TabLib RC is a Universal Windows Platform (UWP) Runtime Component designed to make interaction between web applications and the Intoximeters Alco-Sensor VXL PBT easy and straightforward to implement. Intoximeters offers documented html/JavaScript source code examples in the use of this library as a development aid to ISVs. Developers using TabLib RC can design powerful web applications targeting Windows 10 and above that interact with the ASVXL. TabLib RC makes it possible for your web application to implement most any testing protocol through use of the library, eliminating the need for a custom instrument software development contract with Intoximeters.

To use TabLib RC from within an HTML/JavaScript application, the web application must be running in a hosted web browser control (a WebView), that explicitly permits access to the TabLib Runtime Component. The web application itself also must originate from a website URL on an explicit “allow list”. XLweb is the name of a hosting application with a built-in WebView that Intoximeters provides which will allow your web application to access to the TabLib RC from your web app.

Installation of XLweb will install the TabLib Runtime Component and also registers two custom protocol handlers on the target PC - “asvxl:" (non-SSL) and “asvxls:" (SSL). Once installed, any hyperlink beginning “asvxl://” or “asvxls://” clicked within any page displayed in any web browser on the target user’s PC (Chrome/FireFox/Explorer/Edge, etc.) will automatically navigate to the given target URL you designate, but running in the hosted web application environment. Launching XLweb without any target – just <a href=”asvxl:">Go</a>, for example, will open the default demonstration site for TabLib RC, http://apps.intox.com/tablibdemo/index.html. XLweb site can also be launched from the command prompt by simply entering “start asvxl:" (demo site) or “start asvxls://<URL>” (any site).

To have your website temporarily added to the “allow list” in XLweb during development, please contract softwaresupport@intox.com with the URL you wish to authorize for XLweb. A fully licensed copy of XLweb for your web application can be obtained from Intoximeters Sales, on a per-application basis. The licensed XLweb application will default open your website instead of the demo website.
Declaration Example

`var XLcomm;`

Instance Creation Example

`XLcomm = new TablibRC.ASVxlLib();`

Properties, Events, and Methods

**Property** `connectionType`

Connection type. 0=Serial; 1=Bluetooth Low Energy (BLE). Connection type to establish with AS-Vxl.

**enumPorts()** `Async Method`

`enumPorts()` returns a list of the names of the serial port names available on the PC, in setup for a user selection input box. For use with Serial connection type.

**Property** `port` `As String`

The USB virtual communication port (VCP), i.e. “COM1”, to which the docking cradle is connected. Set this to a user-designated entry returned from `enumPorts()`. For use with Serial connection type.

**Property** `pairString` `As String`

Bluetooth connection type only - A unique string identifying the workstation and application attempting to communicate with this ASVxl (AKA, the “PC Link Association”). This string should be unique to the specific instance of the application that will be accessing the ASVxl; up to 8 alphanumeric characters in length. When calling `connectNew()`, `pairString` will be programmed into the ASVxl during the course of the call. When calling `connectTo(DeviceID)`, `pairString` must match the “PC Link BT Pair” setting already configured on the ASVxl.

**enumPaired()** `Async Method`

`enumPaired()` returns a list of the serial numbers of all BLE ASVxls that are paired with the PC.

**Property** `intoxNetSerial` `As Integer`

Unique ID number for the customer group authorized to access the ASVxl. Under best practices, your application should be uniquely serialized on a per-customer basis. The customer would then set this ID - the `intoxNetSerial` setting - on all of their ASVxls to match the serialization you assign, granting software access to the group of ASVxls belonging to this customer at any PC where the customer has your application installed and also preventing one customer from gaining electronic access to another customer’s instrument data through serial or BLE interface with the instrument. Relying upon or expecting the customer to change the `intoxNetPassword` setting alone is not a best practice. This number must match either its “IntoxNet Serial #1” setting or “IntoxNet Serial #2” setting on the ASVxl. Default value is 00000.
Property IntoxNetPassword As String

Password for electronic ASVXL access. A string of up to 15 characters in length; must also be set identically on the ASVXL as the “IntoxNet Password” setting. Default value is “CHANGEME”. Best practice is that you require the customer to decide and set this.

