CANFIELD INTEGRATION SDK DEVELOPER'S GUIDE

DOCUMENT VERSION: 2.37

MARCH 18, 2010

REVISION CHART

Version	Primary Author(s)	Description of Version	Date Completed
1.0	SVS	Initial creation	12/3/02
1.1	SVS	With all functionality to be included in the version 6.1 release	
1.2	SVS	Version 6.1 Pre-Release	12/24/02
1.3	SVS	Version 6.1 Release – add DateCreated, DateModified properties to CanfieldSDKPatient object	1/9/03
1.4	SVS	Version 6.10 Release - add Open method to CanfieldSDKApp object and implement ThumbnailCount on CanfieldSDKPatient object, add filter parameters to Load method on CanfieldSDKPatientList object	3/20/03
1.5	SVS	Version 6.10 Release – add GetAppInfo method on CanfieldSDKApp object to determine whether an AppID is installed and get its name and path without affecting the active application object	3/24/03
1.6	SVS	Version 6.10 Release – implement 0-based index LoadThumbnail on CanfieldSDKPatient object	3/25/03
1.7	SVS	Add documentation for menu modification methods.	3/27/03
1.8	SVS	Add visible property to CanfieldSDKAppObj so app window state can be determined. Add Hwnd property that returns the HWND of the connected application for low-level access. Add IsOpen method that returns whether the application is open.	6/30/03
1.9	SVS	Add CanfieldSDK object with SDKVersion, SDKVersionMajor, and SDKVersionMinor SDK version info properties to allow for retrieval of SDK version info	7/10/2003
2.0	SVS	add Image List and Image objects for push/ pull of full image data; add event manager object to allow for configuration of event notifications and triggering	10/10/2003
2.1	SVS	add SetImageBLOBFromFile method	4/20/04
2.22	SVS	update version to match Mirror release	5/24/04
2.23	SVS	add LoadSinglePatient method documentation	10/21/04
2.24	SVS	add Database list stuff to Code Examples section	12/16/04
2.25	SVS	add section on C++ to VB sample code translation	12/22/04
2.26	SVS	add CreatePatient demo code to 2.2.2 Code Example	1/4/04
2.27	SVS	Add Race property to CanfieldSDKPatient object, add LoginUI method to CanfieldSDKApplication object	3/25/05
2.28	SVS	Add SelectImages method to CanfieldSDKApplication object	4/29/05
2.29	SVS/AB	Add enhancements for KAA project	1/13/06
2.30	SVS	Add new patient object properties: DateCreatedImage,	4/2/08

Version	Primary Author(s)	Description of Version	Date Completed
		DateModifiedImage, DateModifiedLast.	
2.31	SVS	Add new image object properties: UserCreate, UserModified, DateImage, PatientAge,ImageNumber; implement DateCreate, DateModified	5/16/08
2.33	SVS	Add new image tag properties methods and properties	9/15/08
2.34	SVS	Update documentation on app timeout property	12/8/08
2.35	KB	Identify unsupported objects/methods	12/30/08
2.36	SVS	Add Notes property on CanfieldSDKImage object	4/9/09
2.37	KB	Add .NET sample code	3/18/10

TABLE OF CONTENTS

1. OVERVIEW

This document is intended as a guide for software developers wishing to integrate an external software system (electronic medical records, practice management, etc.) with Canfield imaging software using the Canfield Imaging Systems Integration SDK (Canfield SDK). The Canfield SDK includes, among other features, the ability to create and modify patient records in the Canfield system, retrieve images from the Canfield system and open Canfield's Mirror application to a specified patient chart.

The Canfield SDK is deployed as a collection of Windows COM objects. As such, the Canfield SDK can be accessed from any development platform that supports COM (e.g., C#, VB.NET, C++, VB 6.0). The Canfield SDK COM objects are installed and registered by the Mirror software installer. The Mirror application must be installed on any Windows system that will be used to access Canfield SDK functionality.

2. INSTALLATION AND INTEGRATION NOTES

2.1 Getting Started

Before writing code to use the Canfield SDK, contact Canfield Imaging Systems to obtain the following:

- Mirror application installer
- License key
- AppKey value

2.2 General Notes

2.2.1 Object Lifetimes

Typically you will create a single instance of CanfieldSDKApp, initialize it once and keep it around to use as needed. Other Canfield SDK objects should generally be used in the narrowest scope possible.

2.3 C# (.NET) Code Examples for Common Tasks

2.3.1 Opening an SDK Connection

```
int CanfieldAppId = 1; // Always use 1 (Mirror)... other apps not currently implemented

// Update the following to match the version of Canfield SDK used during integration development
int CanfieldMinVersionMajor = 7;
int CanfieldMinVersionMinor = 3;

String AppKey = "{Obtain an AppKey value from Canfield}";
String AppVendor = "Integrator's Company Name Here"; // the name of the integrating company (not Canfield)
String AppName = "Integrator's Product Name Here"; // the name of the integrating application (not Canfield's app)
String AppVersion "Integrator's Product Version Here"; // the version of the integrating app (not Canfield's app)

CanfieldSDKApp sdkApp;

try
{
    sdkApp = new CanfieldSDKApp();
    sdkApp.AppID = CanfieldAppId;
    sdkApp.Initialize(AppKey, AppName, AppVendor, AppVersion);

    CanfieldSDK sdk = new CanfieldSDK();
    if (sdk.SDKVersionMajor < CanfieldMinVersionMajor
        || sdk.SDKVersionMajor == CanfieldMinVersionMajor && sdk.SDKVersionMinor < CanfieldMinVersionMinor)
        MessageBox.Show("The installed Canfield software is an unsupported version.");

    // Open a connection to the Canfield SDK. The SDK will do this automatically as needed,
    // but if the app isn't already open there can be a 30-90 second time hit,
    // so it's a good idea to do this explicitly during initialization.
    if (!sdkApp.IsOpen())
        sdkApp.Open();
}
catch (System.Runtime.InteropServices.COMException ex)
{
    // SDK errors generally throw COMException
}
```

2.3.2 Adding & Updating Patient Records

```
// sdkApp is a CanfieldSDKApp object initialized as in "Opening an SDK Connection" above

// Generic DataRow object used for illustration purposes.
// In practice a properly typed object would be used instead.

public void AddCanfieldPatient(ref DataRow patient)
{
    String lastName = patient["LastName"] as String;

    // must have at least a Last Name to create a Canfield patient record
    if (String.IsNullOrEmpty(lastName))
        throw new ArgumentException("The supplied patient record must have a value for LastName");

    CanfieldSDKPatient sdkPatient = sdkApp.CreatePatient(lastName, null, null) as CanfieldSDKPatient;
    if (sdkPatient != null && !String.IsNullOrEmpty(sdkPatient.ID))
    {
        // store the foreign key data needed to access this Canfield patient record later
        patient["CanfieldId"] = sdkPatient.ID;
        CanfieldSDKDatabase sdkDb = (CanfieldSDKDatabase)sdkApp.GetDatabase(null);
        patient["CanfieldDb"] = sdkDb.Name;

        // push all data into the new Canfield patient record
        UpdateCanfieldPatient(patient);
    }
    else
    {
        // handle errors here
    }
}

public void UpdateCanfieldPatient(DataRow patient)
{
    // retrieve the correct Canfield record
    CanfieldSDKPatient sdkPatient;
    String canfieldId = patient["CanfieldId"] as String;
    sdkPatient = ((CanfieldSDKPatientList)sdkApp.GetPatientList()).GetAt(canfieldId) as CanfieldSDKPatient;

    if (sdkPatient != null)
    {
        // set all data fields
        sdkPatient.LastName = patient["LastName"] as string;
        sdkPatient.FirstName = patient["FirstName"] as string;
        sdkPatient.MI = patient["Middle"] as string;
        sdkPatient.AltID = patient["MRN"] as string;

        // save the updated Canfield record
        sdkPatient.Save();
    }
    else
    {
        // handle patient not found errors here
    }
}
```

2.3.3 Retrieving Thumbnail Images

```
// sdkApp is a CanfieldSDKApp object initialized as in "Opening an SDK Connection" above
// patient is a record from the integrating app with the Canfield record number stored in a "CanfieldId" field

// retrieve the correct Canfield record using stored Canfield ID
CanfieldSDKPatient sdkPatient;
String canfieldId = patient["CanfieldId"] as String;
sdkPatient = ((CanfieldSDKPatientList)sdkApp.GetPatientList()).GetAt(canfieldId) as CanfieldSDKPatient;

if (sdkPatient != null)
{
    if (sdkPatient.ThumbnailCount > 0)
    {
        // just grabbing first image for illustration purposes
        int index = 0;
        CanfieldSDKThumbnail sdkThumbnail = sdkPatient.LoadThumbnail(index) as CanfieldSDKThumbnail;
        if (sdkThumbnail != null)
        {
            // get IPictureDisp object from the Canfield system and
            // convert to Image object using Microsoft.VisualBasic.Compatibility.VB6.Support.IPictureDispToImage
            System.Drawing.Image image;
            image = Support.IPictureDispToImage(sdkThumbnail.GetPicture(false, false));

            // do something with the retrieved image here
        }
        else
        {
            // handle errors here
        }
    }
    else
    {
        MessageBox.Show("The current patient record has no associated images.");
    }
}
else
{
    // handle patient not found errors here
}
```

2.3.4 Opening the Canfield Patient Chart

```
// sdkApp is a CanfieldSDKApp object initialized as in "Opening an SDK Connection" above
// patient is a record from the integrating app with the Canfield record number stored in a "CanfieldId" field

String canfieldId = patient["CanfieldId"] as String;

// make sure patient can be accessed before opening chart
CanfieldSDKPatient sdkPatient;
sdkPatient = ((CanfieldSDKPatientList)sdkApp.GetPatientList()).GetAt(canfieldId) as CanfieldSDKPatient;

if (sdkPatient != null)
{
    sdkApp.OpenChart(canfieldId, false);
}
else
{
    // handle patient not found errors here
}
```

2.4 C++ Code Examples for Common Tasks

This section of the documentation provides example code for common tasks provided in C++. General guidelines for translating these examples to VB are also provided. A type library (CanfieldSDK.tlb) is available to facilitate Smart Pointer usage from Visual C++ development environments or through use of the Object Browser in Visual BASIC development environments. For example, to use the SDK objects in Visual C++ code using the "Smart Pointer" implementation the following code can be used:

```
#import "CanfieldSDK.tlb" no_namespace named_guids


CComPtr<ICanfieldSDKApp>pApp;
HRESULT hRes = S_OK;
try
{
    hRes = pApp.CoCreateInstance(CLSID_CanfieldSDKApp);
    if (SUCCEEDED(hRes))
    {
        pApp->AppID = ICanfieldSDK_AppID_MirrorSuite;
        // call other methods here
}
}
catch (_com_error &e)
{
    AfxMessageBox(e.ErrorMessage(), MB_OK | MB_ICONSTOP);
}
catch (...)
{

}
```

To use the objects from VBA, the following syntax can be used:

```
On Error GoTo HandleError
Dim strMsg
Dim objApp As Object
' if the CanfieldSDK type library is added to the browser
' with the "References" window the next line could alternatively be used
' so the methods and properties could be viewed in the VB environment
'Dim objApp As CANFIELDSDKLib.CanfieldSDKApp

Set objApp = CreateObject("CanfieldSDK.CanfieldSDKApp")
objApp.AppID = CANFIELDSDKLib.ICanfieldSDK_AppID_MirrorSuite


' call other object methods here

Exit function

HandleError:
strMsg = "Error: " & Str(Err.Number) & " - " & Err.Description
strMsg = strMsg & Chr$(13) & Chr$(10)
Debug.Print strMsg
Resume Next
End function
```

Note that all interface methods return HRESULT values. The return values have been omitted from the method documentation.

