType`(*variable var*)

This function retrieves the type of the value of a variable

Params var
A reference to a variable (int, float or string)

Returns
One of the following string-valued constants that indicates the type of the value of a variable. i = hslInteger, f = hslFloat, s = hslString, null = no type

Return type
Variable

`StrHexIStr`(*variable number*)

Converts the input integer into the corresponding hexadecimal character string

Params number
The integer to be converted

Type
Integer

Returns
The hexadecimal string representation of the integer

Return type
String

`StrIsDigit`(*variable character*)

The StrIsDigit function determines if the specified input string (which should be a character) is a digit or not.

Params character
The input character to be a tested, as a string

Returns
Boolean showing whether the character is a digit (1) or not (0)

Return type
Boolean

`StrIsStr`(*variable number*)

The StrIStr function converts the input integer into the corresponding character string

Params number
The integer to be converted

Returns
The string representation of the integer number

Return type
Variable

`StrIVal`(*variable str*)

The StrIVal function converts the input sequence of digits into the corresponding integer. The input string is treated as a decimal, unless it begins with an 0x in which case it is interpreted as hexadecimal. Conversion aborts at the first character in the input which is neither a digit nor one of the characters "+" or "-".

Params str
The input sequence of digits to be converted

Returns
The numeric value of the sequence of digits contained in the character string, as an integer. Null if the character string cannot be converted into a number. LONG_MAX = 2147483647 if the conversion results in an overflow. LONG_MIN = -2147483647 - 1 if the conversion results in an underflow.

Return type
Variable

`StrLeft`(*variable str*, *variable count*)

The StrLeft function extracts the first (leftmost) characters of a string and returns a copy of the extracted substring. The number of characters extracted is equal to the input variable "count". If "count" is longer than the string, the entire string is returned.

Params str
The input string from which the substring is to be extracted

Params count
The number of characters to be extracted

Returns
A string containing a copy of the specified range of characters. Can be an empty string.

Return type
Variable

`StrMakeLower`(*variable str*)

The StrMakeLower function converts the original string to its lowercase form.

Params str
The string to be converted

Returns
The original string converted to lowercase

Return type
Variable

`StrMakeLowerCopy`(*variable str*)

The StrMakeLowerCopy function returns a copy of the original string converted to lowercase.

Params str
The string to be copied

Returns
A copy of the original string converted to lowercase

Return type
Variable

`StrMakeUpper`(*variable str*)

The StrMakeUpper function converts the original string to its uppercase form.

Params str
The string to be converted

Returns
The original string converted to uppercase

Return type
Variable

`StrMakeUpperCopy`(*variable str*)

The StrMakeUpperCopy function returns a copy of the original string converted to uppercase.

Params str
The string to be copied

Returns
A copy of the original string converted to uppercase

Return type
Variable

`StrMid`(*variable str*, *variable first*, *variable count*)

The StrMid function extracts a substring of length "count" characters from the input variable "str", starting at position "first" which is 0-based. The function returns a copy of the extracted substring.

Params str
The string from which the substring is to be extracted

Params first
The first character to be extracted (0-based)

Params count
The number of characters to be extracted

Returns
A string containing a copy of the specified range of characters, can be empty

Return type
Variable

`StrReplace`(*variable str*, *variable oldSubStr*, *variable newSubStr*)

The StrReplace function searches a string for a specified substring and replaces it with another specified substring.

Params str
The string to be edited

Params oldSubStr
The substring to be replaced by newSubStr

Params newSubStr
The substring to replace oldSubStr

Returns
The number of replaced instances of oldSubStr. Zero if the string is unchanged.

Return type
Variable

`StrReverseFind`(*variable str*, *variable subStr*)

The StrReverseFind function searches a string object for the last match of a sub-string

Params str
The string to be searched

Params subStr
The substring to be searched for

Returns
The zero-based index of the last character in this string that matches the requested substring or characters. -1 if the substring is not found.

Return type
Variable

`StrRight`(*variable str*, *variable count*)

The StrRight function extracts the last (rightmost) characters of a string and returns a copy of the extracted substring. The number of characters extracted is equal to the input variable "count". If "count" is longer than the string, the entire string is returned.

Params str
The input string from which the substring is to be extracted

Params count
The number of characters to be extracted

Returns
A string containing a copy of the specified range of characters. Can be an empty string.

Return type
Variable

`StrSpanExcluding`(*variable str*, *variable subStr*)

The StrSpanExcluding function can be used to search the string for the first occurrence of any character in the specified set subStr. StrSpanExcluding extracts and returns all characters preceding the first occurrence of a character from subStr (in other words, the character from subStr and all characters following it in the string, are not returned). If no character from subStr is found in the string, then StrSpanExcluding returns the entire string.

Params str
The string to be searched

Params subStr
A string containing the set of characters to be searched for

Returns
A sub-string containing characters in the string that are not in subStr, beginning with the first character in the string and ending with the first character found in the string that is also in subStr (that is, starting with the first character in the string and up to but excluding the first character in the string that is found subStr). It returns the entire string if no character in subStr is found in the string.

Return type
Variable

`StrStrToAscii`(*variable character*)

The StrStrToAscii function converts the given character (as a string) into an ASCII code (as an integer)

Params character
The character to convert, inputted as a string

Returns
The ASCII code for the given character as an integer, -1 if the function fails

Return type
Variable

`StrTrimLeft`(*variable str*, *variable character*)

The StrTrimLeft function trims leading whitespace characters from the string (removes newline, space, tab, and user-defined characters)

Params str
The string to trim

Params character
A string containing user-defined characters to be trimmed (may be empty, in which case only newline, space and tabs will be trimmed)

Returns
None

Return type
N/A

`StrTrimRight`(*variable str*, *variable character*)

The StrTrimRight function trims lagging whitespace characters from the string (removes newline, space, tab, and user-defined characters)

Params str
The string to trim

Params character
A string containing user-defined characters to be trimmed (may be empty, in which case only newline, space and tabs will be trimmed)

Returns
None

Return type
N/A

CHAPTER

TWENTY

HSLZIPLIB

https://github.com/theonetruenerd/VenusPackages/blob/main/HSLZipLib.pkg

HSLZipLib is a library that allows you to create zip folders, as well as unzip and extract folders from them. It adds the following four functions:

- *UnpackArchive()*
- *AddItemToArchive()*
- *InitializeArchive()*
- *PackArchive()*

UnpackArchive(*variable i_strArchiveName*, *variable i_strPathOutput*)

The unpack archive function unzips a zip folder and extracts the contents to the specified location.

Params i_strArchiveName
The full name and path to the zip folder you wish to unzip, including the .zip file extension

Params i_strPathOutput
The full name and path of the folder you would like to create when extracting the zip file, with no extension

Returns
Boolean confirming whether unzipping the archive was successful or not

Return type
Boolean

AddItemToArchive(*variable i_strFileOrDirectoryName*, *variable i_strDirectoryPathInArchive*)

This function specifies a file or directory which will be added to the zip file which will be generated with the *PackArchive()* function.

Params i_strFileOrDirectoryName
The path to the file or directory that you wish to include in the zip folder. File extension required for files, no extension required for folders.

Params i_strDirectoryPathInArchive
The path to the desired location of the file within the zip folder.

Returns
Boolean confirming whether adding the file or directory to the archive was successful

Return type
Boolean

`InitializeArchive`(*variable i_strArchiveName*)

This function initializes whatever archive is specified. It is required to call this function for each archive you are interacting with, in advance of calling any of the otehr functions.

Params i_strArchiveName
The name of the desired zip file; can be a file that already exists or one that doesn't exist yet and will be created

Returns
None

Return type
N/A

`PackArchive`()

This function zips/packs whichever archive was most recently initialized.

Returns
Boolean confirming whether zipping the archive was successful or not

Return type
Boolean

CHAPTER

TWENTYONE

HSL_SEQDAILYTOOLS

https://github.com/theonetruenerd/VenusPackages/blob/main/HSL_SeqDailyTools.pkg

The HSL_SeqDailyTools library adds four functions aimed at making some parts of sequence handling slightly easier. The four functions it adds are:

- *CopyPlatePattern96ToTipRack()*
- *CopyPlatePatternToPlate()*
- *GetNumberOfPositionsLeft()*
- *SeqEasyEdit()*

CopyPlatePattern96ToTipRack(*sequence plateSource*, *sequence tipRack*)

This function copies the sequence pattern on a 96 well plate and applies it to a tip rack. It is used to ensure that the 96 head only picks up tips in the wells that the plate is going to be interacting with.

Params plateSource
The sequence pattern on the plate to be copied

Params tipRack
The sequence of the tip rack which will have the template applied to it

Returns
None

Return type
N/A

CopyPlatePatternToPlate(*sequence plateSource*, *sequence plateTarget*)

This function copies the pattern of wells from one plate to another plate. It is useful when a plate is being moved around on deck, or when reagents are being pipetted from one plate into the same locations on another plate.