connectNew() Method

Launches a process that scans for and opens a communications channel with an ASVXL over Serial or Bluetooth (depending upon connectionType setting). If connectionType is Bluetooth, ASVXL must be attempting PC Link, and be looking to establish a new PC Link application association. If the XL has previously been associated with an application, the user must first clear that association by selecting “INIT” from “Test Menu” / “PC Link” on the ASVxl, prior to initiating the link attempt. Communications will not take place using connectNew if the XL already has a PC association (pairString setting is not “INIT”).

connectTo(DeviceID As Integer) Method

Opens a connection to a specific ASVXL where DeviceID is the ASVxl’s serial number.

If the connection type is serial, it is not necessary for the user to initiate the connection from the ASVxl, the instrument need only be powered off and in its cradle.

If the connection type is Bluetooth, the user should be told to initiate communications at the time that connectTo is called. The user may bypass the menu and quickly initiate communications by depressing the trigger on the ASVxl while holding down the left front interface button.

onprogressevent(evt) Event

Provides information about the progress of communications with the device. The evt argument can be JSON parsed to a StatusEventArgs class (see ondisconnectevent below). During this event, you may display the StatusEventArgs.Status messages to the end user in a UI or for debugging. You can also use the StatusEventArgs.StatusCode value to drive a visual progress bar.

StatusCodes are sequenced in the natural order which they would likely occur over the course of a typical communications session, and so may be used to drive a graphical progress bar to indicate communications progress, with StatusCodes.Complete being the maximum value.

ondisconnectevent(evt) Event

Raised by the API when a connection with the ASVXL is completed, closed or broken. The evt argument can be JSON parsed to a StatusEventArgs class.

Public Class StatusEventArgs
    Property Status As String ‘Descriptive message
    Property StatusCode Integer ‘A JSON serialized StatusCodes ennumerable (see appendix 1)
    Property StatusType As Integer
        ‘0 = Status message, 1 = Status message Update, 2 = Error message, 3= Disconnect message
End Class
The `StatusCodes` enumeration is listed in appendix 1, for reference; `StatusCodes.Complete` will always be the highest value. Because the underlying enumerable data type cannot be made directly accessible in the JavaScript namespace, an additional function is available in TabLib RC, `statusCodesDictionary`.

```csharp
Function statusCodeDictionary(Forward As Boolean) As IReadOnlyDictionary(Of String, String)
```

This function returns a read-only dictionary (Map View) that can be used to translate each `StatusCode` to the underlying enumerable name, providing like functionality to an enumerable within JavaScript. Set `Forward = false` to return the reverse dictionary, which will translate the enumerable name to its corresponding status code. Hard-coding checks for specific `StatusCode` integer values into your application is not recommended - these values may change from one release of TabLib RC to the next. Instead, use the `statusCodeDictionary(..)` function and reference the status code by its underlying enumerable name, as in the following example code:

```javascript
Var XLcomm = new TablibRC.ASVx1Lib(); // The TabLib component instance
var statuscodes; // A cached copy of the TabLib status code dictionary

statuscodes = XLcomm.statusCodeDictionary(0); //0 = Reverse lookup dictionary; 1 = forward
XLcomm.ondisconnectevent = XLcomm_Disconnect; //Add a handler the TabLib disconnect event

function XLcomm_Disconnect(evt) {
obj = JSON.parse(evt);
if (obj.StatusCode == statuscodes.lookup("Complete")) {
// No error. No need to display the progress details.
// ...
} else {
// An error occurred. Show the progress details textarea, if hidden, to inform the user
// ...
}
}
```

When the NotifyType is 1 (a “Status update message”), it is suggested that instead of appending the new status message to any displayed progress log, that the last line of display be replaced with this new message.