2.4.1 Opening Application Connections

```
// set to installed Canfield application ID
int nAppID = ICanfieldSDK_AppID_Mirror;

CComPtr<ICanfieldSDKApp>pApp;
try
{
    CString strAppKey(_T("integrator app key here… obtain from Canfield");
    CString strAppName(_T("integrator app name here");
    CString strAppVendor(_T("integrator app vendor (company) here");
    CString strAppVersion(_T("integrator app version here");

    HRESULT hRes = pApp.CoCreateInstance(CLSID_CanfieldSDKApp);
    if (SUCCEEDED(hRes))
    {
        pApp->AppID = nAppID;

        _variant_t varAppKey((LPCTSTR)strAppKey);
        _variant_t varAppName((LPCTSTR)strAppName);
        _variant_t varAppVendor((LPCTSTR)strAppVendor);
        _variant_t varAppVersion((LPCTSTR)strAppVersion);
        pApp->Initialize(varAppKey, varAppName, varAppVendor, varAppVersion);

        // open application connection
        pApp->Open();
    }
}
catch (_com_error &e)
{
    AfxMessageBox(e.ErrorMessage(), MB_OK | MB_ICONSTOP);
}
```

2.4.2 Creating / Modifying Patient Data

```
// note that pApp is an ICanfieldSDKApp object (see Opening Application Connections above)

// bstrID is patient ID that we are working with
_bstr_t bstrID = (LPCTSTR)strID;

// retrieve patient list object from application
IDispatchPtr pDispPatients = pApp->GetPatientList();
CComQIPtr<ICanfieldSDKPatientList>pPatients(pDispPatients);

try
{
    if (pPatients)
    {
        // load patient list if necessary
        pPatients->Load(_variant_t(true));

        CComPtr<IDispatch>pDispPatient = pPatients->GetAt(_variant_t(bstrID));
        // check whether patient exists
        if (pDispPatient)
        {
            CComQIPtr<ICanfieldSDKPatient>pPatient(pDispPatient);
            if (pPatient)
            {
                // save patient data
                // strMRN, strSSN, etc. are CString variables that were
                // populated from dialog controls input by the user
                pPatient->put_AltID(_bstr_t((LPCTSTR)strMRN));
                pPatient->put_SSN(_bstr_t((LPCTSTR)strSSN));

                pPatient->put_FirstName(_bstr_t((LPCTSTR)strFirstName));
```

```
                pPatient->put_LastName(_bstr_t((LPCTSTR)strLastName));

                pPatient->put_Address1(_bstr_t((LPCTSTR)strAddress1));
                pPatient->put_City(_bstr_t((LPCTSTR)strCity));
                pPatient->put_State(_bstr_t((LPCTSTR)strState));

                if (!strDOB.IsEmpty())
                {
                    COleDateTime dt;
                    try
                    {
                        dt.ParseDateTime(strDOB, VAR_DATEVALUEONLY);
                        pPatient->put_DOB((DATE)dt);
                    }
                    catch (...)
                    {
                        pPatient->put_DOB(0.0);
                    }
                }
                else
                {
                    pPatient->put_DOB(0.0);
                }

                pPatient->put_PostalCode(_bstr_t((LPCTSTR)strPostalCode));

                pPatient->Save();
            } // if (pDispPatient
            else
            {
                // patient does not exist, create new one
                CString strLName = _T("new last name");
                CString strFName = _T("new first name");

                // note that FName and MI are optional
                _variant_t varFName = (LPCTSTR)strFName;
                _variant_t varMI;

                    CComPtr<IDispatch>pDispPatient = pApp->CreatePatient(
                                                        _bstr_t((LPCTSTR)strLName),
                                                        varFName,
                                                        varMI);
                    if (pDispPatient)
                    {
                        CComQIPtr<ICanfieldSDKPatient>objPatient(pDispPatient);
                        if (objPatient)
                        {
                            // new patient created, change other properties here
                            // and call Save if anything is being changed

                            // retrieve system assigned ID
                            _bstr_t bstrNewID = objPatient->GetID();
                        }
                    }
            }
        } // if (pPatients
    }
    catch (_com_error &e)
    {
        AfxMessageBox(e.ErrorMessage(), MB_OK | MB_ICONSTOP);
    }
```

2.4.3 Reading Image Attribute Data

```
// lImage contains the index into a loaded patient list object
// of the desired image.  The image ID could also be used
// and passed to the GetAt method
// pImageList is an ICanfieldSDKImageList object already created
LONG lImage = 0L; // set to desired index
CComQIPtr<ICanfieldSDKImage>pImage(pImageList->GetAt(  varAppKey, _variant_t(lImage)));
if (pImage)
{
    CComQIPtr<ICanfieldSDKImageAttributes>pAttributes(pImage->GetImageAttributes());
    if (pAttributes)
    {
        // get image ID
        _bstr_t bstrImageID;
        pImage->get_ID(bstrImageID.GetAddress());

        CString strText;

        // get assigned attributes
        LONG lAttrCount = 0L;
        pAttributes->get_Count(&lAttrCount);
        for (LONG lAttr = 0; lAttr < lAttrCount; lAttr++)
        {
            CComQIPtr<ICanfieldSDKImageAttribute>pAttribute(pAttributes->GetAt(lAttr));
            if (pAttribute)
            {
                _bstr_t bstrAttrID = pAttribute->GetAttributeID();
                _bstr_t bstrAttrName = pAttribute->GetAttributeName();
                _variant_t varValue = pAttribute->GetValue();
                //do something with attribute data here
            }
        }
    }
}
```

2.4.4 Pushing / Pulling Images

2.4.4.1 Pushing a new image

```
try
{
    // note that strPatientID is associated patient for pushed image,
    // strAppKey is integrator's AppKey, and strFileName is full path
    // to existing image file to push
    CComPtr<ICanfieldSDKImageList>pImageList;
    HRESULT hRes = pImageList.CoCreateInstance(CLSID_CanfieldSDKImageList);
    if (SUCCEEDED(hRes))
    {
        _variant_t varAppKey((LPCTSTR)strAppKey);
        _variant_t varPatientID((LPCTSTR)strPatientID);

        // setup optional compression type and Quality factor
        // default compression is ICanfieldSDK_Compress_JPG
        // and default JPG quality is "best". Could use _variant_t()
        // for both of these.
        _variant_t varCompressType((long)nCompressType);
        _variant_t varQFactor((long)nQFactor);
        IDispatchPtr pDispImage = pImageList->PushImage(varAppKey,
                                                _bstr_t((LPCTSTR)strFileName),
                                                varPatientID,
                                                varCompressType,
                                                varQFactor);
        CComQIPtr<ICanfieldSDKImage>pImage(pDispImage);
```

```
        if (pImage)
        {
            CComBSTR bstrID;
            pImage->get_ID(&bstrID);
            strMsg.Format(  _T("image file: %s added, new ID: %s"),
                            (LPCTSTR)strFileName,
                            (LPCTSTR)_bstr_t(bstrID));
        }
    }
}
catch (_com_error &e)
{
    AfxMessageBox(e.ErrorMessage());
}
```

2.4.4.2 Retrieving a list of patient images

```
CComPtr<ICanfieldSDKImageList>pImageList;
HRESULT hRes = pImageList.CoCreateInstance(CLSID_CanfieldSDKImageList);
if (SUCCEEDED(hRes))
{

    CString strAppKey = _T("app key here");
    CString strAppName = _T("app name here");
    CString strAppVendor = _T("app vendor here");
    CString strAppVersion = _T("app version here");

    _variant_t varAppKey((LPCTSTR)strAppKey);
    _variant_t varPatientID((LPCTSTR)lpcszID);

    try
    {
        // note that varPatientID has been set up with
        // load the selected patient's images
        pImageList->Load(   varAppKey,
                            _variant_t(true),
                            varPatientID,
                            _variant_t(), // could supply last name filter here
                            _variant_t(), // could supply first name filter here
                            _variant_t(),
                            _variant_t(),
                            _variant_t());

        pImageList->get_Count(&lCount);
        bool bFirst = true;
        _bstr_t bstrRootPath;

        for (long lIndex = 0L; lIndex < lCount; lIndex++)
        {
            // retrieve image by index
            CComPtr<IDispatch>pDispImage = pImageList->GetAt(varAppKey,
                                                             _variant_t((long)lIndex));
            if (pDispImage)
            {
                CComQIPtr<ICanfieldSDKImage,
                            &IID_ICanfieldSDKImage>pImage(pDispImage);
                if (pImage)
                {
                    if (bFirst)
                    {
                        // get root location of images.
                        // ImagePath is relative to that
                        bstrRootPath = pImage->ImageRootPath;
                        CString strTemp((LPCTSTR)bstrRootPath);
                        if (strTemp.Right(1) != _T('\\'))
```

```
                    {
                        strTemp += _T('\\');
                        bstrRootPath = strTemp;
                    }

                    strImagePath = strTemp;

                    bFirst = false;
                }

                _bstr_t bstrFileName = pImage->ImagePath;
                _bstr_t bstrID = pImage->ID;

                // do something with image info here               }
            }
        }
    }
    catch (_com_error &e)
    {
        hRes = e.Error();
        AfxMessageBox(e.ErrorMessage());
    }
}
```

2.4.5 Pushing / Pulling Image BLOBs

2.4.5.1 Reading BLOB Info for an Image

```
    CComPtr<ICanfieldSDKImageList>pImageList;
    hRes = pImageList.CoCreateInstance(CLSID_CanfieldSDKImageList);
    if (SUCCEEDED(hRes))
    {
        CString strBLOBCount(_T(""));

        // strPatientID has been set up with the desired patient
        // and strImageID has been set up with desired image ID
        // to assign the BLOB to
        _variant_t varPatientID = (LPCTSTR)strPatientID;
        _variant_t varAppKey((LPCTSTR)strAppKey);
        _variant_t varImageID((LPCTSTR)strImageID);

        try
        {
            // load the desired patient's images
            pImageList->Load(   varAppKey,
                                _variant_t(false),
                                varPatientID,
                                _variant_t(),
                                _variant_t(),
                                _variant_t(),
                                _variant_t(),
                                _variant_t());
            IDispatchPtr pDispImage = pImageList->GetAt(varAppKey,
                                                        varImageID);
            if (pDispImage)
            {
                CComQIPtr<ICanfieldSDKImage,
                            &IID_ICanfieldSDKImage>pImage(pDispImage);
                if (pImage)
                {
                    long lBLOBCount = pImage->CountImageBLOB;
                    for (long i = 0L; i < lBLOBCount; i++)
                    {
                        _variant_t varIndex(i);
```

```
                    _variant_t varID;
                    _variant_t varIdOrIndex;
                    _variant_t varFilePath;
                    // retrieve all BLOB info for image
                    pImage->GetImageBLOBInfo(varAppKey,
                                             varIndex,
                                             // unique BLOB ID
                                                 //(also part of filename
                                                 &varID,
                                             // user-defined name
                                                 &varBLOBName,
                                             // BLOB file path
                                                 //relative to main image path
                                                 &varFilePath);
                    _bstr_t bstrFilePath = varFilePath;
                    // the full path from the rest of the dlg data
                    CString strFileName = (LPCTSTR)bstrFilePath);
                }
            }
        }
    }
}
catch (_com_error &e)
{
    AfxMessageBox(e.ErrorMessage());
}
```

