Integrating 3rd Party Applications with EyeTracking EyeWorks™ Software

EyeWorks enables developers to integrate their 3rd party applications (including E-Prime®) via an easy-to-use protocol. To integrate applications with EyeWorks requires a basic to moderate level of programming expertise in the language you intend to use, as well as the necessary compiler and associated tools.

Communicating with EyeWorks is always done via the TCP/IP protocol, whether the client application is running on the same machine or on a remote machine on the same network. Client applications connect to the Data Server hosted within EyeWorks Record, which requires an EyeWorks license at either Pro or Premier level.

Functions Available via the EyeWorks Data Server:

- Receive data in real-time: Your application can receive a real-time stream of point-of-gaze and pupil data, as generated by the eyetracker, and Cognitive Workload data, as generated by EyeWorks Record (if a valid license has been purchased for the Workload Module).
  - Possible uses include: store data in your own custom database, integrate data with your own applications, use data to prototype ‘human-in-the-loop’ systems, augmented cognition systems, and adaptive systems.
- Send synchronization events to EyeWorks: While EyeWorks is able to present most stimuli to subjects during testing, it is not able to control your own custom applications. For example, let’s say you have a Windows® application that you have written with varying screens and you want to be able to analyze eye data on specific screens. You could integrate your application with EyeWorks so that it sends event markers to EyeWorks each time the subject goes to the screen(s) of interest. While analysis can be performed without these event markers using EyeWorks Analyze, it is much less time consuming to set up the analysis project when the timing information is automatic.

Ways to Connect to EyeWorks Data Server:

1. **AppConnect SDK** is supplied with EyeWorks and is a simple to use SDK that will enable most developers to achieve integration with EyeWorks very rapidly. The SDK is supplied as a standard Windows C++ DLL that can be called from other C / C++ Windows applications, CSharp and Visual Basic applications, and even E-Prime experiments.
2. **Writing a custom TCP/IP application.** This is a more advanced approach and only recommended for more advanced developers. The application must adhere to the EyeWorks Data Server protocol.

E-Prime® is a registered trademark of Psychology Software Tools, Inc.
Using the AppConnect SDK

Introduction to the SDK

The AppConnect SDK can be used for integration with EyeWorks Data Server (an integral component of EyeWorks Pro and Premier Editions). The SDK is written for use by applications running on Microsoft Windows XP 32 bit and Vista 32 bit versions.

It is important to note that the Data Server will send data to connected clients at any time while an EyeWorks Record session is running. However, event markers can only be sent to EyeWorks Record when a testing script is not being used in that session.

AppConnect SDK License Terms

You may use the AppConnect redistributable DLLs (as listed in ‘Running and Redistributing Your Application’ below) solely on a revocable and non-exclusive basis to allow connectivity between your custom applications and legally licensed copies of EyeTracking EyeWorks™ software per the supplied documentation. THIS SOFTWARE SDK IS PROVIDED “AS IS”.

AppConnect Sample Applications

EyeWorks includes three example applications with source code that demonstrate how to use the AppConnect SDK. Each example performs the same function but is written in either CSharp, Visual Basic, or C++.

The sample is available as a Microsoft Visual Studio 2008 solution found at:

C:\Program Files\EyeTracking\EyeWorks\ClientTools\Samples\AppConnect\AppConnectExample_2008.sln

This location may be different if you selected an alternative location when installing EyeWorks

Building Your First AppConnect Application (Examples below in CSharp)

Start Microsoft Visual Studio and create a new CSharp ‘Windows Forms Application’ project. Add three buttons to the form and name them appropriately for their function. The functions for these buttons will be 1. To Connect to the Data Server, 2. To Send events to the Data Server and, 3. To Disconnect from the Data Server.
To enable use of the AppConnect SDK in your application you must first import the commands from the AppConnect SDK. Within the class for your new application’s Main Form (Form1) use Import to declare these functions.

