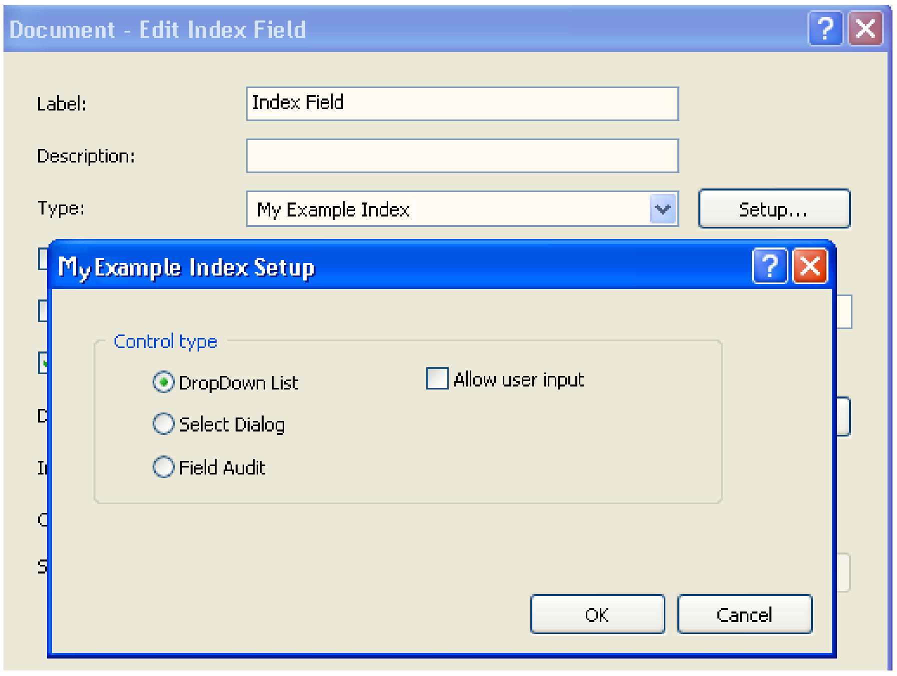
Kodak Capture Pro Index Field Type (Index) API Documentation

Version 1.1.2a April 2009



Document - Edit Index Field

Label: Index Field

Description:

Type: My Example Index

My Example Index Setup

Control type

- [✓ DropDown List]
- Select Dialog
- Field Audit

☐ Allow user input

This document is subject to change, for the latest version please visit:

www.kodak.com/go/capturepro

Table of Contents

1 Purpose of the Index Field Type (Index) API

The Index Field Type Application Programming Interface makes it possible to integrate a custom Index Field validation routine as part of Kodak Capture Pro Software. Kodak Capture Pro uses a “Field Type” to apply rules to validate the contents of any index field. Within Job Setup, you can select a “Type” for each index field. The default Types are “Single Value” and “Drop-down list”. Having selected the index field Type, the appropriate index data validation rules will be applied to that index field data. The validation of the index data can include customer specific rules or check digit algorithms that are not supported by the default field Types.

The Index API provides one or more of the following options:
Validate the operators manually entered index data
Check system derived data (e.g. OCR or Barcode values)
Provide some pre-defined index values for the user to choose during indexing
Substitute an index field’s data e.g. based on an intelligent lookup from an external source.

A custom Index field Type is a 32-bit DLL that is called from Kodak Capture Pro Software during index field population, document navigation, index check (search next invalid), batch validation / output and at other times as appropriate.

This document describes each core function of the Index API explaining its purpose complete with VC++ code snippets. A fully functioning source code example incorporating the example code snippets has been made available.

1.1 Intended audience

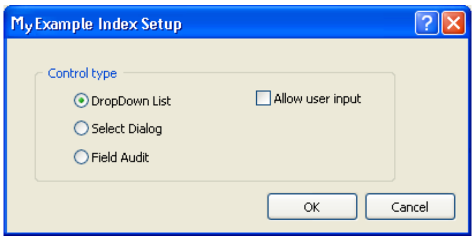
It is assumed programmers are accomplished developers who have created applications / DLL’s for Microsoft Windows.

The Index Field Type is a standard Windows DLL which could be developed in any standard programming language. Although the programming language and compiler of the API are not limited to VC++, all declarations are made in C++. The demonstration source and accompanying documentation has been written for Microsoft Visual Studio (2005) programmers who are familiar with the use of DLL’s and Class Modules within the Microsoft Visual Studio environment and have some knowledge of the Windows API.