2.4.5.2 Adding / Updating BLOBs for an Image

```
CComPtr<ICanfieldSDKImageList>pImageList;
hRes = pImageList.CoCreateInstance(CLSID_CanfieldSDKImageList);
if (SUCCEEDED(hRes))
{
    try
    {
        _variant_t varAppKey((LPCTSTR)strAppKey);
        // note that strID contains the ID of the image
        // to add/update the BLOB for and strPatientID
        // contains the patient ID that has the BLOB
        _variant_t varImageID((LPCTSTR)strID);
        _variant_t varBLOBName((LPCTSTR)strBLOBName);
        _bstr_t bstrFileName((LPCTSTR)strSourceFile);
        _variant_t varPatientID((LPCTSTR)strPatientID);

        // load the selected patient's images
        pImageList->Load(   varAppKey,
                            _variant_t(true),
                            varPatientID,
                            _variant_t(),
                            _variant_t(),
                            _variant_t(),
                            _variant_t(),
                            _variant_t());


        CComPtr<IDispatch>pDispImage = pImageList->GetAt(varAppKey, varImageID);
        if (pDispImage)
        {
            CComQIPtr<ICanfieldSDKImage>pImage(pDispImage);
            if (pImage)
            {
                pImage->SetImageBLOBFromFile( varAppKey,
                                              varBLOBName,
                                              bstrFileName);
                // tell the user it was successful
                CString strImageId((LPCTSTR)_bstr_t(varImageID));
```

```
                    CString strBLOBId((LPCTSTR)_bstr_t(varIdOrIndex));
                    CString strBLOBFile((LPCTSTR)bstrFileName);

                    CString strMsg;
                    strMsg.Format(  _T("BLOB Id: \'%s\' file: \'%s\'\n")
                                        _T("stored for image ID: \'%s\'"),
                                    (LPCTSTR)strBLOBId,
                                    (LPCTSTR)strBLOBFile,
                                    (LPCTSTR)strImageId);
                    AfxMessageBox(strMsg);
                }
            }
        }
        catch (_com_error &e)
        {
            hRes = e.Error();
            MessageBox(e.ErrorMessage(), _T("OnBnClickedOk _com_error"), MB_OK | MB_ICONSTOP);
        }
    }
```

2.4.6 Reading / Writing Image Attributes

** Note that this part of the SDK will not work when running in an IIS web service due to security restrictions on out of process COM object instantiation.

2.4.6.1 Adding an Attribute Value to an Image

```
try
{
    // note that pApp is a created and connected ICanfieldSDKApp object
    // get an image list object
    IDispatchPtr pDispImageList = pApp->GetImageList();
    if (pDispImageList)
    {
        CComQIPtr<ICanfieldSDKImageList,
                    &IID_ICanfieldSDKImageList>pImageList(pDispImageList);
        if (pImageList)
        {
            // get image to assign data to.  Note that calling
            // Load on image list
            // with a patient filter avoids a full load of image list
            pImageList->Load(varAppKey, _variant_t(false), varPatientID);
            IDispatchPtr pDispImage = pImageList->GetAt(varAppKey, varID);
            if (pDispImage)
            {
                CComQIPtr<ICanfieldSDKImage>pImage(pDispImage);
                if (pImage)
                {
                    // get image data assignment object
                    IDispatchPtr pDispDataImage = pImage->GetAttributeDataImage();
                    if (pDispDataImage)
                    {
                        CComQIPtr<IAttributeDataImage>pDataImage(pDispDataImage);
                        if (pDataImage)
                        {
// bstrID contains the Attribute ID to assign,
// varValue contains either a BSTR or a SAFEARRAY
// of 2 BSTRS
                            pDataImage->AddAttribute(bstrID, varValue);
                        }
                    }
                }
```

```
            }
        }
    }
}
catch (_com_error &e)
{
    AfxMessageBox(e.ErrorMessage(), MB_OK | MB_ICONSTOP);
}
```

2.4.6.2 Removing an Attribute Value From an Image

```
try
{
    CComPtr<ICanfieldSDKImageList>pImageList;
    HRESULT hRes = pImageList.CoCreateInstance(CLSID_CanfieldSDKImageList);
    if (SUCCEEDED(hRes))
    {
        CString strAppKey = _T("app key here");
        _variant_t varAppKey((LPCTSTR)strAppKey);

        pImageList->Load(   varAppKey,
                            _variant_t(false),
                            varPatientID);

        // get an attribute manager from the image list to get
        // the Attribute ID from the name
        // retrieve the attribute by name.  Note that it is more
        // desirable to work with
        // attributes by ID and using pMgr->GetAt() instead of
        // FindName to retrieve
        // a library object.
        IDispatchPtr pDispMgr = pImageList->GetAttributeMgr();
        CComQIPtr<IAttributeLibMgr>pMgr(pDispMgr);
        if (pMgr)
        {
            CComBSTR bstrGUID;
            IDispatchPtr pDispLib = pMgr->FindName(bstrAttrName, &bstrGUID);
            CComQIPtr<IAttributeLib>pLib(pDispLib);
            if (pLib)
            {
                _bstr_t bstrAttributeID = pLib->ID;
                IDispatchPtr pDispImage = pImageList->GetAt(varAppKey,
                                                            varImageID);
                CComQIPtr<ICanfieldSDKImage>pImage(pDispImage);
                if (pImage)
                {
                    // get a data manager so we can remove
                    // the attribute value from the image
                    IDispatchPtr pDispDataImage = pImage->GetAttributeDataImage();
                    if (pDispDataImage)
                    {
                        CComQIPtr<IAttributeDataImage>pDataImage(pDispDataImage);
                        if (pDataImage)
                        {
                            // get the value object to delete
                            IDispatchPtr pDispDataItem = pDataImage->FindAttribute(
                                                            bstrAttributeID,
                                                            varValue);

                            CComQIPtr<IAttributeDataItem>pDataItem(pDispDataItem);
                            if (pDataItem)
                            {
                                pDataItem->Remove();
                            }
                        }
                    }
```

```
                }
            }
        }
    }
}
catch (_com_error &e)
{
    AfxMessageBox(e.ErrorMessage());
}
```

2.4.7 Reading Image Attribute Library Information

```
try
{
    // note that pApp is an ICanfieldSDKApp object
(see Opening Application Connections above)
    IDispatchPtr pDispImageList = pApp->GetImageList();
    if (pDispImageList)
    {
        CComQIPtr<ICanfieldSDKImageList>pImageList(pDispImageList);
        if (pImageList)
        {
            // retrieve Attribute Library Manager
            IDispatchPtr pDispLibMgr = pImageList->GetAttributeMgr();
            if (pDispLibMgr)
            {
                CComQIPtr<IAttributeLibMgr>pLibMgr(pDispLibMgr);
                if (pLibMgr)
                {
                    // get total library count
                    long lCount = pLibMgr->Count;
                    CString strEntry;
                    for (long lIndex = 0L; lIndex < lCount; lIndex++)
                    {
                        _variant_t varIndex(lIndex);
                        IDispatchPtr pDispLib = pLibMgr->GetAt(varIndex);
                        if (pDispLib)
                        {
                            CComQIPtr<IAttributeLib>pLib(pDispLib);
                            if (pLib)
                            {
                                _bstr_t bstrName = pLib->Name;
                                _bstr_t bstrID = pLib->ID;
                                short nDataType = pLib->DataType;
                                // don't add separators to list
                                if (IAttributeLib_DATATYPE_SEPARATOR != nDataType)
                                {
                                    // format library entry for adding to a list
                                    strEntry.Format(_T("%s - %s"),
                                                    (LPCTSTR)bstrName,
                                                        (LPCTSTR)bstrID);
                                }
                            }
                        }
                    } // for
                    // do this if we are not going to use the manager anymore
                    pLibMgr->Shutdown();
                }
            }
    }
}
catch (_com_error &e)
{
    AfxMessageBox(e.ErrorMessage(), MB_OK | MB_ICONSTOP);
```

```
}
```

2.4.8 Retrieving SDK Version Information

```
// note that an application object connection is not required before retrieving these properties
CComPtr<ICanfieldSDK>pSDK;
hRes = pSDK.CoCreateInstance(CLSID_CanfieldSDK);
if (SUCCEEDED(hRes))
{
    _bstr_t bstrSDKVersion = pSDK->SDKVersion;
    long lSDKVersionMajor = pSDK->SDKVersionMajor;
    long lSDKVersionMinor = pSDK->SDKVersionMinor;
}
```

2.4.9 Database List Management

2.4.9.1 Iterating the database list

This example iterates the entire database list and finds the index of the active database.

```
// It is assumed that this object has been created and AppID set somewhere
// else (see section 2.2.1)
CComPtr<ICanfieldSDKApp>pApp;

// load database list and get first object
long lCount = 0;
try
{
    // make sure we are connected
    pApp->Open();

    // this initiates a list load
    pApp->get_DatabaseCount(&lCount);

    CString strActiveDB(_T(""));
    // get default database so we can select it in combo
    CComPtr<IDispatch>pDispActiveDatabase = pApp->GetDatabase(_variant_t());
    CComQIPtr<ICanfieldSDKDatabase>pDatabase(pDispActiveDatabase);
    if (pDatabase)
    {
        CComBSTR bstr;
        pDatabase->get_Name(&bstr);
        strActiveDB = CString(bstr);
    }

    long lSelIndex = -1;
    for (long lIndex = 0L; lIndex < lCount; lIndex++)
    {
        CComPtr<IDispatch>pDispDB = pApp->GetDatabase(_variant_t(lIndex));
        if (pDispDB)
        {
            CComQIPtr<ICanfieldSDKDatabase>pDatabase(pDispDB);
            if (pDatabase)
            {
                CComBSTR bstr;
                pDatabase->get_Name(&bstr);
                CString strName = CString(bstr);
                if (!strActiveDB.IsEmpty() &&
                    0 == strActiveDB.CompareNoCase(strName))
                {
                    // do something with index or name for current selection
                    lSelIndex = lIndex;
                }
            }
        }
```

```
    }
}
catch (_com_error &e)
{
    MessageBox(e.ErrorMessage(), _T("OnLoadDbList"), MB_OK | MB_ICONSTOP);
}
```

2.4.9.2 Setting the Active Database

This example selects the active database by name.

```
// It is assumed that this object has been created and AppID set somewhere
// else (see section 2.2.1)
CComPtr<ICanfieldSDKApp>pApp;

// it is assumed that we are selecting based on the name.
// something like _variant_t varIndex(lIndex); could be used
// to select by index.
// See 2.3.10.1 for how to iterate the database list
CString strName; // set this up elsewhere
_variant_t varName(_bstr_t((LPCTSTR)strName));
pApp->SetActiveDatabase(varName);
```

2.5 Translating C++ Code Samples to VB 6.0

COM Object creation

C++ Code	VB Code

`CComPtr<ICanfieldSDKApp>pSDK;`
`HRESULT hRes =`
`pSDK.CoCreateInstance(CLSID_CanfieldSDKApp);`
	`Dim objSDK as Object`
	`Set objApp = CreateObject("CanfieldSDK.CanfieldSDKApp"`