Params plateSource
The sequence pattern from the plate to be copied

Params plateTarget
The sequence of the second plate which is having the template applied to it

Returns
None

Return type
N/A

GetNumberOfPositionsLeft(*sequence seq*, *variable numberOfPositionsLeft*)

This function calculates the number of positions left in a sequence

Params seq
The sequence which is being checked

Params numberOfPositionsLeft
The output value of how many positions are left in the sequence

Returns
None

Return type
N/A

`SeqEasyEdit`(*sequence seq*, *variable timeout*, *variable dialog_title*, *variable dialog_msg*, *device ML_STAR*)

This function is a simplified version of SeqEdit, with only the parameters that are regularly used. It opens a dialog box from which the user can edit whichever sequence is specified in the input.

Params seq
The sequence which is being edited

Params timeout
How long the dialog window will stay open by itself before it closes and continues the method without editing the sequence

Params dialog_title
The title of the dialog window which pops up

Params dialog_message
The text within the dialog window which pops up

Params ML_STAR
The device used. Should be ML_STAR from the dropdown

Returns
None

Return type
N/A

CHAPTER

TWENTYTWO

IF_AND_IF_OR

https://github.com/theonetruenerd/VenusPackages/blob/main/If_And_If_Or.pkg

The if_and_if_or library adds two functions:

- *if_and()*
- *if_or()*

if_and(*variable iVar1*, *variable iVar2*, *variable iVar3*, *variable iVar4*)

This function checks variable iVar1 against iVar2 and iVar3 against iVar4; if both statements are true then the function returns a 1, otherwise it returns a 0

Params iVar1
Variable to be checked against iVar2

Params iVar2
Variable to be checked against iVar1

Params iVar3
Variable to be checked against iVar4

Params iVar4
Variable to be checked against iVar3

Returns
1 if both true, 0 otherwise

Return type
Variable

if_or(*variable iVar1*, *variable iVar2*, *variable iVar3*, *variable iVar4*)

This function checks variable iVar1 against iVar2 and iVar3 against iVar4; if at least one of the statements is true then the function will return a 1, otherwise it will return a 0

Params iVar1
Variable to be checked against iVar2

Params iVar2
Variable to be checked against iVar1

Params iVar3
Variable to be checked against iVar4

Params iVar4
Variable to be checked against iVar3

Returns
1 if at least one is true, 0 if false

Return type
Variable

CHAPTER

TWENTYTHREE

LABWARE PROPERTIES

https://github.com/theonetruenerd/VenusPackages/blob/main/Labware_Properties.pkg

The labware properties library adds a variety of functions which assist with obtaining the physical data of the labware, as well as things like its ID. The functions it adds are:

- *GetCarrierIDandSiteID_FromLabID()*
- Get_ContainerBaseOffset()
- *Get_ContainerBaseThickness()*
- *Get_Height()*
- *Get_NameAndFileName()*
- *Get_NumberOfColumns()*
- *Get_NumberOfRows()*
- *Get_RackBaseToCoverBase()*
- *Get_StackHeight()*
- *Get_XYZ_deckPosition()*
- *Get_XY_dimensions()*

GetCarrierIDandSiteID_FromLabID(*device io_instrument*, *variable i_labware_ID*, *variable o_carrier_ID*, *variable o_site_ID*)

This function takes outputs the carrier ID and site ID which are associated with the given labware ID. It works on both the Nimbus and the STAR.

Params io_instrument
The instrument being used. Will either be ML_STAR or Nimbus.

Params i_labware_ID
The labware ID from which the other IDs are being pulled.

Params o_carrier_ID
The carrier ID associated with the given labware ID.

Params o_site_ID
The site ID associated with the given labware ID.

Returns
A boolean determining whether the labware ID exists (1) or not (0)

Return type
Boolean

Get_ContainerBaseOFfset(*device io_instrument*, *sequence i_sequenceLabware*, *variable i_sequencePosition*, *variable o_ContainerBaseOffset*)

This function outputs the distance from the rack base to the container base for the labware at the given sequence position.

Params io_instrument
The instrument being used. Will either be ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware being checked.

Params i_sequencePosition
The position ID or index of the sequence being checked. Can be either int or str. An input of 0 will auto-select the first position.

Params o_ContainerBaseOFfset
The distance from the rack base to the container base at the specified position

Returns
None

Return type
N/A

Get_ContainerBaseThickness(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_containerBaseThickness*)

This function outputs the base thickness of the container at the first position of a given sequence.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence asspciated with the labware being checked.

Params o_containerBaseThickness
The thickness of the base of the container being checked.

Returns
None

Return type
N/A

Get_Height(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_labwareHeight*)

This function outputs the height of the labware at the first position of a given sequence. This value is only the labware height, not the absolute Z position.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware being checked.

Params o_labwareHeight
The height of the labware being checked.

Returns
None

Return type
N/A

Get_NameAndFileName(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_viewName*, *variable o_fileName*)

This function outputs the labware view name and the file name associated with it (with path)

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware of interest.

Params o_viewName
The view name of the labware being checked.

Params o_fileName
The file name (with path) of the labware being checked.

Returns
None

Return type
N/A

Get_NumberOfColumns(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_labwareColumns*)

This function outputs the number of columns defined in the labware from the first position of a given sequence.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware of interest.

Params o_labwareColumns
The number of columns defined in the labware being checked.

Returns
None

Return type
N/A

Get_NumberOfRows(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_labwareColumns*)

This function outputs the number of rows defined in the labware at the first position of a given sequence. The variable and function description both say columns; this is incorrect.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware being checked.

Params o_labwareColumns
The number of rows defined in the labware being checked.

Returns
None

Return type
N/A

Get_RackBaseToCoverBase(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_RackBaseToCoverBase_Height*)

This function outputs the height from the base of the rack to the base of the cover/lid of the labware at the first position of a given sequence.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware being checked.

Params o_RackBaseToCoverBase_Height
The distance from the rack base to the cover base.

Returns
None

Return type
N/A

`Get_StackHeight`(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_labwareStackHeight*)

This function outputs the stack height of the specified labware at the first position of a given sequence.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware being checked.

Params o_labwareStackHeight
The stack height of the labware being checked, or the covered stack height if the labware is lidded.

Returns
None

Return type
N/A

`Get_XYZ_deckPosition`(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_labware_deckPosition_X*, *variable o_labware_deckPosition_Y*, *variable o_labware_deckPosition_Z*)

This function returns the X, Y and Z coordinates of the upper left well of the specified labware at the first position of a given sequence. The description of this function says it only does the X and Y coordinates, this is incorrect.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware being checked.

Params o_labware_deckPosition_X
The X coordinate of the labware on the deck.

Params o_labware_deckPosition_Y
The Y coordinate of the labware on the deck.

Params o_labware_deckPosition_Z
The Z coordinate of the labware on the deck.

Returns
None

Return type
N/A

Get_XY_dimensions(*device io_instrument*, *sequence i_sequenceLabware*, *variable o_X_width*, *variable o_Y_depth*)

This function outputs the X (width) and Y (depth) dimensions of the specified labware at the first position of a given sequence.

Params io_instrument
The instrument being used. Will be either ML_STAR or Nimbus.

Params i_sequenceLabware
The sequence associated with the labware being checked.

Params o_X_width
The width of the labware being checked.

Params o_Y_depth
The depth of the labware being checked.

Returns
None

Return type
N/A

CHAPTER

TWENTYFOUR

LOOKUP

https://github.com/theonetruenerd/VenusPackages/blob/main/Lookup.pkg

The lookup library provides one function which allows you to locate items in an array and get their index.

- *Lookup()*

Lookup(*array array*, *variable item*)

Input an array and a value to look up. If the value occurs in the array, it will return the 1-based index. If it doesn't occur in the array, it will return a 0.

Params array
The array to search

Params item
The variable to search for

Returns
1-based index if found, 0 if not found

Return type
Variable

CHAPTER

TWENTYFIVE

PIPETTING (FROM HSLEXTENSIONS)

https://github.com/theonetruenerd/VenusPackages/blob/main/Pipetting.pkg

The pipetting library from HSL Extensions adds functions related to pipetting, primarily to do with sequences and channel patterns. The functions it adds are:

- *CreateMixedChannelPattern()*
- *CreateOrderedChannelPattern()*
- *ParseChannelPattern()*
- *Plate384PositionNumberToPositionText()*
- Plate384PositionTextToPositionNumber()
- *Plate96PositionNumberToPositionText()*
- Plate96PositionTextToPositionNumber()

CreateMixedChannelPattern(*array i_arrUseChannel*, *variable i_intTotalNumberOfChannels*)

This function generates a "mixed" channel pattern. The channel pattern generated will be based on the input array, and then will be filled with "0"s up to the total number of channels (normally 8). If the array size is greater than the total number of channels inputted, the channel pattern will be limited to whatever the total number of channels inputted is. E.g. if the array is [1,1,1,1,0,0,1,0,0,0,1] and the input channel number is 8, the channel pattern generated will be "11110010".

Params i_arrUseChannel
A boolean array of which channels should be used.