If you are developing using Microsoft Visual Studio 2005 or 2008 you may need to install the Microsoft Visual C++ 200(x) Redistributable Package (x86) for the appropriate environment for the Index Type library to be recognized within Kodak Capture.

1.2 Sample source code

This document was written around portions of source code built under Microsoft Visual Studio 2005 and does NOT use any managed code.
The entire sample source code has been made available with this document.



The Index Field Type sample source code demonstrates 3 scenarios:
Drop-down list – display a drop-down list in the main indexing screen
Select dialog – display a separate dialog with a list of values for the user to choose
Validate Index – reject the index field unless it starts with a specific string

1.3 Using the sample source code

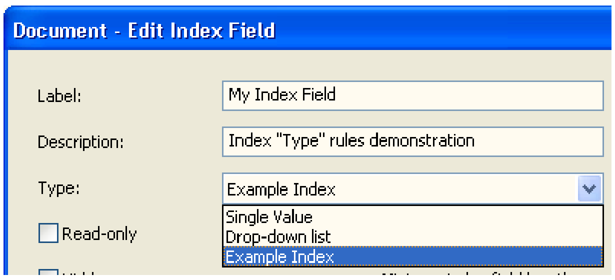
Copy the sample source files to a suitable folder on your hard disk.

In Microsoft Visual Studio 2005 select File->Open->Project/ Solution and then select:
    IDX_EXM\idx_exm.sln

The project is configured to build the output to the following folder.
    IDX_EXM\bin\kcidxapi\idx_exm\idx_exm.dpl

The resulting DPL file should be placed in the folder as described in section *5* Installing your custom Index Field Type on page 21.

In Job setup within Capture Pro, the Index Field Type will show as 'Example Index'.


Document - Edit Index Field

Label: My Index Field

Description: Index "Type" rules demonstration

Type: Example Index

2 Index field Type library description

2.1 Overview

The key objective for the Index API is to provide way to audit, accept/reject or dynamically replace index data (replacing data is only supported during batch Output processing) based on rules or external connections that are not supported by the default index field Types shipped with Kodak Capture Pro.

The index field Type API needs to control the following operations:

- The setup dialog for any index field Type specific configuration.
- Audit the index data applying rules as required to accept, reject or substitute the data
- Index field Type identification.
- Help information for use within the Windows help system (if desired).
- Provide meaningful error text for the operator on index field data reject.

To clarify the terminology used in this document, we will first look at the logical hierarchy of the contents of a batch of scanned papers and where the index data fits in this structure.

A batch is a logical structure to manage images and index data in a multiple level tree. The levels are defined as:
Batch Level
Document Level
Page Level
Image Level

The index data associated with these level groups can in theory be associated at any of the four levels, but for version 1.x of Kodak Capture Pro software only 2 levels are supported. These are the batch level index and the document level index.

A Batch is a collection of Documents. Associated Batch Level index data is related to all documents, pages and images contained within the batch.

A Document is a collection of pages that are logically related to each other (e.g. a report or multi-page letter). Associated Document Level index data is related to all pages and images contained within the document.

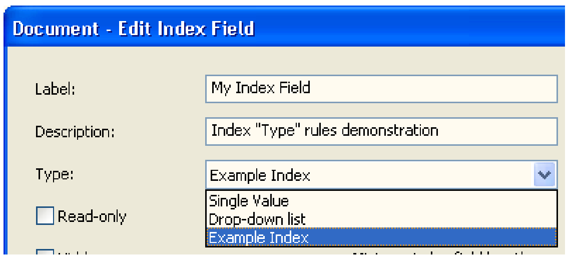
A Page is a collection of images that are the result from the digital imaging of a single piece of paper (front and rear). Currently Page Level index data is not supported. Note that scanning a Page can result in many individual image files taken from a single side of a piece of paper depending on system settings within the main Capture Pro software. E.g scanning an A4 page set to capture both front and back in both colour and black & white will result in a total of 4 image files (2 per side of the piece of paper).

An Image describes an individual image file as captured from the front or rear of a piece of paper. Currently Image Level index data is not supported.

Index data is transferred in a list of key pairs. For each key pair there is an index field name (key) and index field value (value). For example, if there is a batch level index named “BATCHNAME” and its value is “BATCH001”, then the key is “BATCHNAME” and the value is “BATCH001”.

The index keys are defined in Job setup.
The level of the index key (Batch or Document level) is automatically assigned based on the tab used during its definition in Job setup. The level of an index key will change the points during program execution at which the field Type DLL is called.

