This document details the development and runtime requirements, as well as the functions supported in the library.

Runtime Requirements:

The following files must be copied into the working directory:

COLOR200.DLL	XR8000.007	XR8000.804
X2D200.DLL	XR8000.00A	CORRELATION
XR8000.DLL	XR8000.00C	
XR8000.009	XR8000.010	

The following files must be copied into the Windows system directory:

USBX.OCX (this file must also be registered, since it is an ActiveX Control)

The following files are required for successful USB installation of the camera and XR8000 instrument:

OV511P.INF	XRUSB.INF
OV511P.CAB	XRUSB.CAB

These files will be required when the user first powers up the instrument but has not yet installed the drivers for it on that machine.

Supported Platforms:

The XR8000 library has been tested on Windows 98, Windows 2000, and Windows XP. We recommend you only install and use the library on these platforms.

Function Library:

This is the complete library of functions that can be called from the XR8000.DLL. These functions will not operate unless the instrument drivers have been properly installed.

Also note that functions are provided for both Visual Basic and C/C++. Visual Basic (VB) equivalents have been provided to account for various language and interface differences.

In all cases where C/C++ functions return Boolean values, you will notice that their VB counterparts return Integer values instead. In C/C++, you can use TRUE and FALSE to check the success of a function. In Visual Basic, use 1 and 0 instead.

C/C++ Example:

```
If ( Instrument_Standalone() == TRUE )
{
   ...
}
```

VB Example:

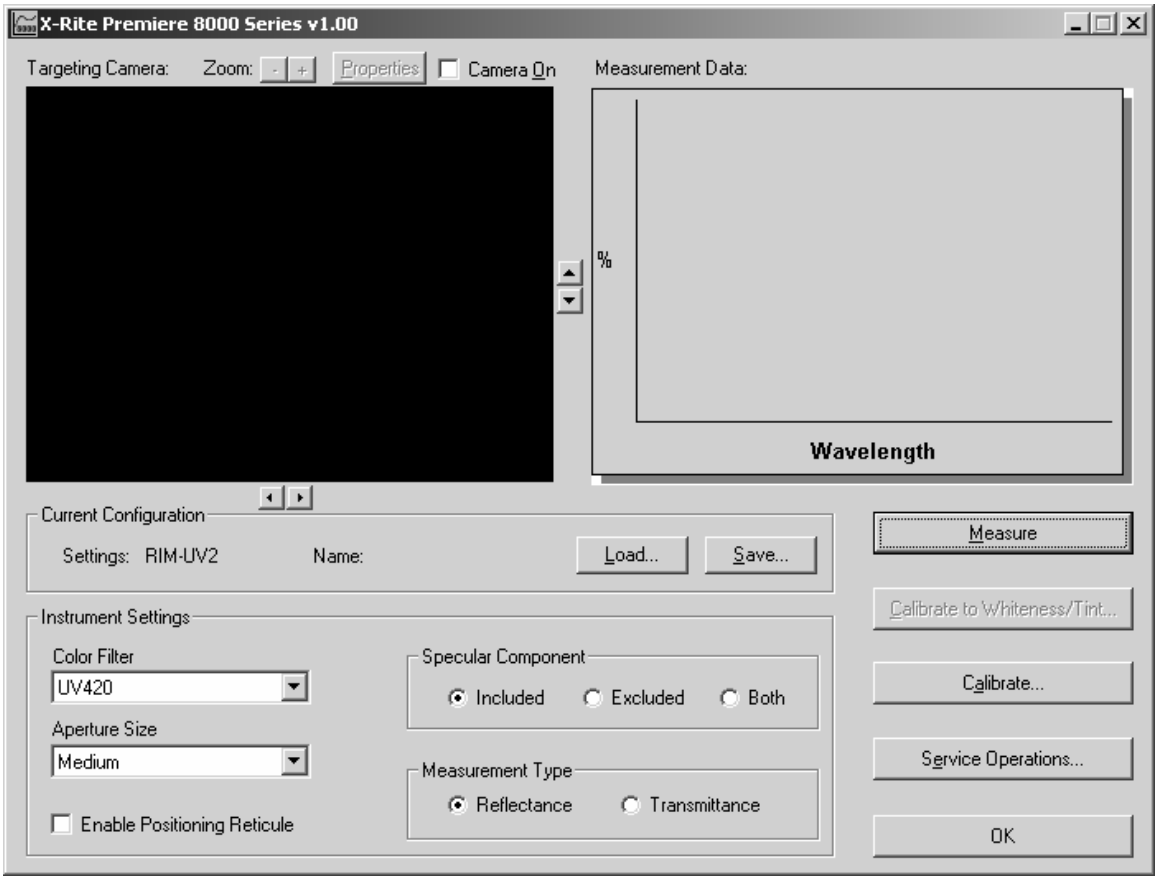
```
If VB_Instrument_Standalone() = 1 Then
   ...
End If
```