Params i_intTotalNumberOfChannels
The total number of channels being used.

Returns
The channel pattern generated by the function

Return type
String

CreateOrderedChannelPattern(*variable i_intNumberOfUsedChannels*, *variable i_intTotalNumberOfChannels*)

This function creates an ordered channel pattern, generating a string with as many "1"s in it as the inputted number of used channels, and adding "0"s until the total length of the channel pattern is equal to the total number of channels inputted. If the total number of used channels is greater than the total number of channels, the channel pattern will be truncated to the total number of channels. E.g. if the number of used channels is 3 and the total number of channels is 8, the channel pattern generated will be "11100000". If these numbers were reversed, the channel pattern generated would simply be "111". :params i_intNumberOfUsedChannels: The number of "active" channels wishing to be used in the channel pattern :params i_intTotalNumberOfChannels: The max number of channels available (usually 8 on a STAR) :type i_intNumberOfUsedChannels: Variable :type i_intTotalNumberOfChannels: Variable :return: The channel pattern generated by the function :rtype: String

ParseChannelPattern(*variable i_strChannelPattern*)

This function parses a channel pattern given as a string and creates a boolean array of the channels being used. If the input has the wrong type, or contains not allowed characters, the result is an empty array.

Params i_strChannelPattern
The channel pattern to be parsed

Returns
The array of booleans with the channels being used in the channel pattern

Return type
Array

Plate384PositionNumberToPositionText(*variable i_intPositionNumber*, *variable o_strPositionText*)

This function converts position number to position text for a 384 well plate. E.g. if given a 1 it will convert it to A1, if given a 16 it will convert it to P1, etc.

Params i_intPositionNumber
The position number to be converted. Any int in the range of 1-384.

Params o_strPositionText
The position text which will be the output. Will be a string of any of the well identifiers on a 384 well plate (i.e. A1-P24)

Returns
A boolean of whether the conversion was successful

Return type
Boolean

Plate384PositionNumberToPositionText(*variable i_strPositionText*, *variable o_intPositionNumber*)

This function converts position text to position number for a 384 well plate. E.g. if given A1 it will convert it to a 1, if given P1 it will convert it to a 16, etc.

Params i_strPositionText
The position text to be converted. Will be a string of any of the well identifiers on a 384 well plate (i.e. A1-P24)

Params o_intPositionNumber
The position number which will be the output. Any int in the range of 1-384.

Returns
A boolean of whether the conversion was successful

Return type
Boolean

Plate96PositionNumberToPositionText(*variable i_intPositionNumber*, *variable o_strPositionText*)

This function converts position number to position text for a 96 well plate. E.g. if given a 1 it will convert it to A1, if given an 8 it will convert it to H1, etc.

Params i_intPositionNumber
The position number to be converted. Any int in the range of 1-96.

Params o_strPositionText
The position text which will be the output. Will be a string of any of the well identifiers on a 96 well plate (i.e. A1-H12)

Returns
A boolean of whether the conversion was successful

Return type
Boolean

`Plate96PositionNumberToPositionText`(*variable i_strPositionText*, *variable o_intPositionNumber*)

This function converts position text to position number for a 96 well plate. E.g. if given A1 it will convert it to a 1, if given H1 it will convert it to a 8, etc.

Params i_strPositionText
The position text to be converted. Will be a string of any of the well identifiers on a 96 well plate (i.e. A1-H12)

Params o_intPositionNumber
The position number which will be the output. Any int in the range of 1-96.

Returns
A boolean of whether the conversion was successful

Return type
Boolean

CHAPTER

TWENTYSIX

STAR_CHANNEL_TOOLS

https://github.com/theonetruenerd/VenusPackages/blob/main/STAR_Channel_Tools.pkg

The STAR_Channel_Tools submethod library adds a variety of functions which are related to the use of the 8 channels. The following functions are added:

- *CHAN_ACCESS_Sort1Sequence()*
- *CHAN_ACCESS_Sort1Sequence1Array()*
- *CHAN_ACCESS_Sort1Sequence2Arrays()*
- *CHAN_ACCESS_Sort2Sequences()*
- *CHAN_ACCESS_Sort2Sequences1Array()*
- *CHAN_ACCESS_Sort2Sequences2Arrays()*
- *LIQUID_LEVEL_GetLiquidLevelHeight()*
- *LIQUID_LEVEL_MeasureLiquidMulti()*
- *LIQUID_LEVEL_MeasureLiquidSingle()*
- *LIQUID_LEVEL_ReturnVolumesFromLiquidLevel()*
- *MOVE_ChannelsToSequencePosition()*
- *MOVE_ChannelsToSequencePosition_5mL()*
- *MOVE_CheckPlateWithTwoChannels()*
- *MOVE_InitDispenseDrive()*
- *MOVE_InitDispenseDrive_5mL()*
- *PLATE_STACK_CountPlateStacks()*
- *QUERY_GetChannelPosition()*
- *QUERY_GetChannelPosition_5mL()*
- *QUERY_GetTipPresentState()*
- *QUERY_GetTipPresentState_5mL()*
- *QUERY_GetTipVolume()*
- QUERY_GetTIpVolume_5mL()
- *SPLIT_WELLS_AddContainersToWell()*
- SPLIT_WELLS_RemoveContainers()
- *TRAVEL_LANES_MoveChannelsToTravelLanes()*

- *TRAVEL_LANES_MoveChannelsToTravelLanes_5mL()*
- *TRAVEL_LANES_MoveChannelsToYPosition()*
- *TRAVEL_LANES_MoveChannelsToYPosition_5mL()*
- *TRAVEL_LANES_MoveChannelsWithTravelLanes()*
- *TRAVEL_LANES_MoveChannelsWithTravelLanes_5mL()*
- *TRAVEL_LANES_SingleSource_ChannelDisplacement()*
- *TRAVEL_LANES_SingleSource_ChannelDisplacement_5mL()*

CHAN_ACCESS_Sort1Sequence(*device ML_STAR*, *sequence io_Sequence_to_Sort*, *variable i_Channel_Type*, *boolean i_Sort_by_Labware*, *boolean i_Sort_by_XY*, *boolean i_Sort_for_Channel_Raster*, *variable i_Max_Channel*, *sequence o_Sorted_Sequence*, *variable o_Channel_Pattern*)

This submethod takes an input sequence and sorts it based on the input parameters of labware, position, and raster. Once sorted, the submethod will choose a position that the current channel can access up to the maximum. If the current channel cannot access the position, it will skip it and move to the next available position. If the current cannot access any more positions, that channel will be skipped. Make sure the channel use setting is set to "All Sequence Positions" in the pipettting step, otherwise the sequence and channel pattern will not line up.

Params ML_STAR
The ML_STAR itself, which will be the only option in the dropdown.

Params io_Sequence_to_Sort
The input sequence to be sorted.

Params i_Channel_Type
The channel type associated with the pipetting step (1mL, 5mL, labware handler). 0 = 1mL, 1 = 5mL, 2 = Labware handler.

Params i_Sort_by_Labware
A boolean determining whether the sequence is to be sorted by labware in ascending order

Params i_Sort_by_XY
A boolean determining whether the sequence is to be sorted by position (X ascending, Y descending)

Params i_Sort_for_Channel_Raster
A boolean determining whether the next position will be at least the raster distance unless no other positions are available

Params i_Max_Channel
The maximum channel that you want to be used from 1 to 16. 0 turns this option off and the maximum number of channels will be used.

Params o_Sorted_Sequence
The outputted sorted sequence for the pipetting step

Params o_Channel_Pattern
The outputted channel pattern for the pipetting step

Returns
The number of sequence positions remaining in the sequence

Return type
Variable

CHAN_ACCESS_Sort1Sequence1Array(*device ML_STAR*, *sequence io_Sequence_to_Sort*, *array io_Array_of_Variables*, *variable i_Channel_Type*, *boolean i_Sort_by_Labware*, *boolean i_Sort_by_XY*, *boolean i_Sort_for_Channel_Raster*, *variable i_Max_Channel*, *sequence o_Sorted_Sequence*, *array o_Sorted_Array*, *variable o_Channel_Pattern*)

This submethod takes in input sequence and sorts it by the conditions given below. After sorting, the submethod will choose a position that the current channel can access up to the maximum. If the current channel cannot access the position, it wil skip it and move to the next available position. If the current channel cannot access any more positions, that channel will be skipped. The array will be sorted with the sequence. The array and the sequence must be the same size. Make sure the channel use setting is set to "All sequence positions" otherwise the sequence and channel pattern will not line up.

Params ML_STAR
The ML_STAR itself, which will be the only option in the dropdown.

Params io_Sequence_to_Sort
The input sequence to be sorted.

Params io_Array_of_Variables
The array to be sorted with the sequence. Used positions will be removed from the array.

Params i_Channel_Type
The channel type associated with the pipetting step (1mL, 5mL, labware handler). 0 = 1mL, 1 = 5mL, 2 = Labware handler.