COM Object method calls

C++ Code	VB Code

`CComPtr<ICanfieldSDKApp>pApp;`
`// create object here`
`pApp->Open();`
	`Dim objApp as Object`
	`' create object here`
	`objApp.Open`

Setting / Getting COM Object properties

C++ Code	VB Code

`CComPtr<ICanfieldSDKApp>pApp;`
`// get AppID property`
`long lAppID = pApp->AppID;`
	`Dim objApp as Object`
	`' create object here`
	`' get AppID property`
	`Dim lAppID as Long`
	`lAppID = objApp.AppID`

String variables

C++ Code	VB Code
`_bstr_t bstrVersion = pSDK->SDKVersion;`	`Dim strVersion as String` `strVersion = objSDK.SDKVersion`

VARIANT variables

C++ Code	VB Code
`_variant_t varVersion;`	`Dim varVersion`

Exception (Error) Handling

C++ Code	VB Code

`try`
`{`
`// call COM methods`
`// set/get COM properties`
`}`
`catch (_com_error &e)`
`{`
`// do something with error`
`}`
	`On Error goto ErrHandler`
	
	`' do something that can`
	`' generate a COM error`
	`Exit Function`
	
	`ErrHandler:`
	`strMsg = "Error: " & Str(Err.Number) & " - " & Err.Des`
	`MsgBox strMsg`
	`Resume Next`

2.6 Menu Installation / Removal

```
Dim objApp As Object
Dim strFileName As String
Dim strText As String

Dim varDirectory
Dim varParameters
Set objApp = CreateObject("CanfieldSDK.CanfieldSDKApp")
objApp.AppID = 1
strText = "Notepad AddMenuItem Test"
strFileName = "notepad.exe"
varDirectory = "C:\windows"
varParameters = "test.txt"
objApp.AddMenuItem strText, strFileName, varDirectory, varParameters
```

2.6.1 Menu Parameter Data Tags

The following tag values are supported within the

Tag Value	Tag Description
"%__ACTIVEPATIENT__%"	ID of active patient
"%__IMAGESELIDLIST__%"	list of selected image IDs separated by commas
"%__IMAGESELCOUNT__%"	count of selected images
"%__VISIAOVERLAY__%"	Visia overlay settings
"%__DBACTIVENAME__%"	name of active DB
"%__DBACTIVEDSN__%"	DSN of active database
"%__DBDEFAULTDSN__%"	DSN of default DB
"%__DBDEFAULTNAME__%"	name of default DB

3. INTERFACE PROPERTIES AND METHODS BY OBJECT

3.1 Application Object (CanfieldSDK.CanfieldSDKApp)

The application object manages communication with the installed server application(s) and retrieves other objects and associated information to facilitate data manipulation.

IMPORTANT: You must call both the Initialize() and Open() methods on a CanfieldSDKApp object before you can access any SDK functionality.

3.1.1 Methods

3.1.1.1 Initialize(
VARIANT varAppKey,
VARIANT varAppName,
VARIANT varAppVendor,
VARIANT varAppVersion)

Register with the server application to enable Canfield SDK integrator's licensed features. The registered information will be applied to all images stored through the SDK.

VARIANT varAppKey - Integrator's unique application key, obtained from Canfield.

VARIANT varAppName – Integrator's application name, used in integrated application messaging. This is NOT the name of the Canfield application.

VARIANT varAppVendor – Application vendor info for the Integrator's application

VARIANT varAppVersion – Integrator's application version string. This is NOT the version of the Canfield application.

3.1.1.2 Open()
Initiate communications with the selected server application (based on value of AppID property). Creates a new server process if necessary.

3.1.1.3 Close()
Shutdown communications with and close the server application.
Note that the CanfieldSDKApp object destructor will automatically close a running server application if the SDK created the process as a result of an SDK object method call or property change.

NOTE: Although this method accepts an optional varAlwaysClose argument, this argument is ignored in the current implementation.

3.1.1.4 Login(
BSTR bstrUID,
BSTR bstrPWD)
Login to the server application.

NOTE: In general, integrators do not need to login to the server application. If the Canfield system is set to require logins, then on the integrator's call to the Open() method the Canfield SDK will present a login dialog to the user. If you have access to a valid Username and Password (the Canfield system uses Windows NT authentication) you can pass these credentials to the Login() method prior to calling the Open() method in order to bypass the Canfield login dialog.

3.1.1.5 LoginUI(
VARIANT_BOOL bDoLogin,
BSTR* pbstrUID,
BSTR* pbstrPWD,
VARIANT_BOOL* pbLoggedIn,
LONG* plUserAction)
Authenticate to server application and optionally login using this object's Login method

VARIANT_BOOL bDoLogin, - whether to call CanfieldSDKApp Login method on successful authentication

BSTR* pbstrUID (out) - typed in UID

BSTR* pbstrPWD (out) - typed in PWD

VARIANT_BOOL* pbLoggedIn - authentication successful?

LONG* plUserAction - login dialog disposition (IDOK, IDCANCEL, etc.)

3.1.1.6 OpenChart(
BSTR bstrID,
VARIANT_BOOL bDoModal,
LPDISPATCH **ppDispPatient)
Open a chart to a particular patient and activate the user interface for the server application . If a patient is specified but not found an error is returned.

BSTR bstrID – patient ID (can be empty string) - if empty, first patient is active.

VARIANT_BOOL bDoModal - . If VARIANT_TRUE(-1), the method blocks until the server application is closed by the user.

LPDISPATCH *ppDispPatient - (out) – returned CanfieldSDKPatient Object IDispatch interface

3.1.1.7 Show(
VARIANT varShow)
Show/hide server application window
VARIANT varShow (optional – default is TRUE) – show/hide flag. Any non-zero value shows the server application window, zero hides.

3.1.1.8 CreatePatient(
VARIANT varLName,
VARIANT varFName,
VARIANT varMI,
LPDISPATCH *ppPatient)
Create a new patient and save initial data to database.

VARIANT varLName – last name - required

VARIANT varFName – first name - optional

VARIANT varMI – middle initial - optional

LPDISPATCH *ppDispPatient - (out) – returned Patient Object IDispatch interface. ID property is set with newly assigned patient ID.

NOTE: Typically the ID value on the returned patient object is stored with the corresponding patient record in the Integrator's system. This value serves as the foreign key into the Canfield system.

3.1.1.9 GetAppInfo(
long lAppID,
VARIANT *pvarName,
VARIANT *pvarPath,
VARIANT_BOOL *pbInstalled)
returns (VARIANT_TRUE/VARIANT_FALSE) based on whether the requested server application can be located on the system based on available server application configuration information. The server application is not guaranteed to successfully launch because the server application configuration information may be invalid.

LONG lAppID – valid application ID, see AppID property on this object.
VARIANT *pvarName – optional application name to be returned, same as Name property on this object below.
VARIANT *pvarName – optional application path to be returned, same as AppPath property on this object below.

3.1.1.10 GetDatabase(
VARIANT varNameOrIndex,
LPDISPATCH *ppDispDatabase)
retrieve a database object by name or index. (DatabaseCount property returns the total number of application databases.)

VARIANT varNameOrIndex – VT_I4 index or VT_BSTR database name. Specify a VT_EMPTY VARIANT to get current active database.

LPDISPATCH *ppDispDatabase (out) – returned CanfieldSDKDatabase Object IDispatch interface or NULL on error.

3.1.1.11 GetPatientList(
LPDISPATCH *ppDispPatientList)
create and return a Patient List Object

LPDISPATCH *ppDispPatientList (out) – returned CanfieldSDKPatientList Object IDispatch interface

3.1.1.12 GetEventManager(
LPDISPATCH *ppDispEventMgr)
create a new Event Manager object for managing event notifications

LPDISPATCH *ppEventMgr (out) – returned CanfieldSDKEventMgr Object IDispatch interface.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.1.1.13 RegisterEventSink()
NOTE: This method has not been implemented. Canfield does not support the use of this method.

3.1.1.14 GetImageList(
LPDISPATCH *ppDispImageList)
create and return a new CanfieldSDKImageList object

LPDISPATCH *ppDispImageList (out) – returned CanfieldSDKImageList Object IDispatch interface.

3.1.1.15 IsOpen(
VARIANT_BOOL *pbOpen)
Returns whether the server application is connected.

VARIANT_BOOL *pbOpen set to VARIANT_TRUE if connected, VARIANT_FALSE if not.

3.1.1.16 WaitForInputIdle(
VARIANT varDelay)
Delays until connected application is idle – used to handle activation during initial application launch.

VARIANT varDelay – VT_I4 delay in milliseconds. Default value if not specified is 1500.

3.1.1.17 SetActiveDatabase(
VARIANT varNameOrIndex,
LPDISPATCH *ppDispDatabase)
Activate and open a database by name or index.

VARIANT varNameOrIndex – operates the same as in GetDatabase above.

LPDISPATCH *ppDispDatabase(out) – returned CanfieldSDKDatabase Object IDispatch interface or NULL on error.

NOTE: If a database is not explicitly specified via SetActiveDatabase(), the user's default start-up database for the Mirror application will be used for all SDK activity.

3.1.1.18 AddMenuItem(
BSTR bstrText,
BSTR bstrFileName,
VARIANT varDirectory,
VARIANT varParameters,
VARIANT varBitmap,
VARIANT varCreationFlags,
VARIANT varAppKey,
VARIANT varShowFlags,
VARIANT varCanRunFlags)
Add an "Extras" menu item to the server application
BSTR bstrText – menu text (also used to uniquely identify the Extra item)

BSTR bstrFileName – name of executable module

VARIANT varDirectory – starting folder (optional)

VARIANT varParameters – command line parameters (optional). The following parameter replacement tags are defined:

Tag String	Description
%__ACTIVEPATIENT__%	ID of current patient
%__IMAGESELIDLIST__%	list of image Ids for selected images, separated with commas
%__IMAGESELCOUNT__%	count of selected images
%__VISIAOVERLAY__%	Visia overlay settings (Visia images only)
%__DBACTIVENAME__%	name of active DB
%__DBACTIVEDSN__%	DSN of active database
%__DBDEFAULTDSN__%	DSN of default DB
%__DBDEFAULTNAME__%	name of default DB

VARIANT varBitmap – optional toolbar bitmap(s). Up to three BITMAP resource Ids can be specified which correspond to small, medium, and large toolbar formats in the server application. Any loadable module is supported (EXE or DLL). Bitmaps will be scaled as necessary.
Data format: resource filename:integer bitmap resource ID small[:res ID medium][:res ID large]
Example: "Extra1.exe:101:102:103"

VARIANT varCreationFlags – optional process creation flags, not currently used.

VARIANT varAppKey – Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varShowFlags – optional created process window show flags, value is passed through unchanged to ShellExecute. Corresponds to SW_XXX value listed in table below. Default is SW_SHOWNORMAL (0x01).

SW_HIDE	Hides the window and activates another window.
SW_MAXIMIZE	Maximizes the specified window.
SW_MINIMIZE	Minimizes the specified window and activates the next top-level window in the z-order.
SW_RESTORE	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.
SW_SHOW	Activates the window and displays it in its current size and position.
SW_SHOWDEFAULT	Sets the show state based on the SW_ flag specified in the STARTUPINFO structure passed to the CreateProcess Windows API function by the program that started the application. An application should call ShowWindow with this flag to set the initial show state of its main window.
SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates the window and displays it as a minimized window.
SW_SHOWMINNOACTIVE	Displays the window as a minimized window. The active window remains active.
SW_SHOWNA	Displays the window in its current state. The active window remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The active window remains active.
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when displaying the window for the first time.