**Example: Import functions from AppConnect SDK**

```csharp
namespace AppConnectEx_CS
{
    public partial class Form1 : Form
    {
        // IMPORT SECTION (Imports required functions from EyeWorks App Connect Library
        [DllImport("appconnect.dll", CharSet = CharSet.Ansi,
            EntryPoint = "ConnectToServer",
            CallingConvention = CallingConvention.StdCall)]
        public static extern int ConnectToServer(string lpAddress, UInt32 nPort);

        [DllImport("appconnect.dll", CharSet = CharSet.Ansi,
            EntryPoint = "CloseConnection",
            CallingConvention = CallingConvention.StdCall)]
        public static extern void CloseConnection();

        [DllImport("appconnect.dll", CharSet = CharSet.Ansi,
            EntryPoint = "SendEventA",
            CallingConvention = CallingConvention.StdCall)]
        public static extern void SendEventA(string lpText);
    }
}
```

Once all functions have been imported, they are now accessible from within your code.

In the code behind your Connect button, call the **ConnectToServer** function passing in the IP address of the computer running EyeWorks Record, as well as the port number that the Data Server is running on. If your application is running on the same computer as EyeWorks Record you can use the “127.0.0.1” loopback IP address as the connection string. The default Data Server port is 14740 although this can be changed in the Data Server tab of EyeWorks Record. If the call to **ConnectToServer** is successful it will return a non-zero number. A return value of zero signifies failure to connect.

Once a connection is made to EyeWorks Record via the Data Server, you will be able to see your computer listed as a connected client on the Data Server screen in EyeWorks Record. Connections can be made to the Data Server at any point once EyeWorks Record is running as long as the Data Server is started.
Example: Connecting your application to the EyeWorks Data Server using the AppConnect SDK

```csharp
private void ButtonConnect_Click(object sender, EventArgs e)
{
    string ipAddress;
    uint nPort;

    ipAddress = "127.0.0.1";
    nPort = 14740;

    try
    {
        ipAddress = TextBoxIP.Text.Trim();
        nPort = Convert.ToUInt32(TextBoxPort.Text.Trim());
    }
    catch (Exception ex)
    {
    }

    // make a network connection to the EyeWorks server
    _connected = ConnectToServer(ipAddress, nPort) != 0;

    if (!_connected)
    {
        MessageBox.Show("Could not connect to EyeWorks Record at IP " +
            ipAddress.ToString() +
            " port " + nPort.ToString() + ");
    }
}
```

Each client application should only make one connection to the Data Server at any one time. Once a client has completed its communications with the Data Server, a client can gracefully close the connection by calling the `CloseConnection`. In our example you should add this call to the code behind for your Disconnect button.

Example: Disconnecting your application from the EyeWorks Data Server using the AppConnect SDK

```csharp
private void ButtonDisconnect_Click(object sender, EventArgs e)
{
    // when our application is done, be sure to close down the connection
    CloseConnection();

    _connected = false;
}
```

Once a connection has successfully been established between EyeWorks Record and your application it is possible to send event markers from your application for synchronization with the eye data using the `SendEventA` function. These event markers will be logged by EyeWorks Record for use in the analysis stage. In EyeWorks Analyze, when importing data containing your custom markers, the markers will be used to segregate the data you collected into tasks. These tasks will be named as per the event markers you provided during the collection phase.
Some conditions must be adhered to when using the **SendEventA** function:

1. Event markers must not be longer than 16 characters in length.

2. Events can only be sent when a session is running (data is being collected) and a script is not being used for that data collection session.

3. The string passed into the SendEventA function must be in ANSI format (not Unicode) – this is taken care of for Visual Basic and CSharp users in the import declaration, however, C++ users should see the included samples to see how to ensure proper use of the function.

4. All events sent during a given recording session (the time between when a given data session is started and stopped) must be unique.

---

**Example: Sending event markers to the EyeWorks Data Server using the AppConnect SDK**

```csharp
private void ButtonSend_Click(object sender, EventArgs e)
{
    string EventID;
    // make up a unique marker to put into the EyeWorks log file
    EventID = "Task " + _counter.ToString();
    // send the task marker to EyeWorks
    // NOTE: The task marker MUST be unique in
    // every call during the EyeWorks session
    SendEventA(EventID);
    _counter = _counter++;
}
```