Params i_Sort_by_Labware
A boolean determining whether the sequence is to be sorted by labware in ascending order

Params i_Sort_by_XY
A boolean determining whether the sequence is to be sorted by position (X ascending, Y descending)

Params i_Sort_for_Channel_Raster
A boolean determining whether the next position will be at least the raster distance unless no other positions are available

Params i_Max_Channel
The maximum channel that you want to be used from 1 to 16. 0 turns this option off and the maximum number of channels will be used.

Params o_Sorted_Sequence
The outputted sorted sequence for the pipetting step

Params o_Sorted_Array
The outputted sorted array which matches the sequence

Params o_Channel_Pattern
The outputted channel pattern for the pipetting step

Returns
The number of sequence positions remaining in the sequence

Return type
Variable

CHAN_ACCESS_Sort1Sequence2Arrays(*device ML_STAR*, *sequence io_Sequence_to_Sort*, *array io_Array_of_Variables*, *array io_Array_of_Variables2*, *variable i_Channel_Type*, *boolean i_Sort_by_Labware*, *boolean i_Sort_by_XY*, *boolean i_Sort_for_Channel_Raster*, *variable i_Max_Channel*, *sequence o_Sorted_Sequence*, *array o_Sorted_Array*, *array o_Sorted_Array2*, *variable o_Channel_Pattern*)

This submethod takes in input sequence and sorts it by the conditions given below. After sorting, the submethod will choose a position that the current channel can access up to the maximum. If the current channel cannot access the position, it wil skip it and move to the next available position. If the current channel cannot access any more positions, that channel will be skipped. The arrays will be sorted with the sequence. The arrays and the sequence must be the same size. Make sure the channel use setting is set to "All sequence positions" otherwise the sequence and channel pattern will not line up.

Params ML_STAR
The ML_STAR itself, which will be the only option in the dropdown.

Params io_Sequence_to_Sort
The input sequence to be sorted.

Params io_Array_of_Variables
The first array to be sorted with the sequence. Used positions will be removed from the array.

Params io_Array_of_Variables2
The second array to be sorted with the sequence. Used positions will be removed from the array.

Params i_Channel_Type
The channel type associated with the pipetting step (1mL, 5mL, labware handler). 0 = 1mL, 1 = 5mL, 2 = Labware handler.

Params i_Sort_by_Labware
A boolean determining whether the sequence is to be sorted by labware in ascending order

Params i_Sort_by_XY
A boolean determining whether the sequence is to be sorted by position (X ascending, Y descending)

Params i_Sort_for_Channel_Raster
A boolean determining whether the next position will be at least the raster distance unless no other positions are available

Params i_Max_Channel
The maximum channel that you want to be used from 1 to 16. 0 turns this option off and the maximum number of channels will be used.

Params o_Sorted_Sequence
The outputted sorted sequence for the pipetting step

Params o_Sorted_Array
The outputted sorted first array which matches the sequence

Params o_Sorted_Array2
The outputted sorted second array which matches the sequence

Params o_Channel_Pattern
The outputted channel pattern for the pipetting step

Returns
The number of sequence positions remaining in the sequence

Return type
Variable

`CHAN_ACCESS_Sort2Sequences`(*device ML_STAR*, *sequence io_Sequence_to_Sort*, *sequence io_Sequence_to_Sort2*, *variable i_Channel_Type*, *boolean i_Sort_by_Labware*, *boolean i_Sort_by_XY*, *boolean i_Sort_for_Channel_Raster*, *variable i_Max_Channel*, *sequence o_Sorted_Sequence*, *sequence o_Sorted_Sequence2*, *variable o_Channel_Pattern*)

This sub method takes the in input sequences and sorts them by the conditions given below. After sorting, the sub will choose a position that the current channel can access in both sequence positions up to the maximum. If the current channel cannot access the positions, it wil skip it and move to the next available position. If the current channel cannot access any more positions, that channel will be skipped. Make sure the channel use setting is set to "All sequence positions" otherwise the sequence and channel pattern will not line up. The first sequence is the driving sequence and the second sequence will be adjusted by the first sort.

Params ML_STAR
The ML_STAR itself, which will be the only option in the dropdown.

Params io_Sequence_to_Sort
The first input sequence to be sorted.

Params io_Sequence_to_Sort2
The second input sequence to be sorted.

Params i_Channel_Type
The channel type associated with the pipetting step (1mL, 5mL, labware handler). 0 = 1mL, 1 = 5mL, 2 = Labware handler.

Params i_Sort_by_Labware
A boolean determining whether the sequence is to be sorted by labware in ascending order

Params i_Sort_by_XY
A boolean determining whether the sequence is to be sorted by position (X ascending, Y descending)

Params i_Sort_for_Channel_Raster
A boolean determining whether the next position will be at least the raster distance unless no other positions are available

Params i_Max_Channel
The maximum channel that you want to be used from 1 to 16. 0 turns this option off and the maximum number of channels will be used.

Params o_Sorted_Sequence
The outputted second sorted sequence for the pipetting step

Params o_Sorted_Sequence2
The outputted second sorted sequence for the pipetting step

Params o_Channel_Pattern
The outputted channel pattern for the pipetting step

Returns
The number of sequence positions remaining in the sequence

Return type
Variable

`CHAN_ACCESS_Sort2Sequences1Array`(*device ML_STAR*, *sequence io_Sequence_to_Sort*, *sequence io_Sequence_to_Sort2*, *array io_Array_of_Variables*, *variable i_Channel_Type*, *boolean i_Sort_by_Labware*, *boolean i_Sort_by_XY*, *boolean i_Sort_for_Channel_Raster*, *variable i_Max_Channel*, *sequence o_Sorted_Sequence*, *sequence o_Sorted_Sequence2*, *array o_Sorted_Array*, *variable o_Channel_Pattern*)

This sub method takes the in input sequences and sorts them by the conditions given below. After sorting, the sub will choose a position that the current channel can access in both sequence positions up to the maximum. If the current channel cannot access the positions, it wil skip it and move to the next available position. If the current channel cannot access any more positions, that channel will be skipped. Make sure the channel use setting is set

to "All sequence positions" otherwise the sequence and channel pattern will not line up. The first sequence is the driving sequence and the second sequence will be adjusted by the first sort.

Params ML_STAR
The ML_STAR itself, which will be the only option in the dropdown.

Params io_Sequence_to_Sort
The first input sequence to be sorted.

Params io_Sequence_to_Sort2
The second input sequence to be sorted.

Params i_Channel_Type
The channel type associated with the pipetting step (1mL, 5mL, labware handler). 0 = 1mL, 1 = 5mL, 2 = Labware handler.

Params i_Sort_by_Labware
A boolean determining whether the sequence is to be sorted by labware in ascending order

Params i_Sort_by_XY
A boolean determining whether the sequence is to be sorted by position (X ascending, Y descending)

Params i_Sort_for_Channel_Raster
A boolean determining whether the next position will be at least the raster distance unless no other positions are available

Params i_Max_Channel
The maximum channel that you want to be used from 1 to 16. 0 turns this option off and the maximum number of channels will be used.

Params o_Sorted_Sequence
The outputted second sorted sequence for the pipetting step

Params o_Sorted_Sequence2
The outputted second sorted sequence for the pipetting step

Params o_Sorted_Array
The sorted array. The array will be the size of the maximum channel.

Params o_Channel_Pattern
The outputted channel pattern for the pipetting step

Returns
The number of sequence positions remaining in the sequence

Return type
Variable

`CHAN_ACCESS_Sort2Sequences2Arrays`(*device ML_STAR*, *sequence io_Sequence_to_Sort*, *sequence io_Sequence_to_Sort2*, *array io_Array_of_Variables*, *array io_Array_of_Variables2*, *variable i_Channel_Type*, *boolean i_Sort_by_Labware*, *boolean i_Sort_by_XY*, *boolean i_Sort_for_Channel_Raster*, *variable i_Max_Channel*, *sequence o_Sorted_Sequence*, *sequence o_Sorted_Sequence2*, *array o_Sorted_Array*, *array o_Sorted_Array2*, *variable o_Channel_Pattern*)

This sub method takes the in input sequences and sorts them by the conditions given below. After sorting, the sub will choose a position that the current channel can access in both sequence positions up to the maximum. If the current channel cannot access the positions, it wil skip it and move to the next available position. If the current channel cannot access any more positions, that channel will be skipped. Make sure the channel use setting is set

to "All sequence positions" otherwise the sequence and channel pattern will not line up. The first sequence is the driving sequence and the second sequence will be adjusted by the first sort.

Params ML_STAR
The ML_STAR itself, which will be the only option in the dropdown.

Params io_Sequence_to_Sort
The first input sequence to be sorted. This positions removed will be removed from this sequence.

Params io_Sequence_to_Sort2
The second input sequence to be sorted. The positions used will be removed from this sequence.

Params io_Array_of_Variables
The input array of variables which will be sorted with the first sequence. The positions used will be removed from the array.

Params io_Array_of_Variables2
The input array of variables which will be sorted with the second sequence. The positions used will be removed from the array.