VARIANT varCanRunFlags – optional flags which control whether the menu item is enabled. Default is CanfieldSDK_CANRUNFLAG_ALL.
The following values are defined:

Flag	Value	Description
CanfieldSDK_CANRUNFLAG_ALL	0x0000	always can run
CanfieldSDK_CANRUNFLAG_SINGLE	0x0001	only one image may be selected
CanfieldSDK_CANRUNFLAG_MULTIPLE	0x0002	needs multiple image selections
CanfieldSDK_CANRUNFLAG_DUAL	0x0004	needs exactly 2 image selections
CanfieldSDK_CANRUNFLAG_ANY	0x0008	one or more images can be selected

3.1.1.19 RemoveMenuItem()

NOTE: This method has not been implemented. Canfield does not support the use of this method.

3.1.1.20 LoadSinglePatient(
BSTR bstrID,
LPDISPATCH *ppDispDatabase);
Load a single patient based on CanfieldSDKPatient.ID property

BSTR bstrID – requested Patient ID

LPDISPATCH *ppDispPatient (out) – returned CanfieldSDKPatient Object IDispatch interface or NULL on error.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method. Use CanfieldSDKPatientList.GetAt() instead.

3.1.1.21 SelectPatient()
NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.1.1.22 SelectImages(
VARIANT varImageIDs,
VARIANT varClear,
VARIANT varPatientID,
LONG *plSelCount)
Select one or more images in the active or specified patient chart.

VARIANT varImageIDs – BSTR SAFEARRAY or single BSTR list of image Ids to select. Id values are retrieved through the ID property of the CanfieldSDKImage object.

VARIANT varClear, - optional flag indicating whether to clear existing selections, VARIANT_TRUE to clear. Default is to clear selections.

VARIANT varPatientID, - optional patient ID. If not specified, current active patient is used. If specified the specified patient becomes the active patient.

LONG *plSelCount (out) – returns the count of successfully selected images.

3.1.1.23 OpenImages()
NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.1.1.24 OpenSelectedImages()
NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.1.1.25 GetCaptureObject(
LPDISPATCH *ppDispCapture)
create and return a new CanfieldSDKCapture object

LPDISPATCH *ppDispCapture (out) – returned CanfieldSDKCapture Object IDispatch interface.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.1.2 Properties

AppID – application identifier
LONG AppID
possible values:
ICanfieldSDK_AppID_MirrorSuite = 1

Only Canfield server applications which are installed on the system can be accessed via automation. **Note that if the AppID property is changed for a connected server application the application will be closed whether the SDK launched it or not. This is the same behavior as when calling this object's Close() method with varAlwaysClose set to VARIANT_TRUE.

Name - Name of server Application (read-only)
BSTR Name

AppPath - Location of server application (read-only)
BSTR AppPath

DatabaseCount – number of registered databases (read-only)
long DatabaseCount

Version – connected application version as tagged value string (read-only), formatted as: VERSION|n.n BUILDDATE|CCYYMMDD.
BSTR Version

ValueElementSeparator – character used to separate elements in tagged value strings such as CanfieldSDKApp::Version property (read-only).

ValueValueSeparator – character used to separate name and value in tagged value strings such as CanfieldSDKApp::Version property (read-only).

VersionMajor – connected application major version as numeric (read-only)
long VersionMajor

VersionMinor – connected application minor version as numeric (read-only)
long VersionMinor

Visible – VARIANT_TRUE if application window is visible (WS_VISIBLE) and valid. (read-only).
VARIANT_BOOL Visible

Hwnd – window handle of connected application (read-only)
OLE_HANDLE Hwnd

OpenTimeout – timeout value in sec. for waiting for server application launch in Open method
LONG lOpenTimeout – minimum allowable value == 90 (sec); values < 90 will set timeout to 90.

3.2 Database Object (CanfieldSDK.CanfieldSDKDatabase)

The Database object represents a single application database. This object is returned in the CanfieldSDKApp object's GetDatabase and SetActiveDatabase methods. Its primary purpose is to facilitate programmable selection of the active application database when there are multiple databases installed.

3.2.1 Methods

NONE

3.2.2 Properties

Name – descriptive name for database created by user in Canfield Database Options (read-only).
BSTR Name

ImagePath – root location of image repository (read-only).
BSTR ImagePath

ThumbnailPath – root location of thumbnail image repository (read-only).
BSTR ThumbnailPath

DSN – associated ODBC datasource name (read-only).
BSTR DSN

IsShared – whether database is shared from the local system (read-only).
VARIANT_BOOL IsShared

IsRemote – whether database has been published as shared from another system (read-only).
VARIANT_BOOL IsRemote

IsPrimary – whether the database is a main application database. Currently this is the only supported type (read-only).
VARIANT_BOOL IsPrimary

3.3 Patient Object (CanfieldSDK.CanfieldSDKPatient)

The Patient object represents a patient in the system. The object permits retrieval and modification of the database information and retrieval of image identifiers associated with a single patient.

3.3.1 Methods

3.3.1.1 Load()

Load patient data from storage

3.3.1.2 LoadThumbnail(
VARIANT varID,
LPDISPATCH *ppThumbnail)

Load a thumbnail image by ID or index

VARIANT varID – image ID to load (VT_EMPTY uses current ID Image). Alternately a 0-based index could be used, valid values 0 – (ThumbnailCount – 1). If patient has no thumbnails, result will be E_INVALIDARG for either index or BSTR value for varID.

LPDISPATCH *ppThumbnail (out) – returned CanfieldSDKThumbnail Object IDispatch interface or NULL on error or not found.

3.3.1.3 Save()

Save current data to permanent storage

3.3.2 Properties

ID - Unique patient identifier
BSTR ID

SSN - Social Security Number
BSTR SSN

AltID - Alternate Patient ID or Medical Record Number (MRN)
BSTR AltID

IDThumbnail - Image ID for current patient ID photo (the image the user has selected to represent the patient when displayed in a Mirror patient list)
BSTR IDThumbnail

LastName - Last Name
BSTR LastName

FirstName - First Name
BSTR FirstName

MI - Middle Initial
BSTR MI

DOB - Date of Birth
DATE DOB – as OLE Automation DATE (COleDateTime compatible)

Gender - Gender
BSTR Gender

Race - race
BSTR Race

Address1 - Address1
BSTR Address1

Address2 - Address2
BSTR Address2

City - City
BSTR City

State – abbreviated state
BSTR State

PostalCode – PostalCode (or ZIP)
BSTR PostalCode

Phone – primary Phone
BSTR Phone

Phone2 – secondary phone
BSTR Phone2

ThumbnailCount – count of total thumbnails for patient
long ThumbnailCount

DateCreated – patient record creation date / time
DATE DateCreated

DateModified – patient demographics modified date / time (may be NULL date if not modified after creation)
DATE DateModified

DateCreatedImage – newest creation date / time (may be NULL date if no images) of images on file for this patient
DATE DateCreatedLastImage

DateModifiedImage – newest modified date / time (may be NULL date if no images have been modified after adding to patient) of images on file for this patient
DATE DateModified

DateModifiedLast – newest modified date / time for any possible type of modification: edit patient demographics, add new image to patient, or edit image data. Never NULL date because at least DateCreated will always be non-NULL.
DATE DateModifiedLast

3.4 Patient List Object (CanfieldSDK.CanfieldSDKPatientList)

The Patient List object is a "manager" object for counting and iterating the list of all patients in the active database.

3.4.1 Methods

3.4.1.1 Load(
VARIANT varForceReloadParm,
VARIANT varLName,
VARIANT varFName,
VARIANT varMI,
VARIANT varSSN,
VARIANT varAltID)
Load a patient list from storage with optional filter criteria.

VARIANT varForceReload - if TRUE, ignores the cache age and reloads entire list. If FALSE, only reloads if patient list object cache is expired.
VARIANT varLName – optional "AND" filter criteria on LastName patient property
VARIANT varFName – optional "AND" filter criteria on FirstName patient property
VARIANT varMI – optional "AND" filter criteria on MI patient property
VARIANT varSSN – optional "AND" filter criteria on SSN patient property
VARIANT varAltID – optional "AND" filter criteria on AltID patient property

3.4.1.2 GetAt(
VARIANT varIdOrIndex,
LPDISPATCH *ppDispPatient)
retrieve a particular patient by 0-based index or ID

VARIANT varIdOrIndex - 0 - based numeric index or patient ID string, valid string or range 0 to Count – 1 for numeric value

LPDISPATCH *ppDispPatient (out) - returned CanfieldSDKPatient Object IDispatch interface

3.4.1.3 LoadSinglePatient(
BSTR bstrID,
LPDISPATCH *ppDispDatabase);
Load a single patient based on CanfieldSDKPatient.ID property.

BSTR bstrID – requested Patient ID

LPDISPATCH *ppDispPatient (out) – returned CanfieldSDKPatient Object IDispatch interface or NULL on error.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method. Use CanfieldSDKPatientList.GetAt() instead.

3.4.2 Properties

Count – total number of patients in the active database (read-only). Retrieval of this property will initiate a load if the list needs to be loaded.
LONG Count

MaxCacheAge – time in milliseconds after Load method is called before list is considered "old". Defaults to 2 minutes.
LONG MaxCacheAge

Application – associated CCanfieldSDKApplication object if Patient List object retrieved via CanfieldSDKApplication.GetPatientList method. Note that this is an AddRef'd object that must be Released by the caller.
LPDISPATCH Application

3.5 Thumbnail Object (CanfieldSDK.CanfieldSDKThumbnail)

The thumbnail object is returned through a call to the LoadThumbnail method on the Patient object. The object implements image thumbnail data retrieval and conversion in Windows Bitmap and IPicture formats.

3.5.1 Methods

3.5.1.1 GetBitmap(
VARIANT varUsePalette,
VARIANT varGrayScale,
OLE_HANDLE *phBitmap)
Return thumbnail data in Windows BITMAP format

VARIANT varUsePalette(optional – default FALSE) – if TRUE, image is returned as 256 color palletized image, FALSE for 24-bit non-palletized image

VARIANT varGrayScale (optional – default FALSE) – if TRUE, image is returned as grayscale, FALSE for color

OLE_HANDLE *phBitmap (out) – Windows HBITMAP. Note that caller is expected to free this object with DeleteObject.

3.5.1.2 GetPicture(
VARIANT varUsePalette,
VARIANT varGrayScale,
IPictureDisp **ppPicture)
retrieve thumbnail data in IPictureDisp format (this is the format used by the VB 6.0 picture control)

varUsePalette and varGrayScale parameters are the same as GetBitmap method

IPictureDisp **ppPicture (out) – returned IPictureDisp object

Note for .NET developers: The returned IPictureDisp object can be converted to a System.Drawing.Image object using Microsoft.VisualBasic.Compatibility.VB6.Support.IPictureDispToImage

3.5.2 Properties

ID - Unique Thumbnail Identifier
BSTR ID

Height – image height in pixels (read-only)
LONG Height

Width – image width in pixels (read-only)
LONG Width

3.6 Image List Object (CanfieldSDK.CanfieldSDKImageList)

The Image List object is a "manager" object for counting and iterating lists of full images in the active database which have been stored through the Canfield SDK.