---

**Special Considerations for integrating from E-Prime® by PST**

E-Prime uses a programming language called E-Basic which is very similar to Visual Basic. Using E-Basic it is possible to synchronize timing information from your E-Prime experiments with EyeWorks. The declaration of the AppConnect functions in E-Basic are slightly different, and should be made in the code behind for your E-Prime experiment. An example of how the declarations should appear are as follows (be aware that the AppConnect Dll must reside in a default DLL load path, otherwise a full path can be specified to the AppConnect.dll in the function declaration):

```csharp
' IMPORT SECTION (Imports required functions from EyeWorks App Connect Library
Declare Function ConnectToServer CDecl Lib "AppConnect.dll" (ByVal lpAddress As String, ByVal nPort As Long) As Boolean
Declare Sub CloseConnection CDecl Lib "AppConnect.dll" ()
```
Declare Sub SendEventA CDecl Lib "AppConnect.dll" (ByVal lpText As String)

Once the functions are declared, they may be used as the script runs as in e.g.

Dim result as Boolean
Dim ip as String
Dim port as Long

' open connection with EyeTracking EyeWorks Server
ip = "127.0.0.1"
port = 14740

result = ConnectToServer(ip, port)

Running and Redistributing Your Application

If your application uses the AppConnect SDK, you will need to make sure that you have the necessary components installed before you run your application.

1. Copy the following binary DLL files to the same directory as your application:
   a. AppConnect.dll

Important: This file must be copied from the directory located at [EyeWorks Install Directory]\ClientTools\Redistributables\AppConnect\. In most cases, EyeWorks will by default be installed at c:\program files\EyeTracking\EyeWorks\.

2. The AppConnect binary files rely on Microsoft’s Visual C++ run-time and MFC runtime version 9.0. You will need to install Microsoft’s MFC9 redistributable package on any computer that you wish to run your application on if it does not already contain these files. The redistributable package is available from Microsoft’s website.

3. Make sure EyeWorks Record is running on your network and its Data Server is started (see the User Manual for further information). Run your application and connect it to the Data Server.
Advanced Functionality - Receiving Eye and Workload Data from EyeWorks

There are two ways to receive data using the AppConnect SDK, 1. the callback method or, 2. the polling method.

Using the AppConnect SDK Callback Method to receive data from EyeWorks

To receive data from EyeWorks required additional steps to be taken:

1. Connect to EyeWorks
2. Register callbacks to handle data receipt
3. Instruct EyeWorks Record to send data
4. Handle incoming data
5. Instruct EyeWorks to Stop sending data
6. Disconnect from EyeWorks

In your existing sample project you must first import the additional functions that will be used in order to receive data from the App Connect library.

Additional function imports required from AppConnect.dll

```csharp
[DllImport("appconconnect.dll", CharSet = CharSet.Ansi,
           EntryPoint = "RequestData",
           CallingConvention = CallingConvention.StdCall)]
public static extern bool RequestData();

[DllImport("appconconnect.dll", CharSet = CharSet.Ansi,
           EntryPoint = "StopData",
           CallingConvention = CallingConvention.StdCall)]
public static extern bool StopData();

[DllImport("appconconnect.dll", CharSet = CharSet.Ansi,
           EntryPoint = "SetEyeData_CB",
           CallingConvention = CallingConvention.StdCall)]
public static extern void SetEyeData_CB (EyeDataCallback eyeDataCB);

[DllImport("appconconnect.dll", CharSet = CharSet.Ansi,
           EntryPoint = "SetICA_CB",
           CallingConvention = CallingConvention.StdCall)]
public static extern void SetICA_CB (ICACallback icaCB);
```

The next step is to set up the callback functions (in C / C++) or the delegates (.NET) that will be called when the App Connect library receives data from EyeWorks Record. Declare the delegates outside of
your class, and then make an instance of the two delegates (one for eye data, and one for workload data) inside of your class).

**Definition of Delegates**

EyeDataCBHandler:

```c
unsigned int nTime, // sample timestamp in milliseconds
float fLX, // left eye gaze intersection coordinate (x-component)
float fLY, // left eye gaze intersection coordinate (y-component)
float fLP, // left eye pupil diameter
float fRX, // right eye gaze intersection coordinate (x-component)
float fRY, // right eye gaze intersection coordinate (y-component)
float fRP // right eye pupil diameter
```

ICACBHandler:

```c
unsigned int nTime, // sample timestamp in milliseconds (corresponds to the timestamp of the FIRST sample in the computed second)
double dLICA, // ICA Workload computed for left eye for second
double dRICA // ICA Workload computed for right eye for second
```

**Further Information on Callback Functions:**

**EyeDataCBHandler:** The gaze intersection coordinates are pixel coordinates of the screen intersected – where (0,0) is the top left of the screen. The max intersection is the maximum resolution of the screen.