A special `StatusCode` is `StatusCodes.InTestStatusCode`. In this case, the status message is originating in real-time from the ASVxl itself. The `Status` string will not be the normally descriptive text that can be displayed directly to the end user, but instead will be an ASVxl status code of the form, “S01”, “S02”, etc. The `getMap(2)` method (`MapTypes.Status =2`) is used to download a dictionary translating these in-test status codes to descriptive status messages from the instrument. This dictionary need only be downloaded once per instrument type. An instrument type is determined by the unique pairing of device software and firmware version (see `ConnectedDeviceSettings`).

```csharp
Method

cancel()

Terminate any connectNew() or connectTo(DeviceID) in progress, and releases associated resources.

Event

onconnectevent(evt)

Raised upon successful login to the ASVxl. The evt argument can be JSON parsed to the integer serial
number of the connected ASVxl. “Connected only” methods and properties (marked with a “★” below) are only available and valid during/after this event, while the connected() property is true, and before the ondisconnectevent(...).

Property connected As Boolean

Read only. true while connected to and logged into an ASVxl. Otherwise, false.

Property connectedDeviceSettings As string ★

Read only. Provides information about the ASVxl with which you are actively communicating. connectedDeviceSettings can be JSON parsed to connectedDeviceSettingsType.

Structure connectedDeviceSettingsType
    Dim SerialNumber As Integer
    Dim Customer As String
    Dim FirmwareVersion As String
    Dim SoftwareVersion As String
    Dim SoftwareCRC As UInteger
    Dim HasBarometer As Boolean
    Dim LanguageIndex As Integer  '0 = US English, otherwise, see GetLanguages()
    Dim CalibrationCRC As UInteger
    Dim LastTestNumber As Integer
    Dim DateLastCalibrated As Date
    Dim SamplesSinceLastCalibration As Integer
    Dim DateLastAccuracyCheck As Date
    Dim SamplesSinceLastAccuracyCheck As Integer
End Structure

logoff(IndicateSuccess As Boolean) ★

Ends the communications session with the ASVxl. You may set IndicateSuccess to true to indicate to the user that the communication has completed successfully. The ASVxl itself indicates successful completion of communications by powering down. Setting IndicateSuccess to false will cause the ASVxl to return to the call-out menu and display, “Call failed, Retry?” when using Blue-Tooth PC Link.

NOTE: Setting IndicateSuccess to true implicitly calls the acknowledgeDownload() method.

Async Method ★

setQAP (QAP As string) As IAsyncOperation(Of Integer)

Installs a quality assurance plan to the ASVxl. If successful, this function will return a checksum of the installed QAP. A return value of zero indicates failure. On subsequent calls, you can set XLcomm.qapcrc (see Property qapcrc, below) to this returned value to ensure the designated quality assurance plan is in place.

Your software application can implement a QAP independent of the device, based upon connectedDeviceSettings and retrieved accuracy check and calibration test results. The setQAP method permits you to have the ASVxl itself independently maintain a QAP. See appendix 2 for the TabLib ASVxl’s default (factory-shipped) QAP, as well as a model US DOT (department of Transportation) testing protocol QAP.

QAP is a json serialized QAPtype
Structure QAPtype

Dim StandardWarningDays As Short
Dim AccuracyCheckRangeType As Integer  '0="Abs", 1="Pct", 2="Both", 3="Either", 4="None"
Dim AccuracyCheckPercentErrorPlus As Byte   '10 = 10%
Dim AccuracyCheckPercentErrorMinus As Byte
Dim AccuracyCheckAbsoluteErrorPlus As UShort  '50 = 0.05 BrAC
Dim AccuracyCheckAbsoluteErrorMinus As UShort
Dim CalibrateWarningDays As Short
Dim CalibrateWarningTests As UShort
Dim CalibrateDisableDays As Short
Dim CalibrateDisableTests As UShort
Dim CalibrateDisableTime As String
Dim AccuracyCheckWarningDays As Short
Dim AccuracyCheckWarningTests As UShort
Dim AccuracyCheckDisableDays As Short
Dim AccuracyCheckDisableTests As UShort
Dim AccuracyCheckDisableTime As String
End Structure

Property qapcrc As Integer

Used to verify or inspect the QAP (quality assurance plan) in place on the ASVxl.

a) If the qapcrc property is 0 (default value), verification of the in-place QAP on the device is skipped.

b) If the qapcrc property is set to -1 before calling the device, then the quality assurance plan in place on the device is “inspected” during the call, before the onconnectevent() is raised. The qapcrc property is updated to reflect the CRC of quality assurance plan in place on the device being contacted. This gives you an opportunity to validate the qapcrc and if incorrect, to call the setqap(...) method if the desired QAP is not in place, all within a single call.