3.6.1 Methods

3.6.1.1 Load(VARIANT varAppKey,VARIANT varForceReloadParm,VARIANT varSubjectID,VARIANT varLName,VARIANT varFName,VARIANT varMI,VARIANT varSSN,VARIANT varAltID)

load a list of Image objects stored with the Canfield SDK.

VARIANT varAppKey – Canfield SDK integrator's unique application key, obtained via software license from Canfield.
VARIANT varForceReload - if TRUE, ignores the cache age and reloads entire list. If FALSE, only reloads if patient list object cache is expired.
VARIANT varSubjectID – optional "AND" filter criteria on ID patient property
VARIANT varLName – optional "AND" filter criteria on LastName patient property
VARIANT varFName – optional "AND" filter criteria on FirstName patient property
VARIANT varMI – optional "AND" filter criteria on MI patient property
VARIANT varSSN – optional "AND" filter criteria on SSN patient property
VARIANT varAltID – optional "AND" filter criteria on AltID patient property

Note that previous filter criteria is remembered if the list is reloaded after cache expiration. The criteria will be modified only by changing the parameters and calling the Load method.

3.6.1.2 GetAt(VARIANT varAppKey,VARIANT varIdOrIndex,LPDISPATCH *ppDispImage)

retrieve a CanfieldSDKImage object stored with the Canfield SDK by Id or 0-based numeric list index of loaded list.

VARIANT varIdOrIndex – image ID or index into loaded ImageList object.

VARIANT varAppKey – Canfield SDK integrator's unique unlock key, obtained via software license from Canfield.

LPDISPATCH *ppDispImage – returned CanfieldSDKImage object or NULL on error or not found. Caller is expected to Release the object

3.6.1.3 PushImage(VARIANT varAppKey,BSTR bstrFileName,VARIANT varPatientID, VARIANT varCompressionType, VARIANT varQuality, LPDISPATCH *ppDispImage)

create a new Image object and store data from the specified file (bstrFileName).

VARIANT varAppKey **-** Canfield SDK integrator's unique application key, obtained via software license from Canfield.

BSTR bstrFileName - full path to image file to store.

VARIANT varPatientID – patient ID with which to associate the new image.

VARIANT varCompressionType – optional compression flags. Valid values are:
ICanfieldSDK_Compress_JPG – JPEG with associated quality factor passed as varQuality (default if not specified)
ICanfieldSDK_Compress_Lossless – lossless compression
ICanfieldSDK_Compress_None – no compression (not recommended because of extra load/save time).

VARIANT varQuality – optional compression quality factor. Valid values 1-64. Used only for varCompressionType == ICanfieldSDK_Compress_JPG. Default is 1=best if not specified.

LPDISPATCH *ppDispImage (out) – newly created Image object with its unique ID set.

3.6.1.4 GetAttributeMgr(LPDISPATCH *ppDispAttributeMgr)

return an initialized Attribute Manager object for reading and writing Attribute Library data (see section

LPDISPATCH *ppDispAttributeMgr (out) – newly created Attribute Manager object.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.6.1.5 LoadCaptureSession(VARIANT varAppKey, BSTR bstrCaptureSessionID)

load the image list with the images for a given capture session

VARIANT varAppKey – Canfield SDK integrator's unique unlock key, obtained via software license from Canfield.

BSTR bstrCaptureSessionID – Session identifier provided by capture object

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.6.1.6 GetByType(VARIANT varAppKey,VARIANT varImageType, LPDISPATCH *ppDispImage)

retrieve an Image object stored with the Canfield SDK by the type of image captured.

VARIANT varAppKey – Canfield SDK integrator's unique unlock key, obtained via software license from Canfield.

VARIANT varImageType – a long value from the CaptureSequenceSteps enumeration

LPDISPATCH *ppDispImage – returned Image object or NULL on error or not found. Caller is expected to Release the object

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.6.1.7 GetImageAttributes(VARIANT varAppKey, LPDISPATCH *ppDispAttributes)

returns a CanfieldSDKImageAttributes object for managing Attribute Library data associated with the images in the image list. See section 3.8 for CanfieldSDKImageAttributes interface documentation.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

LPDISPATCH *ppDispAttributes (out) – newly created and initialized CanfieldSDKImageAttributes object.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.6.2 Properties

Count – count of SDK images in the list. Retrieval of this property will initiate a load if the cache age has expired.

LONG Count

CaptureSessionID – ID of associated capture session, only set if image list is loaded with LoadCaptureSession
BSTR bstrCaptureSessionID

3.7 Image Object (CanfieldSDK.CanfieldSDKImage)

The image object implements manipulation of a full size image that has been stored through the Canfield SDK.

3.7.1 Methods

3.7.1.1 Load(VARIANT varAppKey,VARIANT varForceReload,VARIANT varLoadImage)

reload the Image attribute data optionally bitmap data from storage

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varForceReload (optional) – set to VARIANT_TRUE to force reload from permanent storage without checking for cache expiration (default = VARIANT_FALSE).

VARIANT varLoadImage (optional) – set to VARIANT_TRUE to load image bitmap data, VARIANT_FALSE to load Image attributes only (default == VARIANT_FALSE)

3.7.1.2 SetImageBLOB(VARIANT varAppKey,VARIANT varBLOBName,VARIANT varData)

set an Image data BLOB by name.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varBLOBName – Unique user-defined name for the BLOB (required).

VARIANT varData – SAFEARRAY of bytes for BLOB data (VT_I1|VT_ARRAY or VT_UI1|VT_ARRAY)

3.7.1.3 SetImageBLOBFromFile(VARIANT varAppKey,VARIANT varBLOBName,BSTR bstrFileName)

set an Image data BLOB by ID or index.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varBLOBName – Unique user-defined name for the BLOB, no empty BLOB names are permitted.

BSTR bstrFileName – full path to external file containing BLOB data

3.7.1.4 GetImageBLOB(VARIANT varAppKey,
VARIANT varBLOBName,
VARIANT varDestFile = VT_EMPTY,
VARIANT *pvarData = VT_EMPTY)

retrieve an Image data BLOB previously stored with SetImageBLOB by ID or index into a file or memory buffer.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varBLOBName – unique user-defined name for the BLOB (required).

VARIANT varDestFile (optional) – target file name for returning the image BLOB. File is created if necessary. If name is supplied no data is returned in pvarData below.

VARIANT *pvarData (optional) – SAFEARRAY of bytes of BLOB data (VT_I1|VT_ARRAY or VT_UI1|VT_ARRAY) allocated by SDK. Caller is expected to free the data.

3.7.1.5 DeleteImageBLOB(VARIANT varAppKey,VARIANT varIdOrIndex)

delete an Image data BLOB previously stored with SetImageBLOB.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

3.7.1.6 GetImageBLOBInfo(VARIANT varAppKey, VARIANT varIndex, VARIANT* pvarID, VARIANT *pvarIdOrIndex, VARIANT* pvarFilePath)

retrieve information about an Image BLOB based on 0-based index.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varIndex – 0-based index into the image's BLOB list. If CountImageBLOB property is > 0, valid range is 0 – (CountImageBLOB – 1).

VARIANT *pvarID – returns BLOB unique identifier.

VARIANT *pvarBLOBName – returns BLOB name value that can be used with GetImageBLOB to retrieve the BLOB.

VARIANT *pvarFilePath – returns stored file path in the repository. This name can be used to determine information about the BLOB based on its extension since the filename from which the BLOB was stored using SetImageBLOBFromFile() is retained at the end of the system stored BLOB FilePath. For example, if the BLOB file is set from the file "BLOB001.BLB" the system name would be something similar to "20040610190249495-BLOB001.blb".

3.7.1.7 GetImage(VARIANT varAppKey,VARIANT varFileName)

return image data stored by ImageList::PushImage or Image::SetImage. Data is written to specified file. File extension in varFileName determines the type of file that is written.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varFileName – (optional) full path to image file to create from image data. Supported file extensions (file types): BMP, JPG, TIF, PNG.

3.7.1.8 GetBitmap(VARIANT varAppKey, VARIANT varUsePalette,VARIANT varGrayScale,OLE_HANDLE *phBitmap)

Return associated image data in Windows BITMAP format

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varUsePalette(optional – default FALSE) – if TRUE, image is returned as 256 color palletized image, FALSE for 24-bit non-palletized image

VARIANT varGrayScale (optional – default FALSE) – if TRUE, image is returned as grayscale, FALSE for color

OLE_HANDLE *phBitmap (out) – Windows HBITMAP. Note that caller is expected to free this object with DeleteObject.

3.7.1.9 GetPicture(VARIANT varAppKey, VARIANT varUsePalette,VARIANT varGrayScale,IPictureDisp **ppPicture)

retrieve associated Image data in IPicture format (used VB 6.0 picture control)

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

varUsePalette and varGrayScale parameters are the same as GetBitmap method above.

IPictureDisp **ppPicture (out) – returned IPictureDisp object

Note for .NET developers: The returned IPictureDisp object can be converted to a System.Drawing.Image object using Microsoft.VisualBasic.Compatibility.VB6.Support.IPictureDispToImage

3.7.1.10 Delete(VARIANT varAppKey)

delete the image and all associated data from permanent storage.

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.7.1.11 GetAttributeDataImage(LPDISPATCH *ppDispAttributeDataImage)

returns an AttributeDataImage object for managing Attribute Library data associated with the image.

LPDISPATCH *ppDispAttributeDataImage (out) – newly created Attribute Data Image object.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.7.1.12 GetThumbnail(LPDISPATCH *ppDispThumbnail)

returns a CanfieldSDKThumbnail object from this image object.

LPDISPATCH *ppDispThumbnail (out) – newly created and initialized CanfieldSDKThumbnail object (IDispatch interface).

3.7.1.13 GetImageWithOverlay(VARIANT varAppKey, VARIANT varCropType, VARIANT varBitmapWidth, VARIANT varBitmapHeight, VARIANT varFeatureMask, VARIANT varBitmapFileName, VARIANT varCropX, VARIANT varCropY, VARIANT varCropWidth, VARIANT varCropHeight)

Creates a bitmap file of the image at a specific resolution with selected overlays added

VARIANT varAppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.

VARIANT varCropType – image cropping type value from Image Crop Type Reference.

VARIANT varBitmapWidth – Width in pixels of resultant scaled bitmap

VARIANT varBitmapHeight – Height in pixels of resultant scaled bitmap

VARIANT varBitmapFileName – full path for target FileName of the saved bitmap file

VARIANT varFeatureMask – Long value which is the bitwise OR of the desired values from the CaptureFeatureTypes enumeration

VARIANT varCropX – X coordinate in image coordinates of cropping rectangle.

VARIANT varCropY – Y coordinate in image coordinates of cropping rectangle.

VARIANT varCropWidth – Width of cropping rectangle in image coordinates or 0 for no cropping. -1 uses cropping rectangle at time of capture.

VARIANT varCropHeight – Height of cropping rectangle in image coordinates or 0 for no cropping

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.7.1.14 GetImageAttributes(LPDISPATCH *ppDispAttributes)

returns a CanfieldSDKImageAttributes object for reading Attribute Library data associated with the image. See section 3.8 for CanfieldSDKImageAttributes interface documentation.