The index key's name comes from the Index Field Label used on the Edit Index Field dialog in Job setup. The index fields other properties, such as default value, input mask, output mask and index value Type are also setup on this dialog. Your custom index field Type DLL will be available for selection as an index value Type.


Document - Edit Index Field

Label: My Index Field

Description: Index "Type" rules demonstration

Type: Example Index

The index value (data) is assigned during scanning for automatically indexed index fields (e.g. barcode. OCR or system generated data) or during manual indexing. Index value is defined as a text string.

2.2 *Calling the Index Type API*

Please be aware the Index Type API is called from multiple places within Capture Pro. These include, but are not limited to, batch opening, index validation both during scanning and prior to batch Output processing, during manual indexing, etc.

The Index Type API will be called in different modes depending on the operating mode of Capture Pro.

For example, when entering Capture Pro's indexing mode, Capture Pro will call the GetEditMode() routine to create an appropriate control (ComboBox or Edit). This is to prepare for the operator to select an index value or manually type an index value. Then Capture Pro will call the GetValueList() routine or GetValue() routine dependant on the original return value of the GetEditMode() call.

Another example, when validating a batch, each index field will be validated within the batch. If the index field Type is a custom index Type created using the Index API, Capture Pro will call the ValidateValue() routine to validate the corresponding index value. If the return value of this routine is not 0 (indicating the index value has passed the validation), Capture Pro will then, if this call is during the Output phase, call the FormatOutput() routine to give the opportunity to substitute a different value as the final output index value for that index field.

2.3 *Function overview*

A Kodak Capture Pro Software compatible Field Index Type library needs to support the following routines:

GetEditMode
GetValueList
GetValue
ValidateValue
FormatOutput

2.3.1 GetEditMode()

There are 3 mode flags currently supported: KCIDXAPI_EDITMODE_LIST, KCIDXAPI_EDITMODE_USER and KCIDXAPI_EDITMODE_ALLOWTYPING.

The KCIDXAPI_EDITMODE_LIST mode is used to provide a predefined list of index values for the operator to choose during indexing. Manual typing in the index field is not allowed.
The KCIDXAPI_EDITMODE_ALLOWTYPING mode is used where you want the operator to manually type a value into an index field.
The KCIDXAPI_EDITMODE_USER is used to provide the ability to call another function from within the Capture Pro Indexing Viewer dialog. In this mode Capture Pro will display a three dots button to the right of the relevant index field which will call the function within the Index API. This can be used to implement some custom “get value” logic or show a separate GUI. Manual typing in the index field is not allowed.
The KCIDXAPI_EDITMODE_ALLOWTYPING mode can be combined in an OR function with either the KCIDXAPI_EDITMODE_LIST or the KCIDXAPI_EDITMODE_USER mode. This enables the operator to manually type the value into the index field OR choose from a pre-defined list / custom logic.

2.3.2 GetValueList()

If the Index API returned with KCIDXAPI_EDITMODE_LIST mode for the GetEditMode call, then Capture Pro will call this routine to initialize the DropDownList/ComboBox in the Index Viewer.

Typical Index API implementation is to return a predefined index value list to Capture Pro. The value list is returned as a string with individual index values delimited by “\r\n”.

2.3.3 GetValue()

If the Index API returned with KCIDXAPI_EDITMODE_USER mode for the GetEditMode call, then Capture Pro will call this routine to initialize the index value field as an Edit Window with a three dots button next to it.
When user clicks the three dots button next to the index field, this routine will be called.

2.3.4 ValidateValue()

This function is called in response to several events. See below for a full list of the events that trigger the ValidateValue call.
From V1.1.2 it is possible to update the index value during the ValidateValue call.
For Example, during batch output validation, Capture Pro will check each index field. This is to check if the index value meets the configuration which the administrator set when creating the index field in the job setup dialog. If the index field Type is selected as a custom Index Type API field, while validating this index field, Capture Pro will call this function in the API. It is the Index API’s responsibility to determine if the index value is valid or not.

The full list of the events that trigger ValidateValue to be called is:

- During scanning, it is called when data is placed into the index field on document creation. This call will not occur for a specific index field if the index field setting of "Check field during scanning" is not checked.