c) If qapcrc property is set to any value other than 0 or -1, then it will be verified before the onconnectevent() that the quality assurance plan in place on the device has a checksum matching this value. If the QAP in place on the instrument does not match, the call will fail with an appropriate error code and message, and the onconnectevent will never take place.

Async Method *

setClock(NewDate As DateTimeOffset) As IAsyncOperation(Of Boolean)

Sets the clock on the ASVxl to the date and time you provide. Returns true if successful.

getMap(MapType As integer) As IAsyncOperation(Of String)   Async Method *

Map types (integer): MapTypes.Warning = 0, MapTypes.Disables = 1, MapTypes.Status = 2, MapTypes.Errata = 3

Interrogates the ASVxl and returns a string that can be JSON parsed to a dictionary. The dictionary will map a code like "S03" to its respective description “Install a New Mouthpiece”. Instrument warning, disable, in-test instrument status codes, and test result error code dictionaries are available. ASVxl in-test status codes are not to be confused with TabLib status codes, as the former is obtained from the testing device. Set MapType to the dictionary you wish to retrieve from the ASVxl. Communication session
durations may be shortened (especially when communicating over BLE) by querying these dictionaries only once per instrument type and storing this dictionary to a database. An instrument type is determined by the unique pairing of device software and firmware version (see connectedDeviceSettings).

getTestingStatus()  As  IAsyncOperation(Of String)  Async Method  *

Returns (i) A list of authorizable atomic test types, their associated descriptions and whether the test type is currently enabled, (ii) A list of currently set instrument warning codes and iii) A list of currently set instrument disable codes. A non-empty string return value indicates success. The returned string can be JSON parsed into a TestingStatusType structure.

Structure  TestingStatusType
  Dim  TestsEnabled  As  List(Of  TestEnabledType)
  Dim  WarningCodes  As  List(Of  String)  ‘See  getMap(0)
  Dim  DisableCodes  As  List(Of  String)  ‘See  getMap(1)
End  Structure

Structure  TestEnabledType
  Dim  TestType  As  String  “Subject” / “Passive” / “AccuracyCheck” / “Calibration”
  Dim  TestTypeDescription  As  String
  Dim  IsEnabled  As  Boolean
End  Structure

getInputs()  As  IAsyncOperation(Of String)  Async Method  *

Returns a list of upload fields (“inputs”) required for each test type supported on this instrument version. An empty returned string indicates failure. The returned string can be JSON parsed into a UploadProtocol structure.

Structure  UploadProtocol
  Dim  TestType  As  String
  Dim  Description  As  String
  Dim  TestAuthInputs  As  List(Of  AuthField)
End  Structure

Structure  AuthField
  Dim  Name  As  String
  Dim  Description  As  String
  Dim  Type  As  Integer  ‘Input type:  0=String, 1=Selection, 3=Masked.
  Dim  MaxLength  As  Integer
  Dim  Selections  As  List(Of  String)  ‘Input type = 1 only
  Dim  InputMask  As  String  ‘Input type = 3 only
  Dim  DefaultValue  As  String
End  Structure

Async Method  *

downloadTests(NoMetaData  As  Boolean,  StartingNo  As  Integer)  As  IAsyncOperation(Of String)

Returns test results stored in the ASVxl, from oldest to newest. An empty returned string indicates failure. The returned string can be JSON parsed into a TestResultInfo structure.

The test download consists of test result metadata and test result data. Communication session durations may be shortened (especially when communicating over BLE) by querying the MetaData only once per instrument type, storing this information to a database and turning off the MetaData transmission in
subsequent downloads (setting NoMetaData to true). An instrument type is determined by the unique pairing of device software and firmware version (see ConnectedDeviceSettings).

The download will start with the test number given by StartingNo. Setting StartingNo=0 downloads all stored tests. Your application should remember the highest test number previously downloaded, and pass this value, plus one, when calling downloadTests to ensure optimally fast downloads.

Structure TestResultInfo
Dim MetaData As DownloadMetaDataType
   Dim TestResults() As DownloadTestType
End Structure

MetaData:

Structure DownloadMetaDataType
   Dim CRC As UInteger ‘A checksum summarizing all of the MetaData
   Dim MapFileID As String ‘An alternate checksum summarizing the MetaData
   Dim Protocols() As DownloadProtocolType
   Dim Structures() As DownloadStructureInfoType
End Structure

Structure DownloadProtocolType
   Dim Description As String
   Dim Number As Integer
   Dim Version As String
   Dim CRC As UInteger ‘An instrument checksum of the testing protocol