LPDISPATCH *ppDispAttributes (out) – newly created and initialized CanfieldSDKImageAttributes object.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.7.1.15 GetTagProperty(LONG nTagID, VARIANT* pvarTagValue)

Retrieve an image tag property by property ID

LONG nTagID – tag identifier (see Image Tag Identifier Reference for list of valid values)

VARIANT *pvarTagValue – tag value. One-dimensional two element VARIANT SAFEARRAY. Element 0 == formatted tag value; element 1 == raw tag value.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.7.1.16 GetTagPropertyByIndex(LONG nIndex, LONG* pnTagID, VARIANT* pvarTagValue)

Retrieve a tag property by 0-based index value.

LONG nIndex – 0-based index value. Range == 0 – TagPropertyCount – 1 (see CanfieldSDKImage Properties for list of properties).
LONG* pnTagID – tag ID value (see Image Tag Identifier Reference for list of valid values)
VARIANT *pvarTagValue – tag value. One-dimensional two element VARIANT SAFEARRAY. Element 0 == formatted tag value; element 1 == raw tag value.

NOTE: This method has not been fully implemented. Canfield does not support the use of this method.

3.7.2 Properties – note that all of the following CanfieldSDKImage object properties are read-only

ID – unique image identifier
BSTR ID

PatientID – patient ID associated with the image
BSTR PatientID

DefaultQuality – quality factor used by Save method if not specified (read-only)
SHORT DefaultQuality

CountImageBLOB – number of image data BLOBs associated with the image object.
LONG CountImageBLOB

ProtectMask – data protection options – see Data Protection Mask Reference.
LONG ProtectMask

AppKey - Canfield SDK integrator's unique application key, obtained via software license from Canfield.
BSTR bstrAppKey

AppName – name of application that stored the image (see Application Initialize method)
BSTR bstrAppName

AppVendor – name of vendor that stored the image (see Application Initialize method)
BSTR bstrAppVendor

AppVersion – version of application that stored the image (see Application Initialize method)
BSTR bstrAppVersion

Read-Only image classification information
ImageType - T2K_IMAGE_TYPE image type enum value
LONG ImageType

ImageStatus - T2K_IMAGE_STATUS image status enum value
LONG ImageStatus

DateCreate – date that image record was created
DATE DateCreate

DateModified – date that image record has been modified
DATE DateModified

CaptureSessionID – Identifier for the session that the image belongs to
BSTR CaptureSessionID

SequenceType – if image captured in a sequence this property will contain a value from the Capture Sequence Type Reference.
LONG SequenceType

UserCreate – User who created the image record
BSTR UserCreate

UserModified – User who modified the image record (if there was a modification)
BSTR UserModified

DateImage – User-editable image date
DATE DateImage

PatientAge – Patient age in years based on DateImage property
LONG PatientAge

ImageNumber – optional user-specified display sort order
LONG ImageNumber

TagPropertyCount – total number of tag properties for the image
LONG TagPropertyCount

Notes – Notes associated with the image
BSTR Notes

3.8 Image Attributes Data Object (CanfieldSDK.CanfieldSDKImageAttributes)

** Note that use of the Image Attributes Data Object is not currently supported. **

The SDK Image Attributes Data collection object provides read-only access to Attribute Library values assigned to images.

3.8.1 Methods

3.8.1.1 GetAt(LONG lIndex, LPDISPATCH *ppDispImageAttribute)

Retrieve a single Image Attribute Data Object from Image Attributes collection using an index into the collection.

LONG lIndex – 0-based index into loaded image attributes list; valid index is between 0 and Count – 1.

LPDISPATCH *ppDispImageAttribute – returned CanfieldSDKImageAttribute object or NULL on error or not found. Caller is expected to call Release on the returned object.

3.8.2 Properties

Count – (read-only) count of assigned image attributes based on current filter. Retrieval of this property will initiate a load if the cache age has expired.

LONG Count

3.9 Image Attribute Data Object (CanfieldSDK.CanfieldSDKImageAttribute)

** Note that use of the Image Attribute Data Object is not currently supported. **

The SDK Image Attribute Data object provides read-only access to a single Attribute Library value assigned to an image.

3.9.1 Methods

3.9.1.1 GetValue(VARIANT *pvarValue)

returns a SAFEARRAY of one or two assigned Attribute Library values, depending on datatype.

VARIANT*pvarValue (out) – assigned value(s)

3.9.2 Properties

ID – image identifier associated with assigned value.
BSTR ID

PatientID – patient ID associated with the assigned value
BSTR PatientID

AttributeID – attribute library ID for assigned value
BSTR AttributeID

AttributeName – attribute library name for assigned value
BSTR AttributeName

3.10 SDK Object (CanfieldSDK.CanfieldSDK)

The SDK object provides services related to the integration SDK, not to any particular connected application.

3.10.1 Methods

No methods

3.10.2 Properties

SDKVersion – SDK version as tagged value string (read-only), formatted as:
VERSION|n.n|BUILDDATE|CCYYMMDD|BUILDCOMMENTS|szComments.
BSTR SDKVersion

SDKVersionMajor – SDK major version as numeric (read-only)
long SDKVersionMajor

SDKVersionMinor – minor version as numeric (read-only)
long SDKVersionMinor

3.11 Image Attributes Library Manager Object (IAttributeLibMgr)

Information available from Canfield upon request.

3.12 Image Attributes Library Object (IAttributeLib)

Information available from Canfield upon request.

3.13 Image Attributes Library Value Item Object (IAttributeLibItem)

Information available from Canfield upon request.

3.14 Event Manager Object (ICanfieldSDKEventMgr)

** Note that the Event Manager object is not implemented and may be included in a future SDK revision. **

Data change notifications are implemented to allow for systems on both sides of the interface to fire and subscribe to system events so data modifications and system interactions can be handled in real time as opposed to via periodic polling. Applications notify the SDK of changes via the FireEvent method. Subscribing applications retrieve an Event Manager object for configuring notification events via the CreateEventManager method of the Application Object. Event notifications are sent to the subscribing application by the SDK via WM_NOTIFY messages. The WM_NOTIFY message is structured as follows:
WPARAM: CanfieldSDK_NOTIFY_EVENT
LPARAM: ICanfieldSDK_EventData *pData
pData->hdr.hwndFrom == HWND of Canfield server application
pData->hdr.idFrom == 0
pData->hdr.code =

```
    NMHDR       hdr;              // see WM_NOTIFY documentation
    VARIANT  varParm01;     // event specific parameter 01
    VARIANT  varParm02;     // event specific parameter 02
    VARIANT  varParm03;     // event specific parameter 03
    VARIANT  varFlags;          // event flags – usage to be defined, always VT_EMPTY
} ICanfieldSDK_EventData *pICanfieldSDK_EventData;
```

The following data structure is taken from the Windows SDK and subject to change:

```
typedef struct tagNMHDR
{
    HWND     hwndFrom;    // Window handle to the control sending a message
    UINT_PTR idFrom;    // Identifier of the control sending a message
    UINT       code;       // Notification code (Canfield SDK Event ID)
} NMHDR;
typedef NMHDR FAR * LPNMHDR;
```

The Event Manager object provides a means to enumerate available events and for retrieving information about them through the GetEventInfo method and the EventCount property.

3.14.1 Methods

3.14.1.1 ConfigureEvent(VARIANT_BOOL bNotify,LONG lEventID,OLE_HANDLE hWnd)

configure the event manager for one or all events.

VARIANT_BOOL bNotify – VARIANT_TRUE to enable, VARIANT_FALSE to disable

LONG lEventID – Event ID of event to configure or ICanfieldSDK_EventAll to enable or disable all available events for this EventSink object

OLE_HANDLE hWnd – Window handle (HWND) to post the event WM_NOTIFY messages to.

3.14.1.2 GetEventInfo(VARIANT varIDOrIndex,LONG *plEventID,VARIANT *pvarEventName,VARIANT *pvarEventData)

Retrieve information about an event using either a 0-based index into the available event list or an Event ID value such as ICanfieldSDK_Event_DataChgPatient (see the Events Reference section 3.9.3 for a list of all defined Event ID values.

LONG *plEventID – returns the Canfield SDK EventID value.

VARIANT *pvarEventName (optional) – descriptive name

VARIANT *pvarEventData (optional) – additional event specific information (currently unused).

3.14.2 Properties

EventCount - total Number of Events that the SDK supports. Used in conjunction with GetEventInfo to iterate through available events (read-only).
LONG EventCount

3.15 Capture Object (CanfieldSDK.CanfieldSDKCapture)

** Note that use of the Capture Object is not currently supported. **

The Capture object is provides services to capture images using Canfield supported hardware. The capture object provides facilities to start and stop hardware, to manage capture sequence steps, and to drive the capture process. The capture object provides methods to facilitate custom user interfaces for the capture process. The capture object provides no user interface.

3.15.1 Methods

3.15.1.1 Initialize(VARIANT varCaptureType)

Setup the capture object for a particular type of capture (see Capture Type Reference) and initialize any required capture hardware.

VARIANT varCaptureType – value from Capture Type Reference

3.15.1.2 Uninitialize(VARIANT varCaptureType)

Shutdown capture hardware initialized with Initialize method and free any associated resources.

VARIANT varCaptureType – value from Capture Type Reference

3.15.1.3 StartSequence(VARIANT varhWndParent, VARIANT varX, VARIANT varY, VARIANT varWidth, VARIANT varHeight, VARIANT *pvarhWndDisplay)

Starts a capture sequence that has been established, first step normally creates a live preview. The window for the live preview is returned.
VARIANT varhWndParent – Handle to parent window for live preview window

VARIANT varX – X coordinate for preview window in parent window client coordinates

VARIANT varY – Y coordinate for preview window in parent window client coordinates

VARIANT varWidth – Width for live preview window

VARIANT varHeight – Height for live preview window

VARIANT *pvarhWndDisplay – Handle (HWND) to the live preview window (retval).

3.15.1.4 GetPrompt(BSTR *pbstrMsg1, BSTR *pbstrFileNameBmp1, BSTR *pbstrMsg2, BSTR *pbstrFileNameBmp2)

Returns the prompts for the user for the current step in the capture sequence

BSTR *pbstrMsg1 – Text for first prompt for the current step in the sequence

BSTR *pbstrFileNameBmp1 – Name of bitmap file to accompany the first prompt for the current step in the sequence

BSTR *pbstrMsg2 – Text for second prompt for the current step in the sequence

BSTR *pbstrFileNameBmp2 – Name of bitmap file to accompany the second prompt for the current step in the sequence

3.15.1.5 Next(VARIANT_BOOL *pbLast)

Moves to the next step in the capture sequence

VARIANT_BOOL *pbLast – Set to TRUE if moving to the last step in the sequence

3.15.1.6 Prev(VARIANT_BOOL *pbFirst)

Moves to the previous step in the capture sequence

VARIANT_BOOL *pbFirst – Set to TRUE if moving to the first step in the sequence

3.15.2 Properties

PatientID – associated patient ID
BSTR PatientID

Sequence – SAFEARRAY of sequence steps from CaptureSequenceSteps enumeration
VARIANT Sequence

Application – IDispatch CanfieldSDK Application object

3.15.3 Events

3.15.3.1 StepComplete

This event fires when the current step has finished whatever it had to do. This event is fired in response to the StartSequence and Next methods being called.

3.16 SDK Security Reference

The SDK uses a feature-based security model that is implemented by the Canfield Authorization subsystem. The SDK security is defined and configured based on the requirements of the SDK integrator and the end user in coordination with Canfield Tech Support. Access to patient data, image data, and image bitmaps are controlled through this security configuration.