1. Instrument_Standalone

This function brings up a graphical user interface that enables the user to exercise all the commands below, as well as access to various service operations and calibration sequences.

C Prototype: `BOOL Instrument_Standalone()`
VB Prototype: Declare Function VB_Instrument_Standalone Lib "xr8000.dll" () As Integer

This function returns FALSE if the dialog successfully appears. Otherwise, the function returns TRUE when the dialog is dismissed. The dialog that appears is shown below:



Use this function only if you wish to use the interface as shown in the figure. If you wish to use your own interface, you should start with the Instrument_Initialize function shown below.

2. Instrument_Initialize

This function establishes communication with the instrument so that you can exercise all the other commands below. It is the first function you should call before using any other.

C Prototype: `BOOL Instrument_Initialize()`
VB Prototype: Declare Function VB_Instrument_Initialize Lib "xr8000.dll" () As Integer

This function tries to detect the instrument on the USB port for five seconds. If detected, the USB port is opened and the function waits 10 seconds for a ready signal from the instrument. If the instrument is ready, the function

returns TRUE. If the instrument could not be detected on the port, or if the instrument port cannot be opened, the function returns FALSE.

As part of the initialization process, this function checks a local INI file to get the latest configuration the instrument was set to. If no prior configuration has been set, the function resorts to an small aperture, no filter, and reflectance measurement mode. You can change these parameters by calling Instrument_Configure.

3. Instrument_Configure

This function brings up the XR8000 dialog, which can be used to configure the instrument. This dialog is shown in the information on the Instrument_Standalone section

C Prototype: `void Instrument_Configure()`
VB Prototype: Declare Sub VB_Instrument_Configure Lib "xr8000.dll" ()

The dialog this function brings up enables the user to change all the parameters that can be seen on that dialog. In addition, the user can perform calibrations, take measurements, or perform service operations from this general "command center."

4. Instrument_SetConfiguration

This function allows the developer to set configuration settings without the user interface in the Instrument_Configure command described above.

C Prototype:
```
void Instrument_SetConfiguration( int nAperture,
      int nFilter, float fPctEnergy, BOOL bReflectance,
      int nAngle )
```

VB Prototype: Declare Sub VB_Instrument_SetConfiguration Lib "xr8000.dll" (ByVal nAperture As Long, ByVal nFilter As Long, ByVal fPctEnergy As Single, ByVal bReflectance As Boolean, ByVal nAngle As Long)

The aperture size is determined by nAperture and is dictated as follows:

0 – Small 1 – Medium 2 – Large

The filter type is determined by nFilter and is dictated as follows:

0 – Open (no filter)
1 – SP64 Whiteness
2 – SP64 DayGlo
3 – UV 400 Filter
4 – UV 420 Filter
5 – UV 420 Filter partially inserted

If nFilter = 5, you must specify the amount of UV energy allowed through the filter in the fPctEnergy value. This can range from 0% (filter fully closed) to 100% (filter wide open).

If the user is measuring reflectance data, set bReflectance to TRUE. If transmittance, set bReflectance to FALSE.

The angle is set by change nAngle to one of the following values:

0 – Included 1 – Excluded 2 – Both

These configuration settings will take effect when you call Instrument_Measure or Instrument_Calibrate.

5. Instrument_GetConfigString

This function returns a calibration string which indicates the current configuration of the instrument.