End Structure

Structure DownloadStructureInfoType
   Dim Description As String ‘Description, i.e. “Test Common Info”
   Dim StructureType As StructureTypes
   Dim CRC As UInteger
   Dim TimeFormat As TimeFormats
   Dim Fields() As FieldInfo
End Structure

Enum StructureTypes
   Custom = 0 ‘All structure types other than those below
   TestCommonInfo = 1 ‘This structure is common to all test types
   TestResults = 2 ‘This structure is common to all test results
   EndOfTestStatus = 3 ‘This structure is also common to all test types
   TestWFfileData = 4 ‘Does not apply to the ASVxl
End Enum

Enum TimeFormats ‘All time formats within a given structure are the same.
   ShortTime ‘HH:NN = 0
   LongTime ‘HH:NN:SS = 1
End Enum

Structure FieldInfo
   Dim FieldName As String
   Dim FieldDescription As String
   Dim FieldType As DownloadFieldTypes
   Dim MaxLenOrMemoSpec As Integer
End Structure

Enum DownloadFieldTypes
   StringType ‘Fixed maximum length string
   DateType ‘MM/DD/YYYY
   TimeType ‘HHNN or HHNNSS, depending upon Structure TimeFormat setting
   MemoType ‘Variable length string
End Enum

Data:
Structure DownloadTestType
    Dim ProtNum As Integer
    Dim TestNum As Integer
    Dim TestSteps() As TestStepType
End Structure

Structure TestStepType
    Dim StructNum As Integer 'Structure number within its download
    Dim SeqNum As Integer
    Dim OtherFields() As String
End Structure

Async Method ★

acknowledgeDownload() As IAsyncOperation(Of Boolean)

Acknowledges that you have downloaded and stored the test results from the ASVxl. The TabLib ASVxl is configured to disallow the authorization of tests as long as any result is on file that has not been downloaded. As such, this function must be called after successful download to re-enable the authorization of new testing on the instrument. Calling the logoff(true) method implicitly invokes this function.

Property printWidth As String

The beginTest(...) and print(...) methods (see below) can obtain and return a test result printout directly from the Alco-Sensor Vxl. The printout is intended for display or print using a monospaced font. To support multiple printers, one may specify the width, in characters, of the output target area, by setting this property before calling the beginTest or print methods. The default printWidth value is 32 (characters per line). A minimum value of 24 characters per line is required.

Async Method ★

beginTest(AuthTestTypeName as String, Inputs As String) As IAsyncOperation(Of String)

AuthTestTypeName: Test type to authorize: “Subject” / “Passive” / “AccuracyCheck” / “Calibration”.

Inputs: An array of key value pairs comprising the input data fields required to authorize this atomic test type (see GetInputs command), JSON serialized to string.

For all test types, Inputs will include the following four fields:
CONTROLID, CSTITLE, CONFSTRING, DISPTTYPE

CONTROLID is a string up to 64 ANSI characters long, that is intended for use in storing data that uniquely identifies the test being authorized. For example, a hash (e.g, SHA-256 message digest) of all COC data collected on the tablet (i.e. Subject name, reason for test, technician name) plus the instrument serial number and anticipated test number setting retrieved immediately before the beginTest method is called (connectedDeviceSettingsType..LastTestNumber + 1). The ASVxl will store the CONTROLID with the test result and return it with the downloaded or printed test result.

CSTITLE, CONFSTRING: Before conducting the breath test, the ASVxl displays a confirmation screen to ensure the strongest possible chain of custody. CSTITLE (the confirmation string title) is displayed directly above the confirmation string, as shown in the left image below:
CONFSTRINGTITLE, if not set, defaults to “Subject Name” if the type of test being conducted is “subject”, and defaults to “Technician Name” for accuracy checks and calibrations. In the example above, CONFSTRING has been set to “Rankine Forrester”.

DISPTTYPE (displayed test type): Before conducting the breath test, the ASVXL will confirm the test type and test number. DISPTTYPE is displayed above the test number, as illustrated above right. The default value is “Subject Test” for test type “subject”, or “Accuracy Check” and “Calibration” for “accuracycheck” and “calibration” test types, respectively. If set to an empty string, this ASVXL display screen is skipped.

When calling the `beginTest(…)` method, the returned string can be JSON parsed into a `BeginTestResult` structure, with ReturnCode indicating whether the command succeeded.

If the ASVXL remains connected for the entire course of the test, the `BeginTestResult` returned have a returncode of 3, and will include the actual test result printout. See `PrintTestResult` structure below.