The SDK features that can be enabled / disabled is as follows:

Feature	Description	Comments
Unowned Read	read data stored under another integrator's AppKey	
Unowned Write	write data stored under another integrator's AppKey	
Create New	create a new patient, image, or attribute library	
Patient Data Read	read patient data	
Patient Data Write	write patient data	
Patient Delete	delete a patient	
Image Data Read	read data properties associated with the image	for the purposes of security access, image thumbnails are considered image data not images.
Image Data Write	write data properties associated with the image	
Image Delete	delete an image or Image BLOB	
Image Read	read full image or image BLOB	
Image Write	write full image of image BLOB	
Image Delete	delete an image or image BLOB	
Attribute Library Read	read attribute library data	
Attribute Library Write	write attribute library data	
Attribute Library Delete	delete attribute library	

3.17 Events Reference

The following is a table of the supported SDK system events.

Event ID (varEventID)	Description	varParm01	varParm02	varParm03	v
ICanfieldSDK_Event_All	pseudo ID for all available events				
ICanfieldSDK_Event_DataChgPatient	Patient	Patient ID	array of		

	Data Changed		Patient object property names that have changed		
ICanfieldSDK_Event_DataChgPatientIDImage	Patient ID Image Changed	Patient ID	ID of ID Image		
ICanfieldSDK_Event_DataDelPatient	Patient Deleted	Patient ID			
ICanfieldSDK_Event_NavPatient	Navigated to new patient	Patient ID	Previous Patient ID		
ICanfieldSDK_Event_AppLaunch	Launched an application via the Extras Menu	executable name for launched application	command line parameters	AppKey value registered with Application::Initialize method	
ICanfieldSDK_DB_Changed	active database changed	name of new database (used in SetActiveDatabase)			

3.17.1 Event Notification Parameters

The following data structure is sent as the LPARAM of the WM_NOTIFY message.

```
typedef struct _ICanfieldSDK_EventData
{
   NMHDR      hdr;              // see WM_NOTIFY documentation
   VARIANT   varParm01;      // event specific parameter 01
   VARIANT   varParm02;      // event specific parameter 02
   VARIANT   varParm03;      // event specific parameter 03
   VARIANT   varFlags;       // event flags - usage to be defined, always VT_EMPTY
} ICanfieldSDK_EventData *pICanfieldSDK_EventData;
```

3.18 Data Protection Mask Reference

The following is a table of the supported data protection values. Values are bit flags, so multiple values can be ORed together to create a composite mask.

Protect Flag	Description

ICanfieldSDK_Protect_None	no mask
ICanfieldSDK_Protect_Delete	don't allow delete
ICanfieldSDK_Protect_DataChange	don't allow data change

3.19 Image Compression Types Reference

The following is a table of the supported compression type values used by the Image and Image List objects.

Compression Type	Description
ICanfieldSDK_Compress_JPG	JPEG compression
ICanfieldSDK_Compress_None	no compression
ICanfieldSDK_Compress_Lossless	lossless compression

3.20 Image Quality Factor Reference

The following is a table of the supported compression type values used by the Image and ImageList objects.

Compression Type	Description
ICanfieldSDK_Quality_Min	minimum allowable quality value (best quality)
ICanfieldSDK_Quality_Max	maximum allowable quality value (worst quality)
ICanfieldSDK_Quality_Default	default quality value

3.21 Capture Type Reference

The following is a table of the supported capture type values used by the Capture object.

Compression Type	Description
ICanfieldSDK_Capture_VISIA	Capture VISIA images

3.22 Capture Sequence Type Reference

The following is a table of the supported capture steps used by the Capture object.

CaptureSequenceSteps	Description
ICanfieldSDK_Seq_None	Undefined sequence
ICanfieldSDK_Seq_Standard_Left	Standard image left view
ICanfieldSDK_Seq_Standard_Front	Standard image front view
ICanfieldSDK_Seq_Standard_Right	Standard image right view
ICanfieldSDK_Seq_UV_Left	UV image left view
ICanfieldSDK_Seq_UV_Front	UV image front view
ICanfieldSDK_Seq_UV_Right	UV image right view
ICanfieldSDK_Seq_Left_AdjustMask	Adjust mask for left image(s)

ICanfieldSDK_Seq_Front_AdjustMask	Adjust mask for front image(s)
ICanfieldSDK_Seq_Right_AdjustMask	Adjust mask for right mage(s)
ICanfieldSDK_Seq_Analyze	Perform analysis on unanalyzed captures in sequence

CaptureFeatureTypes	Description
ICanfieldSDK_Feature_None	
ICanfieldSDK_Feature_Mask	Region of interest mask
ICanfieldSDK_Feature_Texture	Texture
ICanfieldSDK_Feature_Spots	Spots
ICanfieldSDK_Feature_Pores	Pores
ICanfieldSDK_Feature_Wrinkles	Wrinkles
ICanfieldSDK_Feature_UV_Spots	UV spots
ICanfieldSDK_Feature_Porphyrin	Porphyrins

3.23 Image Crop Type Reference

The following is a table of the supported image cropping types

ICanfieldSDK_CropType	Description
ICanfieldSDK_Crop_None	No Cropping – full image
ICanfieldSDK_Crop_Auto	Auto Cropping – use application defaults
ICanfieldSDK_Crop_User	User Cropping – user-specified values

3.24 Image Tag Identifier Reference

These identifiers are used with the GetTagProperty ICanfieldSDKImage method

TagID	Description
ICanfieldSDK_ImageTag_HL_COLOR_FILTER	
ICanfieldSDK_ImageTag_HL_SHARPEN_FILTER	
ICanfieldSDK_ImageTag_HL_PHOTOGRAPHER	
ICanfieldSDK_ImageTag_HL_CAPTURE_DATE	
ICanfieldSDK_ImageTag_HL_CAPTURE_TIME	
ICanfieldSDK_ImageTag_HL_IMPORT_DATE	
ICanfieldSDK_ImageTag_HL_IMPORT_TIME	
ICanfieldSDK_ImageTag_HL_IMPORT_MODULE	
ICanfieldSDK_ImageTag_HL_SOURCE_FORMAT	
ICanfieldSDK_ImageTag_HL_APPLICATION	
ICanfieldSDK_ImageTag_HL_SOURCE_NAME	
ICanfieldSDK_ImageTag_HL_CREATION_DATE	
ICanfieldSDK_ImageTag_HL_CREATION_TIME	
ICanfieldSDK_ImageTag_HL_SOFTEN_FILTER	
ICanfieldSDK_ImageTag_HL_CREATE_MODULE	
ICanfieldSDK_ImageTag_HL_AFFINE_TRANSFORMATION	
ICanfieldSDK_ImageTag_HL_MANUFACTURER	

ICanfieldSDK_ImageTag_HL_POSE_TEMPLATE	
ICanfieldSDK_ImageTag_HL_RECOMPRESSED	
ICanfieldSDK_ImageTag_HL_RESIZED	
ICanfieldSDK_ImageTag_HL_DX_RECORD_DURATION	
ICanfieldSDK_ImageTag_HL_DX_RECORD_FRAME_RATE	
ICanfieldSDK_ImageTag_HL_DX_RECORD_NUM_FRAMES	
ICanfieldSDK_ImageTag_HL_AVI_IMPORT_WIDTH	
ICanfieldSDK_ImageTag_HL_AVI_IMPORT_HEIGHT	
ICanfieldSDK_ImageTag_HL_SAVED_FROM_LOUPE	
ICanfieldSDK_ImageTag_HL_IMAGE_WIDTH	
ICanfieldSDK_ImageTag_HL_IMAGE_HEIGHT	
ICanfieldSDK_ImageTag_HL_IMAGE_SCALED	
ICanfieldSDK_ImageTag_HL_IMAGE_CROPPED	
ICanfieldSDK_ImageTag_HL_RESOLUTION	
ICanfieldSDK_ImageTag_HL_QUALITY	
ICanfieldSDK_ImageTag_HL_ISO	
ICanfieldSDK_ImageTag_HL_APERTURE	
ICanfieldSDK_ImageTag_HL_IMAGE_X_OF_Y	
ICanfieldSDK_ImageTag_HL_LIGHTING	
ICanfieldSDK_ImageTag_HL_BOOTH_NAME	
ICanfieldSDK_ImageTag_HL_WB_MODE	
ICanfieldSDK_ImageTag_HL_VIEW	
ICanfieldSDK_ImageTag_HL_FLASH_SETTINGS	
ICanfieldSDK_ImageTag_HL_SAVE_DUPLICATE_IMAGES	
ICanfieldSDK_ImageTag_HL_FILTER	
ICanfieldSDK_ImageTag_HL_FLASH_ERROR	
ICanfieldSDK_ImageTag_HL_HARDWARE_MANUFACTURER	
ICanfieldSDK_ImageTag_HL_HARDWARE_MODEL	
ICanfieldSDK_ImageTag_HL_AE_MODE	
ICanfieldSDK_ImageTag_HL_SHUTTER_SPEED	
ICanfieldSDK_ImageTag_HL_COMPENSATION	
ICanfieldSDK_ImageTag_HL_LANGUAGE_ID	
ICanfieldSDK_ImageTag_HL_ROTATION	
ICanfieldSDK_ImageTag_HL_PREVIEW_BRIGHTNESS	
ICanfieldSDK_ImageTag_HL_3D_PROCESSED	
ICanfieldSDK_ImageTag_HL_3D_CALIBRATION_FILE	
ICanfieldSDK_ImageTag_HL_ZOOM	
ICanfieldSDK_ImageTag_HL_FOCUS_TYPE	
ICanfieldSDK_ImageTag_HL_PHOTO_EFFECT	
ICanfieldSDK_ImageTag_HL_EXPOSURE_LEVEL	
ICanfieldSDK_ImageTag_HL_ZOOM_MODE	
ICanfieldSDK_ImageTag_HL_PIXELS_PER_INCH_X	
ICanfieldSDK_ImageTag_HL_FOCAL_LENGTH	
ICanfieldSDK_ImageTag_HL_HARDWARE_ID	
ICanfieldSDK_ImageTag_HL_PIXELS_PER_INCH_Y	
ICanfieldSDK_ImageTag_HL_HARDWARE_SN	
ICanfieldSDK_ImageTag_HL_FIRMWARE_REV	

ICanfieldSDK_ImageTag_HL_CROPRECT_DISPLAY	
ICanfieldSDK_ImageTag_HL_CROPRECT_ANALYZE	
ICanfieldSDK_ImageTag_HL_CAMERA_SN	

3.25 Windows SDK Data Structure Reference

The following data structures are copied from the Windows SDK and are subject to change:

3.25.1 STARTUPINFO

```
typedef struct _STARTUPINFO
{
    DWORD cb;
    LPTSTR lpReserved;
    LPTSTR lpDesktop;
    LPTSTR lpTitle;
    DWORD dwX;
    DWORD dwY;
    DWORD dwXSize;
    DWORD dwYSize;
    DWORD dwXCountChars;
    DWORD dwYCountChars;
    DWORD dwFillAttribute;
    DWORD dwFlags;
    WORD wShowWindow;
    WORD cbReserved2;
    LPBYTE lpReserved2;
    HANDLE hStdInput;
    HANDLE hStdOutput;
    HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

3.25.2 NMHDR

```
typedef struct tagNMHDR
{
    HWND        hwndFrom;   // Window handle to the control sending a message
    UINT_PTR    idFrom; // Identifier of the control sending a message
    UINT        code;       // Notification code
} NMHDR;
typedef NMHDR FAR * LPNMHDR;
```