Params i_Channel_Type
The channel type associated with the pipetting step (1mL, 5mL, labware handler). 0 = 1mL, 1 = 5mL, 2 = Labware handler.

Params i_Sort_by_Labware
A boolean determining whether the sequence is to be sorted by labware in ascending order

Params i_Sort_by_XY
A boolean determining whether the sequence is to be sorted by position (X ascending, Y descending)

Params i_Sort_for_Channel_Raster
A boolean determining whether the next position will be at least the raster distance unless no other positions are available

Params i_Max_Channel
The maximum channel that you want to be used from 1 to 16. 0 turns this option off and the maximum number of channels will be used.

Params o_Sorted_Sequence
The outputted second sorted sequence for the pipetting step

Params o_Sorted_Sequence2
The outputted second sorted sequence for the pipetting step

Params o_Sorted_Array
The sorted array. The array will be the size of the maximum channel.

Params o_Sorted_Array2
The second sorted array. The array will be the size of the maximum channel.

Params o_Channel_Pattern
The outputted channel pattern for the pipetting step

Returns
The number of sequence positions remaining in the sequence

Return type
Variable

`LIQUID_LEVEL_GetLiquidLevelHeight`(*device ML_STAR*, *variable i_str_LiquidLevelReturn*, *sequence i_seq_Labware*, *variable i_int_Channel*, *variable o_flt_LiquidHeight*)

This function will return the liquid level height relative to the container bottem.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown.

Params i_str_LiquidLevelReturn
The return value of the liquid level detect from the pipetting step.

Params i_seq_Labware
The input sequence from which the height is to be determined

Params i_int_Channel
The channel which will be used to determine the liquid level height

Params o_flt_LiquidHeight
The detected liquid level height

Returns
None

Return type
N/A

`LIQUID_LEVEL_MeasureLiquidMulti`(*device ML_STAR*, *array i_arrseq_FullReservoirSequences*, *sequence i_seq_TipsToUse*, *sequence i_seq_TipWaste*, *variable i_str_TipCounter*, *variable i_int_LLD_Sensitivity*, *variable i_bool_ConvertTouL*, *array o_arr_VolumesMeasured*)

This function will pick up the desired tips and will measure the liquid level at the center most well of the desired reservoirs and will return the volumes in uL. Ensure the sequence used for the reservoir contains the FULL number of positions of the reservoir, otherwise volume estimation will be incomplete! The tip types supported are: 50uL Filter, 50uL, 300uL Filter, 300uL, 1000uL Filter, and 1000uL.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown.

Params i_arrseq_FullReservoirSequences
The array of full reservoir sequences to be checked.

Params i_seq_TipsToUse
The sequence of the tips to be used to measure the liquid level.

Params i_seq_TipWaste
The sequence of the waste used to eject the tips.

Params i_str_TipCounter
The tip counter to be used for the tips on pickup. Place “” if no tip counter will be used.

Params i_int_LLD_Sensitivity
Integer representing the liquid level sensitivity to be used. 1 is very high, 4 is low, 5 is from labware definition.

Params i_bool_ConvertTouL
Boolean determining whether it converts to uL (hslTrue) or stays as mL (hslFalse).

Params o_arr_VolumesMeasured
The array of volumes that were measured.

Returns
Whether the measurement has been successful or not

Return type
Boolean

LIQUID_LEVEL_MeasureLiquidSingle(*device ML_STAR*, *sequence i_seq_FullReservoirSequence*, *sequence i_seq_TipsToUse*, *sequence i_seq_TipWaste*, *variable i_str_TipCounter*, *variable i_bool_IncrementTipSequence*, *variable i_int_LLD_Sensitivity*, *variable i_bool_ConvertTouL*, *variable o_flt_VolumeMeasured*)

This function will pick up the desired tip and will measure the liquid level at the center most well of the desired reservoir and will return the volume in uL. Ensure the sequence used for the reservoir contains the FULL number of positions of the reservoir, otherwise volume estimation will be incomplete! The tip types supported are: 50uL Filter, 50uL, 300uL Filter, 300uL, 1000uL Filter, and 1000uL.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown.

Params i_seq_FullReservoirSequences
The full reservoir sequence to be checked.

Params i_seq_TipsToUse
The sequence of the tips to be used to measure the liquid level.

Params i_seq_TipWaste
The sequence of the waste used to eject the tips.

Params i_str_TipCounter
The tip counter to be used for the tips on pickup. Place “” if no tip counter will be used.

Params i_bool_IncrementTipSequence
Boolean determining whether the tip sequence should be incremented after pickup or not.

Params i_int_LLD_Sensitivity
Integer representing the liquid level sensitivity to be used. 1 is very high, 4 is low, 5 is from labware definition.

Params i_bool_ConvertTouL
Boolean determining whether it converts to uL (hslTrue) or stays as mL (hslFalse).

Params o_flt_VolumeMeasured
A float of the volume that was measured.

Returns
Whether the measurement has been successful or not

Return type
Boolean

LIQUID_LEVEL_ReturnVolumesFromLiquidLevel(*device ML_STAR*, *variable i_str_PipettingReturn*, *variable i_str_LiquidLevelReturn*, *variable i_bool_ConvertTouL*, *array o_arr_VolumesMeasured*)

This function will return the volumes that were measured from a previous aspiration step.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_str_PipettingReturn
The return value of the pipetting step to measure liquid level

Params i_str_LiquidLevelReturn
The return value of the liquid level detect

Params i_bool_ConvertTouL
Boolean determining whether it converts to uL (hslTrue) or stays as mL (hslFalse)

Params o_arr_VolumesMeasured
An array of the volumes which were measured for each channel

Returns
None

Return type
N/A

`MOVE_ChannelsToSequencePosition`(*device ML_STAR*, *variable i_str_ChPattern*, *sequence i_seq_Positions*, *variable i_flt_ZHeight*)

This function moves the 1mL channels to set positions. This function will only move the channels that are activated by the channel pattern. The positions in the sequence will skip over the positions where the channel is turned off.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_str_ChPattern
The channel pattern of the pipettes to move

Params i_seq_Positions
The X Y positions to move the channels to

Params i_flt_ZHeight
The Z positions to end the channels in

Returns
None

Return type
N/A

`MOVE_ChannelsToSequencePosition_5mL`(*device ML_STAR*, *variable i_str_ChPattern*, *sequence i_seq_Positions*, *variable i_flt_ZHeight*)

This function moves the 5mL channels to set positions. This function will only move the channels that are activated by the channel pattern. The positions in the sequence will skip over the positions where the channel is turned off.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_str_ChPattern
The channel pattern of the pipettes to move

Params i_seq_Positions
The X Y positions to move the channels to

Params i_flt_ZHeight
The Z positions to end the channels in

Returns
None

Return type
N/A

`MOVE_CheckPlateWithTwoChannels`(*device ML_STAR*, *variable i_int_FrontMostChannel*, *sequence i_seq_PlateToCheck*, *variable i_flt_TapWidth*)

This function will take the front most channel and the channel behind it to tap the labware at the sequence position. This function requires both channels to either have a tip or tool loaded on them before calling.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_FrontMostChannel
The front-most channel being used to check plates

Params i_seq_PlateToCheck
The sequence position to perform a tap to check for plate existence

Params i_flt_TapWidth
The distance in mm for the channels to be separated before tapping

Returns
Boolean determining whether plate was found (hslTrue) or not (hslFalse)

Return type
Boolean

`MOVE_InitDispenseDrive`(*device ML_STAR*, *variable i_int_ChannelNumber*)

This function moves the dispense drive for the specified 1mL channel to its home position

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChannelNumber
The channel to initialise the dispense drive

Returns
None

Return type
N/A

`MOVE_InitDispenseDrive_5mL`(*device ML_STAR*, *variable i_int_ChannelNumber*)

This function moves the dispense drive for the specified 5mL channel to its home position

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChannelNumber
The channel to initialise the dispense drive

Returns
None

Return type
N/A

`PLATE_STACK_CountPlateStacks`(*device ML_STAR*, *sequence i_seq_PlateStack_Full*, *sequence o_seq_PlateStack_Count*, *variable o_int_PlateCount*)

This function will use the channels to measure the number of plates in a plate stack

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_seq_PlateStack_Full
The full sequence of the plate stack to measure

Params o_seq_PlateStack_Count
The sequence of the plate stack measured

Params o_int_PlateCount
The total number of plates measured in the plate stack

Returns
None

Return type
N/A

QUERY_GetChannelPosition(*device ML_STAR*, *variable i_int_ChNumber*, *variable o_flt_XCoord*, *variable o_flt_YCoord*, *variable o_flt_ZCoord*)

This function will return the current coordinate of the specified 1mL channel

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChNumber
The number of the channel whose position is being checked

Params o_flt_XCoord
The X coordinate of the channel

Params o_flt_YCoord
The Y coordinate of the channel

Params o_flt_ZCoord
The Z coordinate of the channel

Returns
None

Return type
N/A

QUERY_GetChannelPosition_5mL(*device ML_STAR*, *variable i_int_ChNumber*, *variable o_flt_XCoord*, *variable o_flt_YCoord*, *variable o_flt_ZCoord*)