```
Structure BeginTestResult
    Dim ReturnCode As Integer 'A BeginTestReturnCodes as listed below
    Dim TestPrintoutInfo As PrintTestResult
End Structure

Enum BeginTestReturnCodes
    FailedToStart = 0
    StartedWithDisconnect = 1
    CompletedPrintoutDownloadError = 2
    CompletedWithResult = 3
End Enum

Structure PrintTestResult
    Dim ControlID As String
    Dim TestResult As String 'i.e. “.082”
    Dim TestStatus As String
    'Test status code, i.e. “S03” – see GetMap(2) noting that MapTypes.Status = 2
    Dim TestPrintout As String
End Structure
```

**Async Method** *

```
print(TestNumber As Integer) As IAsyncOperation(Of String)
```

Interrogates the ASVXL and returns a string that can be JSON parsed to a `PrintTestResult` of test #<TestNumber> (or of the last test conducted on the ASVXL when TestNumber = 0). An empty string return value indicates failure.

Example `PrintTestResult.TestPrintout:`
Serial Number:              5019
Test Number:                 110
Test Date:                   04/07/2017
Test Time:                   14:08
Test Temperature:           24.0°C

Standard Type:               Dry
Standard Value:              .082
Target Value:                .081
Standard Lot Number:        AG321303
Standard Tank Number:       1
Standard Expiration Date:   04/07/2019
Barometric Pressure:        751.1 mmHg

<table>
<thead>
<tr>
<th>Type</th>
<th>Time</th>
<th>g/210L</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLNK</td>
<td>.000</td>
<td>14:09</td>
</tr>
<tr>
<td>ACC</td>
<td>.081</td>
<td>14:09</td>
</tr>
</tbody>
</table>

Test Status:             Success

---

Async Method

serialPrint(PrinterPort As String, BaudRate As Integer, HandShaking As Integer, SetupBytes As Byte(), PrintOutput As String) As IAsyncOperation(Of Boolean)

If you are using a serial printer that utilizes a USB virtual COM port interface to print text-only test results from your application, it is possible to bypass the Windows printing subsystem and send print data directly to the print device using TabLib. This can be used to skip the standard printer selection dialog associated with the use of the Windows printing subsystem. The serialPrint method can send text print output directly to the designated serial port. It is not intended for graphics mode printing, and as such must strictly use fonts resident on the printing device. A series of setup bytes can be designated to be sent once to the device preceding the print job. These bytes may be used for printer feature selection such as base font or print intensity setup. Be sure to consult the programming guide for the serial printer you will be using; there may likely be setup codes specific to your print device you will need to send to obtain the desired output format, or setup procedures for the print device itself to designate your desired interface settings.

Parameters:

**PrinterPort:** The USB virtual communication port (VCP), i.e. “COM1”, to which the printer is connected. Set this to a user-designated entry returned from enumPorts().

**BaudRate:** The serial interface speed of the printer, which is likely configurable on the device.

**HandShaking:** 0 = no handshaking. 1=RTS flow control. Higher print speeds are possible with flow control enabled, however flow control may need to be enabled on the print device itself.

**SetupBytes:** An array of bytes to be send to the printer immediately before the print data, for any desired printer setup.

**PrintOutput:** The print job data to send to the printer, for example, a string like PrintTestResult.TestPrintout.
A return value of `True` indicates success in sending the designated data to the selected serial port.
TabLib RC usage example

1) Declare it