- When editing index data in Capture Pro’s Indexing mode:

Index field input box (EditCtrl):	change value or lose focus
Drop-down list:	change selection
Set field value using the drag&drop OCR tool	

- Open an existing batch
- Output the batch

- User language setting changed (in View and Indexing mode)

- Document navigation in Capture Pro Indexing mode: "next doc"/"previous doc"/"next invalid" in the "index" menu

- Document navigation in Capture Pro View mode: The document which is navigated to will invoke ValidateValue in response to the document receiving the focus, but the document which is navigated away from will NOT invoke ValidateValue. The event will be triggered by user actions such as: clicking a document/image in a different document within the batch explorer or thumbnail viewer, using the Next/Previous document buttons, when moving an image/page across documents using drag&drop in batch explorer.

  NOTE: If "Bypass audit during scanning" is checked on the Job Setup…Index dialog, ValidateValue is NOT called in View mode during document navigation.

- Returning from Indexing mode to View mode and an index error occurs.

  Steps could be:
  "Indexing error exists, continue saving index?" dialog popped up
  Click Cancel to stay in Indexing mode to fix the index error
  Capture Pro will reload the index data from file
  Index data is validated (ValidateValue)

You can see from the list above that ValidateValue is invoked in response to many events, the most frequent calls being triggered by document navigation and creation. Care should be taken to avoid any action in the ValidateValue routine that may suffer from time delays as this may have an impact on scanning performance or document navigation. For example, if the ValidateValue call takes long time to query a remote database, Capture Pro may appear to “wait” for a while or appear to slow down during scanning.

Note: Plugins written for versions prior to v1.1.2 will need to be modified to make use of the *szValidatedValue* structure member to allow for index updating at scan time. If the index is to be updated the new value is returned in the *szValidatedValue* member.
There is a new version of the KCIDXAPI.h which includes the new structure definition.

2.3.5 FormatOutput()

FormatOutput is called after the ValidateValue function during batch output processing only. It is not called during the normal index functionality within Capture Pro.
After calling ValidateValue, Capture Pro will call FormatOutput to get a final index value for the corresponding index key. The Index API has the chance here to modify the output of the index value. For example, add leading zeros to index value or apply an output mask to make a telephone number easier to read by adding spaces between the area code, etc.

3 Data structures

The Capture Pro Index Type API makes use of several data structures which are defined in kcidxapi.h. Some of these structures are detailed below for clarity while reading the example functions.
For the full supporting data structures, refer to the kcidxapi.h, myplugin.h and the other header files included in the sample project

4 Functions in detail

4.1 Unicode

IMPORTANT. Kodak Capture Pro software supports Unicode libraries only, so ensure your project is compiled for Unicode.

4.2 Index Type naming and identification

The file name and folder name of the Index Type must conform to the following pattern.
IDX_XYZ for the folder and hence the filename will be **IDX_XYZ.DPL**

The **IDX_** must remain as it signifies this as an Index field Type library.
The **XYZ** is a unique 3-digit identification part to identify this individual Index Type. E.g the example supplied builds to a naming of IDX_EXM, the EXM naming was chosen because this is an EXaMple Index Type.
This name should be unique within the KCIDXAPI folder.

The Index Type needs to respond to calls from Capture Pro for the following:

- The Name to display as the Index Field Type name in the Index Field Type list. This name is returned to Capture Pro in response to the GetPluginInfoEx() call.
- The description and version information for the DLL listing within the Help, About Capture Pro, System Info… screen. This description and version information should match the version information from the project resource file.
- Size to display the setup screen for the Index Field Type

4.3 Default Functions and supporting code

Several functions listed within the example source code are not detailed in this document. These functions call supporting functions in the inline code and should not be changed. An example of this type of function is the CreateUI() function.

4.4 Debug settings

By default *debug* is not supported within the Capture Pro environment and therefore you cannot *debug* your Index Type during testing on a standard installation. To use Debug within your library during development, you must first enable debug within the Capture Pro environment. To do this, simply add the following entry into the **general** section of the **env.info** file. The *env.info* file is located in the Capture Pro\System folder in the program files folder on your system. On an English Operating System the path for this file will be ***C:\Program Files\Kodak\Capture Pro\System\env.info***

```
[general]
ForegroundProcess = 1
```

If this file does not exist on your system, it can be manually created whilst Capture Pro is not running. If there is no `[general]` section in the *env.info* file, add it to the bottom of the file.

Remember to remove this setting after testing is completed & before testing the final non-debug version of your Index field Type library.

4.5 Index API Functions

Please refer to the KCIDXAPI.h header file for details of the index function data PARAM* definitions.