This function will return the current coordinate of the specified 5mL channel

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChNumber
The number of the channel whose position is being checked

Params o_flt_XCoord
The X coordinate of the channel

Params o_flt_YCoord
The Y coordinate of the channel

Params o_flt_ZCoord
The Z coordinate of the channel

Returns
None

Return type
N/A

QUERY_GetTipPresentState(*device ML_STAR*, *variable i_int_ChNumber*, *variable o_bln_TipPresent*)

This function outputs true if a tip is loaded and false if a tip is not loaded on the specified 1mL channel

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChNumber
The number of the channel which is being checked

Params o_bln_TipPresent
A boolean output of whether the tip is present (hslTrue) or not (hslFalse)

Returns
None

Return type
N/A

`QUERY_GetTipPresentState_5mL`(*device ML_STAR*, *variable i_int_ChNumber*, *variable o_bln_TipPresent*)

This function outputs true if a tip is loaded and false if a tip is not loaded on the specified 5mL channel

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChNumber
The number of the channel which is being checked

Params o_bln_TipPresent
A boolean output of whether the tip is present (hslTrue) or not (hslFalse)

Returns
None

Return type
N/A

`QUERY_GetTipVolume`(*device ML_STAR*, *variable i_int_ChNumber*, *variable o_flt_MaxVolume*, *variable o_flt_CurrentChannelVolume*)

This function queries the specified 1mL channel to get the max and the current channel volume. This volume includes the air gap and the conversion made by the correction curve.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChNumber
The number of the channel which is being checked

Params o_flt_MaxVolume
The maximum volume of the channel

Params o_flt_CurrentChannelVolume
The current volume in the channel

Returns
None

Return type
N/A

`QUERY_GetTipVolume`(*device ML_STAR*, *variable i_int_ChNumber*, *variable o_flt_MaxVolume*, *variable o_flt_CurrentChannelVolume*)

This function queries the specified 5mL channel to get the max and the current channel volume. This volume includes the air gap and the conversion made by the correction curve.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_int_ChNumber
The number of the channel which is being checked

Params o_flt_MaxVolume
The maximum volume of the channel

Params o_flt_CurrentChannelVolume
The current volume in the channel

Returns
None

Return type
N/A

SPLIT_WELLS_AddContainersToWell(*device ML_STAR*, *sequence i_seq_SequenceToSplit*, *variable i_int_SequenceIndex*, *variable i_int_MaxSplitNumber*, *sequence io_seq_SplitSequence*)

This function will split a well into a sequence of multiple containers, each of which can be aspirated from individually.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_seq_SequenceToSplit
The sequence that contains the well to be split into multiple containers

Params i_int_SequenceIndex
The index of the sequence that is to be split into multiple containers

Params i_int_MaxSplitNumber
The number of times that the selected container will be split. Won't exceed the width of the well

Params io_seq_SplitSequence
The sequence containing the split wells. Will append to the end of the sequence.

Returns
None

Return type
N/A

SPLIT_WELLS_Remove_Containers(*device ML_STAR*, *variable i_bool_UpdateVolumes*)

This function will remove the containers added by the split wells function

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_bool_UpdateVolume
A boolean determining whether to update the split source volume with the volume set in sample tracking.

Returns
None

Return type
N/A

TRAVEL_LANES_MoveChannelsToTravelLanes(*device ML_STAR*)

This function will move the 1mL channels to the travel lanes

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Returns
None

Return type
N/A

TRAVEL_LANES_MoveChannelsToTravelLanes_5mL(*device ML_STAR*)

This function will move the 5mL channels to the travel lanes

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Returns
None

Return type
N/A

TRAVEL_LANES_MoveChannelsToYPosition(*device ML_STAR*, *sequence i_seq_TargetSequence*, *variable i_flt_XOffsetFromOrigin*)

Parameters include the Instrument, a destination sequence, and whether or not a shift in the x-direction is wanted before moving the 1mL channels in y-direction (to doubly ensure no crossver of open wells). Channels will be moved to their Y coordinates at the X origin of the next labware in the sequence plus the X offset. The channels will move to the current position of the input Sequence.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_seq_TargetSequence
The sequence to move the channels to

Params i_flt_XOffsetFromOrigin
The distance from the plate's origin to start the channel approach

Returns
None

Return type
N/A

TRAVEL_LANES_MoveChannelsToYPosition_5mL(*device ML_STAR*, *sequence i_seq_TargetSequence*, *variable i_flt_XOffsetFromOrigin*)

Parameters include the Instrument, a destination sequence, and whether or not a shift in the x-direction is wanted before moving the 5mL channels in y-direction (to doubly ensure no crossver of open wells). Channels will be moved to their Y coordinates at the X origin of the next labware in the sequence plus the X offset. The channels will move to the current position of the input Sequence.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_seq_TargetSequence
The sequence to move the channels to

Params i_flt_XOffsetFromOrigin
The distance from the plate's origin to start the channel approach

Returns
None

Return type
N/A

TRAVEL_LANES_MoveChannelsWithTravelLanes(*device ML_STAR*, *sequence i_seq_TargetSequence*, *variable i_flt_XOffsetFromOrigin*)

This command is designed to shift all the 1mL channels on the instrument to either the front and/or the back of the instrument in a layout that ensures tips will not crossover any carriers. Parameters include the Instrument, a destination sequence, and whether or not a shift in the x-direction is wanted before moving the channels in

y-direction (to doubly ensure no crossver of open wells). Channels will be moved to their Y coordinates at the X origin of the next labware in the sequence plus the X offset. The channels will move to the current position of the input Sequence.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_seq_TargetSequence
The sequence to move the channels to

Params i_flt_XOffsetFromOrigin
The distance from the plate's origin to start the channel approach

Returns
None

Return type
N/A

`TRAVEL_LANES_MoveChannelsWithTravelLanes_5mL`(*device ML_STAR*, *sequence i_seq_TargetSequence*, *variable i_flt_XOffsetFromOrigin*)

This command is designed to shift all the 5mL channels on the instrument to either the front and/or the back of the instrument in a layout that ensures tips will not crossover any carriers. Parameters include the Instrument, a destination sequence, and whether or not a shift in the x-direction is wanted before moving the channels in y-direction (to doubly ensure no crossver of open wells). Channels will be moved to their Y coordinates at the X origin of the next labware in the sequence plus the X offset. The channels will move to the current position of the input Sequence.

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_seq_TargetSequence
The sequence to move the channels to

Params i_flt_XOffsetFromOrigin
The distance from the plate's origin to start the channel approach

Returns
None

Return type
N/A

`TRAVEL_LANES_SingleSource_ChannelDisplacement`(*device ML_STAR*, *variable i_strStepReturn*)

This command is designed to move unused 1mL channels to the back of the instrument when pipetting one well at a time while more than one channel has tips/liquid. It requires the Instrument type and Step Return variable from the Aspirate/Dispense step (this is used to determine which channel needs to be moved).

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_strStepReturn
The bound step return from given desired pipetting step

Returns
None

Return type
N/A

`TRAVEL_LANES_SingleSource_ChannelDisplacement_5mL`(*device ML_STAR*, *variable i_strStepReturn*)

This command is designed to move unused 5mL channels to the back of the instrument when pipetting one well at a time while more than one channel has tips/liquid. It requires the Instrument type and Step Return variable from the Aspirate/Dispense step (this is used to determine which channel needs to be moved).

Params ML_STAR
The ML_STAR itself, will be the only option in the dropdown

Params i_strStepReturn
The bound step return from given desired pipetting step

Returns
None

Return type
N/A

CHAPTER

TWENTYSEVEN

STRTOKENIZE

https://github.com/theonetruenerd/VenusPackages/blob/main/StrTokenize.pkg

The string tokenize library allows you to split a string into an array based on delimiters. It adds one function:

- *StrTokenize()*

StrTokenize(*variable strIn*, *variable strDelimiter*, *array arrTokens*, *boolean bAttendEmptyTokens*)

This function takes the input string and splits it at any point where it finds the specified delimiter, returning the substrings as members of an array. Can be told to add empty strings or not (occurs when the delimiter appears twice in a row).

Params strIn
The input string to be split

Params strDelimiter
The character to be used as the delimiter

Params arrTokens
The output array of split-substrings

Params bAttendEmptyTokens
Boolean determining whether to include empty substrings or not in the array

Returns
None

Return type
N/A

CHAPTER

TWENTYEIGHT

STRING (FROM HSLEXTENSIONS)

https://github.com/theonetruenerd/VenusPackages/blob/main/String.pkg

The String library from HSLExtensions adds a few functions to facilitate easier manipulation of strings. It adds the following functions:

- *ConvertToAsciiArray()*
- *ConvertToCharArray()*
- *FromAsciiArray()*
- *Join()*
- *JoinWithDelimiter()*
- *Split()*
- *Trim()*

ConvertToAsciiArray(*variable i_strValue*)

This function converts the input string into an array with the regarding ASCII codes. If the input parameter is not a string the function returns an empty array.