**ICACBHandler:** The left and right workload values are generated second by second independently. The workload value ranges between 0.0 and 1.0 when processing is completed successfully. A value of -99 indicates that there was no valid pupil data available for that second to enable ICA processing (effectively ‘N/A’ or NaN). Further, depending on the task, you may want to evaluate the ICA by keeping a running average of the ICA over 5 seconds, 10 seconds… a minute. The level of averaging that is relevant is task / study dependent. Ideally second by second data would be stored then the data can be viewed with or without averaging etc. during the analysis phase.

**Declaration of Delegate Instances**
// callbacks
private EyeDataCallback eyeCB;
private ICACallback icaCB;

public Form1()
{
    InitializeComponent();
}

private void Form1_Load(object sender, EventArgs e)
{
    _counter = 1;
    _connected = false;
    eyeCB = new EyeDataCallback(SetEyeData_CBHandler);
    icaCB = new ICACallback(SetICA_CBHandler);

    Now, after you have made the ConnectToServer call from the App Connect library, register your callbacks / delegates with the library, then call RequestData to instruct EyeWorks Record to begin sending data. As data arrives, your delegate handler functions will be called with the new data passed in as parameters.

    Register the Callbacks and Request Transmission of New Data

    // make a network connection to the EyeWorks server
    _connected = ConnectToServer(ipAddress, nPort) != 0;

    if (!_connected)
    {
        MessageBox.Show("Could not connect to EyeWorks Record at IP
    }
    SetEyeData_CB(eyeCB);
    SetICA_CB(icaCB);
    RequestData();

    Lastly, upon termination of the network connection, be sure to call StopData to instruct EyeWorks to stop sending data. Your application can now shutdown cleanly.
Using the AppConnect SDK Polling Method to receive data from EyeWorks

To receive data from EyeWorks using polling the following sequence should be followed:

1. Connect to EyeWorks
2. Instruct EyeWorks Record to send data
3. Poll for data
4. Instruct EyeWorks to Stop sending data
5. Disconnect from EyeWorks

When using the AppConnect SDK to poll for data, it is important to note that the `getdata` function(s) return the most recent data received by the SDK at the time the call to retrieve data is made. These methods are not buffered.

Additional function imports required from AppConnect.dll

```csharp
[DllImport("appconnect.dll", CharSet = CharSet.Ansi,
    EntryPoint = "GetCurrentWorkloadData",
    CallingConvention = CallingConvention.StdCall)]
public static extern void GetCurrentWorkloadData (ref UInt32 time,
    ref double ICAL, ref double ICAR);

[DllImport("appconnect.dll", CharSet = CharSet.Ansi,
    EntryPoint = "GetCurrentEyeData",
    CallingConvention = CallingConvention.StdCall)]
public static extern void GetCurrentEyeData(ref UInt32 time,
    ref float fLX, ref float fLY, ref float fLA,
    ref float fRX, ref float fRY, ref float fRA);
```
Questions & Answers

What is the latency I can expect when using the Data Server?

Some latency may be present in the data stream depending on factors such as speed of the server and client computers, network traffic and architecture, and delay in the eyetracker generating the raw data. Typically a latency of < 200ms is normal, although the actual delay will be different in every setup. For the purposes of clarification the term ‘latency’ as discussed here is defined as the time from which the eyetracking camera took a sample of the eye until the data sample arrives across the network for the client application to access the corresponding sample generated from the data.

Will I have to change my code if I use a different eyetracker?

No. As long as EyeWorks is compatible with the different eyetrackers you are using, your client code will not have to change. However, you should be aware that some eyetrackers only report eyetracking data on one eye, some may not report pupil data, and some may not be compatible with the Cognitive Workload Module. In any of these cases, the corresponding data will not be present in the data stream.

My task runs on Mac OS or Linux, can I still integrate my application with EyeWorks?

Yes. Currently EyeTracking does not supply an SDK for these operating systems, however, it is still possible to write your own TCP/IP application for these operating systems, and have them comply with the EyeWorks Data Server protocol.