4.5.1 Function GetEditMode

	Data Type	**Comment**
Input	KCIDXAPI_GETEDITMODE_PARAM*	Index Type
return	BOOL	TRUE if no errors else FALSE

This function is used to set the operating mode of the index field Type.
3 modes are supported; List, Allow Typing and User.
In practice you would normally code your Index Type to support only one mode. This example demonstrates all modes.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCIDXAPI_GetEditMode)
{
      KCIDXAPI_GETEDITMODE_PARAM* p = (KCIDXAPI_GETEDITMODE_PARAM*) param;
      if (p == NULL) return FALSE;

      //Get config data
      CMyCFGACIData data;
      data.FromDataBlock(m_CurrentConfig);

      // Set the appropriate mode based on the setup data
      switch( data.m_lControlType )
      // Locally defined: 0 = MODE_LIST, 1 = MODE_USER, 2 = MODE_ALLOWTYPING
      {
      case MODE_LIST:          // Index field is a dropdown list
            m_bValidate = FALSE;
            p->wMode = KCIDXAPI_EDITMODE_LIST;
            if( data.m_lAllowUserTyping ) // Can the user type an entry as
                                          // well as choose from list
                  p->wMode = KCIDXAPI_EDITMODE_LIST |
                             KCIDXAPI_EDITMODE_ALLOWTYPING;
            break;
      case MODE_USER:          // Index field button will call a user function
            m_bValidate = FALSE;
            p->wMode = KCIDXAPI_EDITMODE_USER;
            if( data.m_lAllowUserTyping )
                  p->wMode = KCIDXAPI_EDITMODE_USER |
                             KCIDXAPI_EDITMODE_ALLOWTYPING;
            break;
      case MODE_ALLOWTYPING:  // Index field will look normal
            m_bValidate = TRUE;
            p->wMode = KCIDXAPI_EDITMODE_ALLOWTYPING;
            break;
      }

      return TRUE;
}
```

4.5.2 Function GetValueList

	Data Type	Comment
Input	KCIDXAPI_GETVALUELIST_PARAM*	Pointer to value list
return	bool	TRUE if no errors else FALSE

This function is used to populate the list of index data from which the operator can select an entry. It is called when the GetEditMode is set to `KCIDXAPI_EDITMODE_LIST` and is used to populate the list values before the Capture Pro Indexing Viewer dialog is shown to the operator. If `KCIDXAPI_EDITMODE_LIST` has been set as an OR with `KCIDXAPI_EDITMODE_ALLOWTYPING`, the operator will be allowed to choose from the list OR type a new value into the index field.
This function is typically called when a batch is created / opened to populate the list prior to any index operation.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCIDXAPI_GetValueList)
{
      // This function is called to populate the list prior to its display

      KCIDXAPI_GETVALUELIST_PARAM* p = (KCIDXAPI_GETVALUELIST_PARAM*) param;
      if( p == NULL )
            return FALSE;

      //Get config data
      CMyCFGACIData data;
      data.FromDataBlock(m_CurrentConfig);

      CString  strParameters;
      CString  strSeperator = _T("\r\n");

      // Using list mode - Fill the list with data
      State_Country* pItem = NULL;
      for (pItem = &g_map[0]; pItem < &g_map[NUMOFLIST]; pItem++)
      {
            if (pItem->nCountry == COUNTRY_US) // e.g. set to COUNTRY_US
            {
                  strParameters.Append(pItem->pchState);
                  strParameters.Append(strSeperator);
            }
      }

      // Now we have loaded the data into our list
      // get the length
      int nLen = strParameters.GetLength()*2+2;
      // Alloc data in our data block
      p->ValueList.Alloc(nLen);
      p->ValueList.Lock();
      // and copy the KCIDXAPI_GETVALUELIST_PARAM list
      wcscpy((LPTSTR)(p->ValueList.m_pData), (LPCTSTR)strParameters);
      p->ValueList.Unlock();

      return TRUE;
}
```

4.5.3 Function GetValue

	Data Type	**Comment**
Input	KCIDXAPI_GETVALUE_PARAM*	Index Value from Field
return	bool	TRUE if no errors else FALSE

When the index field Type mode is set to `KCIDXAPI_EDITMODE_USER`, Capture Pro shows a three dots button in the Index Viewer dialog next to the relevant index field. When the operator selects this button, Capture Pro calls the GetValue function in the API. This gives the possibility to construct a dialog to be shown to the operator and use it to select the index data for this index field. If `KCIDXAPI_EDITMODE_USER` has been set as an OR with `KCIDXAPI_EDITMODE_ALLOWTYPING`, the operator will be allowed to select the three dots button OR type a new value into the index field.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCIDXAPI_GetValue)
{
      // In this function we will show a dialog with the list of
      // states as used in the ListMode example.