Params i_strValue
The input string to be converted into an array

Returns
An array of ASCII codes, or an empty array if the input parameter is not a string

Return type
Array

ConvertToCharArray(*variable i_strValue*)

This function converts the input string into an array with the regarding characters. If the input parameter is not a string the function returns an empty array.

Params i_strValue
The input string to be converted into an array

Returns
An array of characters (strings with length 1), or an empty array if the input parameter is not a string

Return type
Array

FromAsciiArray(*array i_arrAsciiValues*)

This function converts an input array with ASCII codes into a string. If the input parameter is not an array with ASCII codes, the function returns an empty string.

Params i_arrAsciiValues
The input array of ASCII codes to be converted into a string

Returns
The output string formed by the concatenation of the converted versions of the ASCII codes. An empty string if the input parameter is not an array with ASCII codes.

Return type
Variable

`Join`(*array i_arrValues*)

This function joins an array of strings into a single string. Can be used to concatenate any number of strings into a single one. If the input parameter is not an array with strings, the function returns an empty string.

Params i_arrValues
The array of strings to be concatenated

Returns
The concatenated form of all the strings in the array, or an empty string if the input parameter is not an array of strings

Return type
Variable

`JoinWithDelimiter`(*array i_arrValues*, *variable i_strDelimiter*)

This function joins an array of strings into a single string and adds a delimiter between each substring. If the input parameter is not an array with strings, the function returns an empty string.

Params i_arrValues
The input array of strings to be concatenated

Params i_strDelimiter
The delimiter to be inserted between each substring

Returns
The concatenated strings from the array, with delimiters between each substring. An empty string if the input parameter is not an array of strings.

Return type
Variable

`Split`(*variable i_strValue*, *variable i_strDelimiter*, *variable i_bTrimWhitespaces*)

This function splits a string into substrings, forming an array of strings. The input string is split based on a delimiter that the user inputs.

Params i_strValue
The input string to be split into substrings

Params i_strDelimiter
The delimiter to be used to split the string

Params i_bTrimWhitespaces
Boolean determining whether leading and trailing whitespaces will be removed or not

Returns
An array of strings containing each substring formed from splitting the original string

Return type
Array

Trim(*variable i_strValue*)

This function trims leading and trailing whitespace characters from the input string. If the input parameter is not a string the function returns an empty string.

Params i_strValue
The input string to trim

Returns
The trimmed string

Return type
Variable

CHAPTER

TWENTYNINE

TOOLS LIBRARY

https://github.com/theonetruenerd/VenusPackages/blob/main/Tools_Library.pkg

The tools library is designed to add some QoL functions to Venus. It adds the following functions.

- *CalculateNumberOfTubesAndVolumePerTube()*
- *Reformat_Sequence()*
- *Reformat_ValuesForDialog()*

CalculateNumberOfTubesAndVolumePerTube(*variable i_str_ReagentName*, *variable i_int_NumberOfSamples*, *variable i_flt_VolumePerSample*, *variable i_flt_MaximumContainerVolume*, *variable i_flt_DeadLabwareVolume*, *variable i_flt_DeadVolumeReagent*, *variable o_int_NumberOfContainersNeeded*, *array o_arr_VolumesPerContainer*, *array o_arr_DescriptionForDialog*)

This function calculates the number of containers needed for a reagent and the volume per container.

params i_str_ReagentName
The name of the reagent being calculated

params i_int_NumberOfSamples
The number of samples requiring the reagent

params i_flt_VolumePerSample
The volume of reagent required for each sample

params i_flt_MaximumContainerVolume
The maximum volume a single container can hold

params i_flt_DeadVolumeLabware
The dead volume in the labware

params i_flt_DeadVolumeReagent
The percentage dead volume of the reagent

params o_int_NumberOfContainersNeeded
The calculated number of containers needed

params o_arr_VolumesPerContainer
An array of the volumes required for each tube

params o_arr_DescriptionForDialog
An array of the descriptions for the dialog

Returns
None

Return type
N/A

`Reformat_Sequence`(*sequence io_sequenceInput*, *variable i_intNumberOfPositions*)

This function reformats a sequence to have as many positions as needed: Source Sequence: A1, B1

Total Number of positions needed = 4

Output Sequence: A1, B1, A1, B1

Params io_sequenceInput
The sequence of interest

Params i_intNumberOfPositions
The number of positions required

Returns
None

Return type
N/A

`Reformat_ValuesForDialog`(*variable i_flt_valueToReformat*, *variable o_str_valueFormatted*)

This function translates floats into strings and gets rid of any trailing 0s. Use the output value to display volumes in dialogs only.

Params i_flt_valueToReformat
The float to turn into a string

Params o_str_valueFormatted
The output string after it has been converted from a float

Returns
None

Return type
N/A

CHAPTER

THIRTY

TRACELEVEL

https://github.com/theonetruenerd/VenusPackages/blob/main/TraceLevel.pkg

The TraceLevel library adds a variety of functions related to the trace file. The functions it adds are:

- *GetTraceLevel()*
- *SetTraceLevel()*
- *SetStringIndicator()*
- *Trace_02()*
- *Trace_04()*
- *Trace_06()*
- *Trace_08()*
- *Trace_10()*
- *TraceArray()*
- *TraceArrayHorizontally()*
- *TraceArraysFaceToFace()*
- *TraceSequence()*
- *TraceSequenceParameter()*
- *TraceSequencePositions()*
- *TraceAction()*
- *SetActionIndicator()*

GetTraceLevel()

This function returns the current trace level.

Returns

This function returns one of the following predefined constants; TRACE_LEVEL_NONE (0), which corresponds to no traces at all. TRACE_LEVEL_RELEASE (1), which corresponds to only items with release trace level being traced. TRACE_LEVEL_DEBUG (2), which corresponds to everything being traced.

Return type

Variable

SetTraceLevel(*variable i_intTraceLevel*)

This function is used to set the trace level for the method or library.

Params i_intTraceLevel
The trace level for the library. Set to one of the following constants; TRACE_LEVEL_NONE (0), which corresponds to no traces at all. TRACE_LEVEL_RELEASE (1), which corresponds to only items with release trace level being traced. TRACE_LEVEL_DEBUG (2), which corresponds to everything being traced.

Returns
None

Return type
N/A

`SetStringIndicator`(*variable i_strStringIndicatorCharacter*)

This function is used to set one or more characters to indicate strings (by flanking them) in all traces. It can be useful to identify leading or trailing spaces in strings. You can do this by setting i_strStringIndicatorCharacter to ' or *.

Params i_strStringIndicatorCharacter
Characters that are being added at the beginning and the end of each string trace.

Returns
None

Return type
N/A

`Trace_02`(*variable i_intTraceLevel*, *variable i_varToTrace_01*, *variable i_varToTrace_02*)

This function is used to trace the value of 2 variables in one line. Will not automatically insert a space between the two variables.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_varToTrace_01
The first variable to trace.

Params i_varToTrace_02
The second variable to trace.

Returns
None

Return type
N/A

`Trace_04`(*variable i_intTraceLevel*, *variable i_varToTrace_01*, *variable i_varToTrace_02*, *variable i_varToTrace_03*, *variable i_varToTrace_04*)

This function is used to trace the value of 4 variables in one line. Will not automatically insert a space between the variables.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_varToTrace_01
The first variable to trace.

Params i_varToTrace_02
The second variable to trace.

Params i_varToTrace_03
The third variable to trace.

Params i_varToTrace_04
The fourth variable to trace.

Returns
None

Return type
N/A

`Trace_06`(*variable i_intTraceLevel*, *variable i_varToTrace_01*, *variable i_varToTrace_02*, *variable i_varToTrace_03*, *variable i_varToTrace_04*, *variable i_varToTrace_05*, *variable i_varToTrace_06*)

This function is used to trace the value of 6 variables in one line. Will not automatically insert a space between the variables.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_varToTrace_01
The first variable to trace.

Params i_varToTrace_02
The second variable to trace.

Params i_varToTrace_03
The third variable to trace.

Params i_varToTrace_04
The fourth variable to trace.

Params i_varToTrace_05
The fifth variable to trace.

Params i_varToTrace_06
The sixth variable to trace.

Returns
None

Return type
N/A

`Trace_08`(*variable i_intTraceLevel*, *variable i_varToTrace_01*, *variable i_varToTrace_02*, *variable i_varToTrace_03*, *variable i_varToTrace_04*, *variable i_varToTrace_05*, *variable i_varToTrace_06*, *variable i_varToTrace_07*, *variable i_varToTrace_08*)

This function is used to trace the value of 8 variables in one line. Will not automatically insert a space between the variables.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_varToTrace_01
The first variable to trace.

Params i_varToTrace_02
The second variable to trace.

Params i_varToTrace_03
The third variable to trace.

Params i_varToTrace_04
The fourth variable to trace.

Params i_varToTrace_05
The fifth variable to trace.