C Prototype: `void Instrument_GetConfigString( LPSTR lpString )`
VB Prototype: Declare Sub VB_Instrument_GetConfigString Lib "xr8000.dll" (ByVal pRet As String)

The string that gets returned plays by the following rules

Position	Title	Description
1	R or T	R = Reflectance (R) or Transmittance (T)
2	Angle	Included (I), Excluded (E) or Both (B)
3	Aperture	Small (S), Medium (M), Large (L)
4	Correction	"C" if SP64 correction is enabled
5	Filter	OPN = Open SP1 = SP64 Whiteness SP2 = SP64 DayGlo UV1 = UV 400 UV2 = UV420 UV1P-x% = UV1 partially inserted at x percent

6. Instrument_Measure

This function takes a measurement and stores the reflectance data. This data can be obtained by calling the Instrument_GetReflectances function. The function performs the following operations:

1. Put the instrument into the configuration currently specified.
2. Check to see if the aperture matches the port reducer and prompt if not.
3. Check to see if instrument is calibrated for this configuration and prompt if not.
4. Take the measurement.
5. Store the incoming reflectance or transmittance data.

C Prototype: `BOOL Instrument_Measure()`
VB Prototype: Declare Function VB_Instrument_Measure Lib "xr8000.dll" () As Integer

The function returns TRUE if measurement data has been obtained; otherwise FALSE if the user did not want to do the measurement, or if some other problem took place.

7. Instrument_Calibrate

This function takes the instrument in its currently specified configuration and performs a calibration sequence. This sequence is incorporated in a wizard-like dialog that steps the user through the entire process.

C Prototype: `void Instrument_Calibrate()`
VB Prototype: Declare Sub VB_Instrument_Calibrate Lib "xr8000.dll" ()

8. Instrument_GetReflectances

This function returns the reflectance (or transmittance, depending on the current configuration) data obtained from the last measurement using the Instrument_Measure command.

C Prototype: `void VB_Instrument_GetReflectances( float* pRefl )`
VB Prototype: Declare Sub VB_Instrument_GetReflectances Lib "xr8000.dll" (ByRef pRefl As Single)

In VB, this first parameter should point to the first element of a 39-element array, which gets filled with reflectance data from 360-740 nm in 10 nm increments. In C or C++, pass a pointer to an array of 39 elements.

Note: In this case, note that the VB function is used for both C/C++ and Visual Basic applications. The Instrument_GetReflectances function returns data in a 45-element array designed to run with a 320-780 nm range. Any reflectance values not used are set to –1. This function was designed for X-RiteColor Master® and is therefore not recommended for general use. Please use VB_Instrument_GetReflectances instead.

9. Instrument_GetLoWavelength

This function returns the low wavelength of the XR8000 instrument, which is currently set to 360 nm.

C Prototype: `int Instrument_GetLoWavelength()`
VB Prototype: Declare Function VB_Instrument_GetLoWavelength Lib "xr8000.dll" () As Integer

10. Instrument_GetHiWavelength

This function returns the high wavelength of the XR8000 instrument, which is currently set to 740 nm.

C Prototype: `int Instrument_GetHiWavelength()`
VB Prototype: Declare Function VB_Instrument_GetHiWavelength Lib "xr8000.dll" () As Integer

11. Instrument_GetModel

This function returns the instrument model number as well as the configuration string.

C Prototype: `void Instrument_GetModel( LPSTR pModel )`
VB Prototype: Declare Sub VB_Instrument_GetModel Lib "xr8000.dll" (ByVal pRet As String)

An example of this string in a small aperture, included data, with no filter would be
`<XR>8000 RIS-OPN`.

12. Instrument_GetSerialNo

This function returns the instrument serial number. This is a six-digit hexadecimal number.

C Prototype: `void Instrument_GetSerialNo( LPSTR pSerialNo )`
VB Prototype: Declare Sub VB_Instrument_GetSerialNo Lib "xr8000.dll" (ByVal pRet As String)

13. Instrument_IsReflectance

This function returns TRUE if the configuration is set for reflectance, and FALSE if set for transmittance.

C Prototype: `BOOL Instrument_IsReflectance()`
VB Prototype: Declare Function VB_Instrument_IsReflectance Lib "xr8000.dll" () As Integer

14. Instrument_CaptureVideo

This function is used to display a window containing the camera view on the instrument. The prototype indicates how this function should be used:

C Prototype: `BOOL Instrument_CaptureVideo( HWND hParent )`

VB Prototype: Declare Function VB_Instrument_CaptureVideo Lib "xr8000.dll"
(ByVal hWnd As Long) As Integer

Pass the window handle of a window to the function, and the camera output will be displayed in that window. You must call Instrument_ReleaseVideo when finished viewing the camera output, typically prior to when that window gets destroyed. The function returns TRUE if successful, otherwise it returns FALSE.