      KCIDXAPI_GETVALUE_PARAM* p = (KCIDXAPI_GETVALUE_PARAM*) param;
      if( p == NULL )
            return FALSE;

      //Get config data
      CMyCFGACIData data;
      data.FromDataBlock(m_CurrentConfig);

      CSelectStateDlg dlg;

      CStringList lst;
      State_Country* pItem = NULL;
      for (pItem = &g_map[0]; pItem < &g_map[NUMOFLIST]; pItem++)
      {     // loop through the list and add the states to the listbox
            if (pItem->nCountry == COUNTRY_US)  // e.g set to COUNTRY_US

            {     // add each entry to the list
                  lst.AddTail(pItem->pchState);
            }
      }

      // Display our dialog which will show the list
      dlg.InitList(lst);

      CString strSelectState;

      if (IDOK == dlg.DoModal())
            dlg.GetSelectItem( strSelectState );
      else
            strSelectState.Empty();

      // Set the value of the selected entry
      p->szValue.Set( strSelectState );

      return TRUE;
}
```

4.5.4 Function ValidateValue

	Data Type	Comment
Input	KCIDXAPI_VALIDATEVALUE_PARAM	Data Structure
return	bool	TRUE if no errors else FALSE

This function is called in response to several events that are not exclusively related to an index value being entered or edited. It is also called on batch open, document navigation and batch output. Please refer to section 2.3.4 ValidateValue() for a complete description.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCIDXAPI_ValidateValue)
{
      KCIDXAPI_VALIDATEVALUE_PARAM* p = (KCIDXAPI_VALIDATEVALUE_PARAM*) param;
      if (p == NULL)
            return FALSE;

      // p->bIsValid    Set to TRUE if data value is valid
      // p->szValue     Copy of entered index value

      // In this example we will only validate data if the mode is set to
      // KCIDXAPI_EDITMODE_USER (User has manually typed an index value)
      if( !m_bValidate )
      {
            p->bIsValid = TRUE;
            p->szValidatedValue = strValue
            return TRUE;
      }

      BOOL bValid = FALSE;
      // Get the index data
      CString strValue(p->szValue.Get());

      // call our private Audit function
      if( !AuditUserEntry( strValue ) )
      {                                         // If it fails
            SetLastError(EC_AUDIT_FAILED);      // Set the error message
            p->bIsValid = FALSE;          // Indicate the data has an error
            return FALSE;
      }
      Else
      {
            p->bIsValid = TRUE;                       // Data is valid
            // new from v1.1.2
            // Must put the original value into szValidateValue member other
            // wise index will be deleted
            p->szValidatedValue = strValue;
      }

      return TRUE;
}
```

4.5.4.1 Function CMyPlugin::AuditUserEntry

Locally defined function to make the sample code more readable. This function audits the index value and returns true/false.
In the example below, we have used a simple string comparison on the starting 2 characters in the index value. We check that the index data starts with the string **V8** and return true if it does and false if it does not. It is at this point that you would implement your own audit logic such as a table lookup or pattern match/check digit algorithm.

```
BOOL CMyPlugin::AuditUserEntry( CString &sEntry )
{
      // Private Audit function
      if( sEntry.Left(2) != "V8" ) // data must start with "V8"
            return FALSE;

      return TRUE;
}
```

4.5.5 Function FormatOutput

	Data Type	Comment
Input	KCIDXAPI_FORMATOUTPUT_PARAM*	Data structure
return	bool	TRUE if no errors else FALSE

This function is called after the ValidateValue function during the 'Batch Output Process' phase of Capture Pro processing. It is not called during normal index funtionality. In this function you have the option to change or re-format the data before it is passed to the Batch Output routines if required. This is typically where you would re-format the index data by, for example, adding leading 0's.
In the example below, we have simply added the string **-0** to the index value currently associated with this index field. The Capture Pro Output processing routines will now be passed the new index value for this index field. E.g. if the original index value that was validated as True by the ValidateValue function was **123ABC**, after the FormatOutput function the value will be **123ABC-0**. The Capture Pro output processing routines will use the value **123ABC-0** as the index data for this index field.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, KCIDXAPI_FormatOutput)
{
      KCIDXAPI_FORMATOUTPUT_PARAM* p = (KCIDXAPI_FORMATOUTPUT_PARAM*) param;
      if (p == NULL)
            return FALSE;