```javascript
var XLcomm = new TablibRC.ASVxLib();
```

2) Handle the onprogress event. In this event, we’ll show progress messages to the end user in an html textarea element with id “txtaProgress”

```javascript
XLcomm.onprogressevent = XLcomm_Progress;

function XLcomm_Progress(event) {
    obj = JSON.parse(event);
    txtaProgress.innerHTML += obj.Status;
}
```

3) Handle the onconnectevent. Later, we’ll define this function to do some interesting things.

```javascript
XLcomm.onconnectevent = XLcomm_Connect;
```

4) Set instrument access security settings. These must match configured setting on menu of ASVxl

```javascript
XLcomm.intoxNetSerial = 10000;
XLcomm.intoxNetPassword = "Cust_A_Password";
```

5) Set Properties necessary to support the connection type

```javascript
XLcomm.connectionType = 0;  // ConnectionTypes.Serial = 0
XLcomm.port = "COM4";
```

or

```javascript
XLcomm.connectionType = 1;  // ConnectionTypes.BluetoothLE = 1
XLcomm.pairString = "OICU812";
```

6) Connect to a specific ASVxl (SN# 20001), using the connectTo method.

```javascript
XLcomm.connectTo(20001);
```

7) In the Connect event, we’ll set the instrument clock, initiate a subject test, and display the test result printout in a textarea with id “txtaTestPrintout”, using a monospaced font.

```javascript
function XLcomm_Connect(event) {
    var now = new Date();
    var ClockSetSuccess = await XLcomm.setClock(now);
    if (!ClockSetSuccess) { XLcomm.logoff(false); return; }
    var dict = [];
    dict.push({ key: "CONFSTRING", value: "Alexander Hamilton" });
    dict.push({ key: "DISPTYPE", value: "Screening Test" });
    dict.push({ key: "CONTROLID", value: "689c8efa-008a-4805-a7d4-c54c9acf14da" });
```
```javascript
var MyInputs = JSON.stringify(dict);
BeginTestResult = await XLcomm.beginTest("Subject", MyInputs)

if (BeginTestResult == '') { XLcomm.logoff(false); return; } else {
  ParsedResult = JSON.parse(BeginTestResult);
  if (ParsedResult.ReturnCode != 3) { XLcomm.logoff(false); return; } else {
    var tpi = ParsedResult.TestPrintoutInfo;
    if (tpi !== null) txtaTestPrintout.innerHTML = tpi.TestPrintout;
  }
}
XLcomm.logoff(true);
```
Appendix 1. TabLib Status Codes

```
Enum StatusCodes
  InvalidSerialPort
  DevicesFound
  ConfiguringForNotification
  InitiatePClink
  BeginningScan
  PairCodeEntry
  PairingComplete
  ScanningHalted
  PairingStarted
  PairingFailed
  PairingResult
  SendingENQ
  ServiceAccessDenied
  ServiceAccessFailed
  EnquiryReply
  InitialPromptTimeout
  LoginChallengeTimeout
  LoggedOnStatusCheckTimeout
  CommandResponseTimeout
  DataEventTimeout
  DeterminingInstrumentStatus
  LoginProgress
  ResumingScanFromError
  HaltingScan
  SessionLogoutPromptTimeout
  NoOrBadCmdReplyCRC
  LogonSuccess
  LogonFailure

  SettingPairString
  SetPropertyStringSuccess
  SetPropertyStringFailure
  InvalidSettingsResponse
  WasNotTheTargetedDeviceSerialNumber
  QueryingDeviceStats
  GotDeviceStats
  CheckingHardwareConfig
  BarometerPresenceDetectionTimeout
  BarometerPresenceDetectionError
  HardwareCheckComplete

  VerifyingQAP
  InspectingQAP
  QAPvalidated
  QAPinspected
  QAPvalidationFailed
  InvalidQAPsettingsResponse
  ActivatingQAP
  QAPupdateError
  QAPupdateSuccess