15. Instrument_ReleaseVideo

This function should be called when the camera display is no longer needed or the window containing that camera data is about to be destroyed.

C Prototype: `BOOL Instrument_ReleaseVideo()`
VB Prototype: Declare Function VB_Instrument_ReleaseVideo Lib "xr8000.dll" () As Integer

The function returns TRUE if successful, otherwise it returns FALSE.

16. Instrument_IsDataAvailable

This function can be used to see if reflectance/transmittance data is available to be read from the instrument. You can use this function in conjunction with the Instrument_GetReflectances function.

C Prototype: `BOOL Instrument_IsDataAvailable()`
VB Prototype: Declare Function VB_Instrument_IsDataAvailable Lib "xr8000.dll" () As Integer

The function returns TRUE if data is available, otherwise it returns FALSE. When data is read using Instrument_GetReflectances, the read buffer is cleared and this function will return FALSE.

17. Instrument_SetLanguage

This function can be used to change the language of the XR8000 library to match that of the calling thread.

C Prototype: `BOOL Instrument_SetLanguage()`

VB Prototype: Declare Function VB_Instrument_SetLanguage Lib "xr8000.dll"
(ByVal lcid As Long) As Integer

The function returns TRUE if the language resource DLL could be loaded (XR8000.009 for English, XR8000.007 for German, etc.) and FALSE if it could not be found or could not be loaded. This function would be useful if the calling application can change languages “on the fly” and needs to call XR8000 to change over and match the receptor language.

Development Requirements:

Use the XR8000.DLL in your application to access all the functions in the library. If working in C++, use the GetProcAddress function to obtain a pointer to each function in the library. For example, the following lines of code can be used in C++ to gain access to the `Instrument_Initialize` function:

```
HINSTANCE hLibrary = LoadLibrary("xr8000.dll");

typedef BOOL INSTRUMENT_INITIALIZE();

INSTRUMENT_INITIALIZE* Instrument_Initialize;

Instrument_Initialize =
      (INSTRUMENT_INITIALIZE*)GetProcAddress( s_hBenchtop,
      "Instrument_Initialize" );

// Now that we have a working function, we can call it
if( Instrument_Initialize )
{
    Instrument_Initialize();
}
```

If you implement a solution in Visual Basic 6.0, you can simply create Declare statements in the declarations section of a code module, as shown below.

```
Declare Function VB_Instrument_Standalone Lib "xr8000.dll" () As Integer

Declare Function VB_Instrument_Initialize Lib "xr8000.dll" () As Integer

Declare Sub VB_Instrument_Configure Lib "xr8000.dll" ()

Declare Sub VB_Instrument_SetConfiguration Lib "xr8000.dll" (ByVal nAperture As Long,
             ByVal nFilter As Long, ByVal fPctEnergy As Single,
             ByVal bReflectance As Boolean, ByVal nAngle As Long)

Declare Sub VB_Instrument_GetConfigString Lib "xr8000.dll" (ByVal pRet As String)

Declare Function VB_Instrument_Measure Lib "xr8000.dll" () As Integer

Declare Sub VB_Instrument_Calibrate Lib "xr8000.dll" ()

Declare Sub VB_Instrument_GetReflectances Lib "xr8000.dll" (ByRef pRefl As Single,
             ByVal iAngle As Long)

Declare Function VB_Instrument_GetLoWavelength Lib "xr8000.dll" () As Integer

Declare Function VB_Instrument_GetHiWavelength Lib "xr8000.dll" () As Integer

Declare Sub VB_Instrument_GetModel Lib "xr8000.dll" (ByVal pRet As String)

Declare Sub VB_Instrument_GetSerialNo Lib "xr8000.dll" (ByVal pRet As String)

Declare Function VB_Instrument_IsReflectance Lib "xr8000.dll" () As Integer

Declare Function VB_Instrument_IsDataAvailable Lib "xr8000.dll" () As Integer

Declare Function VB_Instrument_SetLanguage Lib "xr8000.dll" (ByVal lcid As Long) As
Integer

Declare Function VB_Instrument_CaptureVideo Lib "xr8000.dll" (ByVal hWnd As Long) As
Integer

Declare Function VB_Instrument_ReleaseVideo Lib "xr8000.dll" () As Integer
```

These functions can then be called from anywhere within Visual Basic. For a working example, see the sample VB test application named XR8000Text.vbp located in the installation directory of the SDK.