      // p->szInputValue       Copy of the data as entered in the index field
      // p->szOutputValue      Formated data to be output

      // This example will simply add a '-0' to the end of the index value
      CString strInput(p->szInputValue.Get());

      // append '-0' to the value
      strInput += _T("-0");

      // and set it back to the index value to be output
      p->szOutputValue.Set( strInput );

      return TRUE;
}
```

4.5.6 Function GetErrorMsg

	Data Type	Comment
Input	LPCTSTR	Language
Input	DWORD	Error Code
Input	CString&	Error Message
return	bool	TRUE if no errors else FALSE

If you encounter an error you should call SetLastError with an Error Number and then return FALSE. After the library has returned FALSE, Capture Pro will call GetErrorMsg to get the error description for the Error Number set with the SetLastError call. This error message will be shown to the user by Capture Pro. You should NOT display a MessageBox in this function.
In the example below, EC_AUDIT_FAILED is the error code set in response to a failed audit (refer to the full source code example for definition of this error number). In the example code this is because the string did not start with **V8**. In this case, Capture Pro will show the error message **Audit failed! Data must start with 'V8'** in the indexing mode dialog.

```
BOOL CMyPlugin::GetErrorMsg(LPCTSTR lpszLanguage, DWORD dwErrCode, CString&
strMsg)
{
      switch (dwErrCode)
      {
      Case EC_AUDIT_FAILED:
            strMsg = _T("Audit failed! Data must start with 'V8'");
            break;
      default:
            strMsg = _T("Undefined index failure");
            break;
    }
    return TRUE;
}
```

4.6 Configuration and support Functions

To access your Index Type Library specific configuration, if applicable, you should create an CMyCFGACIData object and then call its FromDataBlock function to populate the data object members as follows