  Connected
  SettingASVxlClock
  ClockSetSuccess

  QueryingLanguages
  QueryingLanguagesFoundOne
  QueryingLanguagesError

  QueryingWarningsMap
  QueryingWarningsMapSuccess
  QueryingDisablesMap
  QueryingDisablesMapSuccess
  QueryingStatusCodesMap
```
QueryingStatusCodesMapSuccess
QueryingErrorCodesMap
QueryingErrorCodesMapSuccess
QueryingProtocolInputs
QueryingProtocolInputsSuccess
QueryingTestEnables
QueryingTestEnablesSuccess
QueryingWarnings
QueryingWarningsSuccess
QueryingDisables
QueryingDisablesSuccess

DownloadingTests
BadDownloadVersion
UnsupportedDownloadVersion
BadProtocolCount
ReceivingProtocols
InvalidTestProtocolDescriptionHeaderReceived
InvalidProtocolNameHeaderReceived
InvalidProtocolVersionHeaderReceived
BadStructureCount
ReceivingStructures
InsufficientStructureLines
InvalidStructureDescriptionReceived
InvalidStructureNumberReceived
InvalidStructureFieldCountReceived
InvalidStructureNameReceived
InvalidStructureTimeFormatReceived
InvalidStructureRecordReceived
BadMetaDataCRC
BadTestCount

NoNewTestsOnFile
ReceivingTests
InvalidTestStepReceived
ReceivedTestCountDidntMatchAnnounced
InvalidTamperStampReceived
DownloadingTestsSuccess

AcknowledgingDownload
DownloadAcknowledgeFailure
DownloadAcknowledged

MissingControlData
UploadingControlData
UploadControlDataSuccess
NoOrInvalidPrepareResponse
PrepareResponseNotSuccess
StartingTest
StartTestCommandFailed
TestStarted

InTestStatusCode
RealtimeLinkBrokenDownloadWhenComplete
TestComplete
PrePrintPromptTimeout
RequestingPrintout
PrintoutReceived
UnparseableTestResultMetaData
TimeOutReceivingTestPrintout
TimeOutReceivingTestPrintoutChecksum
BadTestPrintoutChecksum
PrintoutNotOnFile
TestPrintoutReceived

OpeningSerialPrinterPort
OpenSerialPrinterPortFailure
SendingPrintData
WriteToSerialPrinterFailure
WriteToSerialPrinterComplete

InvalidClearTestsParameters
ClearingTests
ClearTestsTimeout
ClearTestsFailed
ClearTestsRangeNotFound
ClearTestsSuccess

LogOffSuccess
Complete

End  Enum
Appendix 2. TabLib ASVxl Quality Assurance plans

Default plan

AccuracyCheckAbsoluteErrorMinus: 50
AccuracyCheckAbsoluteErrorPlus: 50
AccuracyCheckDisableDays: 0
AccuracyCheckDisableTests: 0
AccuracyCheckDisableTime: "00:00:00"
AccuracyCheckPercentErrorMinus: 5
AccuracyCheckPercentErrorPlus: 5
AccuracyCheckRangeType: Either {3}
AccuracyCheckWarningDays: 0
AccuracyCheckWarningTests: 0
CalibrateDisableDays: 0
CalibrateDisableTests: 0
CalibrateDisableTime: "00:00:00"
CalibrateWarningDays: 0
CalibrateWarningTests: 0
StandardWarningDays: 10

qapcrc = -1980173064

US DOT model plan

AccuracyCheckAbsoluteErrorMinus: 50
AccuracyCheckAbsoluteErrorPlus: 50
AccuracyCheckDisableDays: 30
AccuracyCheckDisableTests: 0
AccuracyCheckDisableTime: "00:00:00"
AccuracyCheckPercentErrorMinus: 10
AccuracyCheckPercentErrorPlus: 10
AccuracyCheckRangeType: Absolute {0}
AccuracyCheckWarningDays: 25
AccuracyCheckWarningTests: 0
CalibrateDisableDays: 0
CalibrateDisableTests: 0
CalibrateDisableTime: "00:00:00"
CalibrateWarningDays: 0
CalibrateWarningTests: 0
StandardWarningDays: 10

qapcrc = -1478141137