Params i_varToTrace_06
The sixth variable to trace.

Params i_varToTrace_07
The seventh variable to trace.

Params i_varToTrace_08
The eighth variable to trace.

Returns
None

Return type
N/A

`Trace_10`(*variable i_intTraceLevel*, *variable i_varToTrace_01*, *variable i_varToTrace_02*, *variable i_varToTrace_03*, *variable i_varToTrace_04*, *variable i_varToTrace_05*, *variable i_varToTrace_06*, *variable i_varToTrace_07*, *variable i_varToTrace_08*, *variable i_varToTrace_09*, *variable i_varToTrace_10*)

This function is used to trace the value of 10 variables in one line. Will not automatically insert a space between the variables.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_varToTrace_01
The first variable to trace.

Params i_varToTrace_02
The second variable to trace.

Params i_varToTrace_03
The third variable to trace.

Params i_varToTrace_04
The fourth variable to trace.

Params i_varToTrace_05
The fifth variable to trace.

Params i_varToTrace_06
The sixth variable to trace.

Params i_varToTrace_07
The seventh variable to trace.

Params i_varToTrace_08
The eighth variable to trace.

Params i_varToTrace_09
The ninth variable to trace.

Params i_varToTrace_10
The tenth variable to trace.

Returns
None

Return type
N/A

`TraceArray`(*variable i_intTraceLevel*, *variable i_strDescription*, *array i_arrvarToTrace*)

This function is used to trace an array of variables. It will trace each value of the array in its own line, along with the array description and the index of the value.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_strDescription
A description of the array, which will be at the start of each line of the array trace.

Params i_arrvarToTrace
The array to be traced.

Returns
None

Return type
N/A

`TraceArrayHorizontally`(*variable i_intTraceLevel*, *variable i_strDescription*, *array i_arrvarToTrace*)

This function is used to trace an array of variables. It will trace the array description, followed by each array index and value pair, all on one line.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_strDescription
A description of the array, which will be at the start of the array trace.

Params i_arrvarToTrace
The array to be traced.

Returns
None

Return type
N/A

`TraceArraysFaceToFace`(*variable i_intTraceLevel*, *variable i_strDescription_1*, *variable i_strDescription_2*, *array i_arrvarToTrace_1*, *array i_arrvarToTrace_2*)

This function is used to trace two arrays of variables at the same time, with values at the same index being shown on the same line as one another.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_strDescription_1
A description of the first array, which will be at the start of the array trace.

Params i_strDescription_2
A description of the second array, which will be at the start of the array trace.

Params i_arrvarToTrace_1
The first array to be traced.

Params i_arrvarToTrace_2
The second array to be traced.

Returns
None

Return type
N/A

`TraceSequence`(*variable i_intTraceLevel*, *sequence i_seqToTrace*)

This function is used to trace a sequence. It will list the sequence name, current position, the count and total positions in the sequence, the max number of positions available, and the number of used positions. It will then list the labware ID and position ID for each value of the sequence.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_seqToTrace
The sequence to be traced

Returns
None

Return type
N/A

`TraceSequenceParameter`(*variable i_intTraceLevel*, *sequence i_seqToTrace*)

This function is used to trace the parameters of a sequence. It will list the sequence name, current position, the count and total positions in the sequence, the max number of positions available, and the number of used positions.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_seqToTrace
The sequence to be traced

Returns
None

Return type
N/A

`TraceSequencePositions`(*device ML_STAR*, *variable i_intTraceLevel*, *sequence i_seqToTrace*, *variable i_blnCurrentPositionOnly*)

This function is used to trace the deck positions of a sequence. It will trace the sequence name, then for each part of the sequence it will trace the position ID and the x, y, z, and r coordinates of the position.

Params ML_STAR
The STAR device.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_seqToTrace
The sequence to be traced

Params i_blnCurrentPositionOnly
A boolean determining whether the function will trace positions for all sequence positions (0) or just the current sequence position (1).

Returns
None

Return type
N/A

`TraceAction`(*variable i_intTraceLevel*, *variable i_intAction*, *variable i_strFunctionName*, *variable i_strMethodName*, *variable i_strComment*)

This function is used to trace different action states for a module. *SetActionIndicator()* can be used to help identify actions in log files more easily.

Params i_intTraceLevel
The trace level for the entry. Can be set to either TRACE_LEVEL_RELEASE (1) or TRACE_LEVEL_DEBUG (2). If set to 1, the function will show up in the trace when the trace level is set to either TRACE_LEVEL_RELEASE or TRACE_LEVEL_DEBUG, if set to 2, the function will only show up when the trace level is TRACE_LEVEL_DEBUG.

Params i_intAction
The action to be traced. Set to one of the following constants. START (1) corresponds to the entry being traced as starting. COMPLETE (2) corresponds to the entry being traced as finished successfully. ERROR (3) corresponds to the entry being traced as error occurred. PROGRESS (4) corresponds to the entry being traced as progressing. COMPLETE_WITH_ERROR (5) corresponds to the entry being traced as finished unsuccessfully.

Params i_strFunctionName
The function name for the action trace. Can be set to the return value of the HSL function *GetFunctionName*, which will automatically be formatted correctly.

Params i_strMethodName
The method name for the action trace. Can be set to the return value of the HSL function *GetMethodFileName*, which will automatically be formatted correctly.

Params i_strComment
A comment to be traced with the action trace.

Returns
None

Return type
N/A

`SetActionIndicator`(*variable i_intAction*, *variable i_strIndicator*)

This function is used to set one or more characters to indicate actions. By setting *i_strIndicator* to different characters you can make it easy to identify different actions in the trace.

Params i_intAction
The action to be traced. Set to one of the following constants. START (1) corresponds to the entry being traced as starting. COMPLETE (2) corresponds to the entry being traced as finished successfully. ERROR (3) corresponds to the entry being traced as error occurred. PROGRESS (4) corresponds to the entry being traced as progressing. COMPLETE_WITH_ERROR (5) corresponds to the entry being traced as finished unsuccessfully.

Params i_strIndicator
Character(s) that are being repeated at a total length of up to 100 and traced before and after the action trace.

Type
i_intAction: Variable

Returns
None

Return type
N/A

CHAPTER

THIRTYONE

WINDOWS (FROM HSLEXTENSIONS)

https://github.com/theonetruenerd/VenusPackages/blob/main/Windows.pkg

The windows library from HSLExtensions adds functions related to registries and special directories. The functions it adds are as follows:

- *GetRegistryValue()*
- *GetSpecialDirectory()*
- *SetRegistryValue()*

GetRegistryValue(*variable i_intHKey*, *variable i_strKey*, *variable i_strValueName*, *variable o_varValue*)

This function reads a value from the specified registry

Params i_intHKey
The main key of the registry. 1 = ClassesRoot, 2 = CurrentUser, 3 = LocalMachine, 4 = Users, 5 = CurrentConfig.

Params i_strKey
The key to the path (e.g. SOFTWAREPhoenixDirectoriesMethods)

Params i_strValueName
The name of the value to be read

Params o_varValue
The value which was read (type according to value type)

Returns
Boolean showing whether the key exists (hslTrue) or not (hslFalse)

Return type
Boolean

GetSpecialDirectory(*variable i_intSpecialDirectory*)

This function gets the specified special directory

Params i_intSpecialDirectory
The type of the special directory. 1 = WindowsDirectory, 2 = SystemDirectory, 3 = TemporaryDirectory.

Returns
The special directory

Return type
String

`SetRegistryValue`(*variable i_intHKey*, *variable i_strKey*, *variable i_strValueName*, *variable i_varValue*, *variable i_intValueType*)

This function writes a value to the specified registry

Params i_intHKey
The main key of the registry. 1 = ClassesRoot, 2 = CurrentUser, 3 = LocalMachine, 4 = Users, 5 = CurrentConfig.

Params i_strKey
The key to the path (e.g. SOFTWAREPhoenixDirectoriesMethods)

Params i_strValueName
The name of the value to be read

Params i_varValue
The value which was will be written in

Params i_intValueType
The type of the value. 1 = String, 2 = Number, 3 = Binary, 4 = ExpandableString

Returns
Boolean showing whether the function was successful (hslTrue) or not (hslFalse)

Return type
Boolean

CHAPTER

THIRTYTWO

ZERO UL SCANNER

https://github.com/theonetruenerd/VenusPackages/blob/main/ZerouLScanner.pkg

The zero uL scanner library adds a single function which is designed to be added at the end of a method. It scans the trace file for any steps in which 0uL was pipetted and returns the number of times that happened. This is to check for steps where (for example) it is pipetting a volume based on a variable, and there is a spelling mistake in the variable, or the variable isn't defined correctly. It will work in both simulation mode and normal mode.

- *CheckFor0s()*

CheckFor0s()

This function checks for any steps in which 0uL has been pipetted and returns the number of times that has happened.

Returns
Number of times 0uL has been pipetted

Return type
Variable

INDEX

C

F

G

I

J

L

M

P

Q

R

S

T

U