```
      CMyCFGACIData data;
      data.FromDataBlock(config);
```

4.6.1 Function GetPluginInfoEx

	Data Type	**Comment**
Input	LPCTSTR	Language
Input	CString&	Plugin Name
Input	CString&	Plugin Description
Input	CString&	Plugin Copyright information
return	void	

This function is called by Capture Pro to get the name of the Index Type Library to display in the Type list under Job Setup. It is also used to populate the Help menu ->About Kodak Capture Pro ->System Info… details.

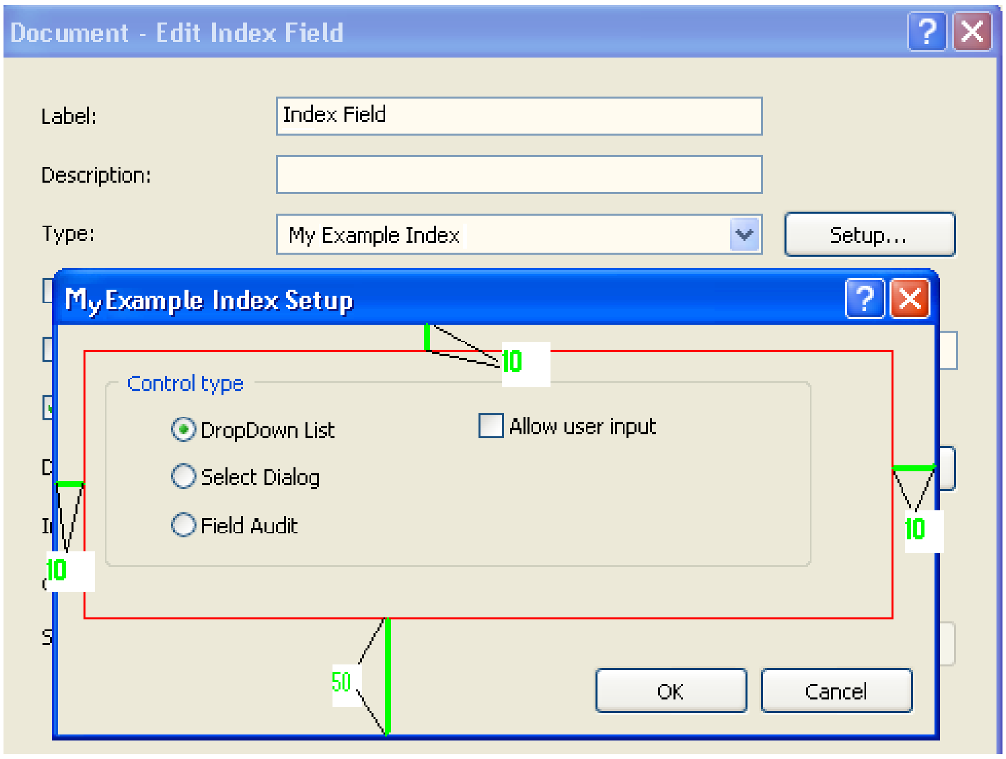
```
void CMyPlugin::GetPluginInfoEx(LPCTSTR lpszLanguage, CString& strName, 
CString& strDescription, CString& strCopyright)
{
      // You should populate strName, strDescription and strCopyright with the 
      appropriate data
      strName = _T("My Example Index");   // This is the name displayed in the 
                                    Job // Setup Screen
      strDescription = _T("This is an Example Plugin for Kodak Capture Pro.");
      strCopyright = _T("Copyright (C) 1998-2008 Eastman Kodak Company.  All 
                  rights reserved.");
#ifdef      _DEBUG
      strName += _T(" (Debug)");
#endif      //_DEBUG
}
```

4.6.2 Function GetUISize()

As a developer you will need to provide a setup dialog to allow the user to configure the Index Type specific settings.

Capture Pro handles the display of the Type setup dialog. Capture Pro displays the setup dialog, you as an Index Type programmer need to provide the template. To enable Capture Pro to size this correctly you need to create the template and also supply the width & height of the template to Capture Pro.

During Job setup, Capture pro needs to know how much space is needed on the Index Type setup dialog to display the Type's setup information.
To find this out Capture Pro will call the **GetUISize()** function. The Index Type library must return the size of the setup screen area required in device units, typically pixels.
Capture Pro will create a dialog and embed the Index Type's screen area within it as shown below.

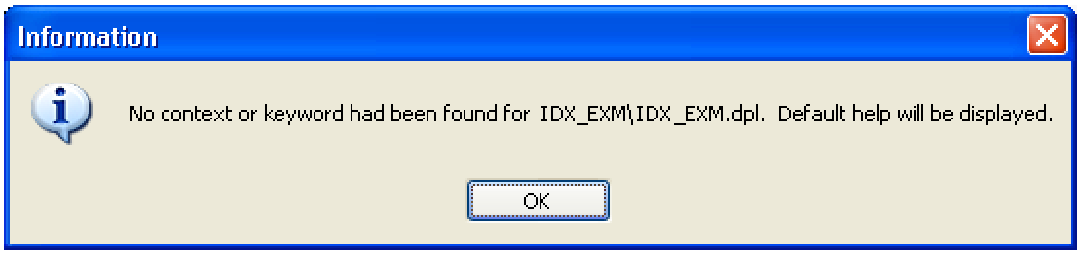


The red box outlines the template screen area provided by the Index Type (the red box will not appear on the dialog, it is for illustration only). This screen area has been placed in a section of the Setup dialog which was set to the size returned by the GetUISize() Type call. Capture Pro will create the dialog complete with OK and Cancel buttons, the Dialog title is set to the Index Type Name as defined in the GetPluginInfoEx function call. Capture Pro adds the margin space as shown in green on the screenshot above, 10 pixels top, left, right and 50 pixels on the bottom. If your Index Type returns an area too small to display the template screen area you define, scroll bars will be added automatically within the red box.

4.7 Help file information

It is the responsibility of the Index Type API developer to provide a compiled Help file for their custom Type if desired. To do this you must supply a compiled help file for your Index Type.

If you do not include the appropriate help files with your Index Type, you will get the following message when you click on the 'Help' icon on the Setup dialog.



Information

No context or keyword had been found for IDX_EXM\IDX_EXM.dpl. Default help will be displayed.

When you OK this message, the Capture Pro help will be displayed but no Index Type specific information will be available to the user.

To include help information with your Index Type you must include the Windows compatible 'help' file, complied in Microsoft Help format (chm) and a *help.info* configuration file.

The *Help.info* configuration file is a plain ASCII text format file that tells the Index Type Setup which *.chm* file to use for each language.

This is an example *help.info* file showing multiple languages:

```
[General]
ReportError = 1

[Language]
default = English_help_file.chm
en-us = English_help_file.chm
de-de = help_file_de.chm
fr-fr = help_file_fr.chm
it-it = help_file_it.chm
ja-ja = help_file_ja.chm
ko-ko = help_file_ko.chm
nl-nl = help_file_nl.chm
sv-se = help_file_sv_se.chm
zh-cn = help_file_zh_cn.chm
zh-tw = help_file_zh_tw.chm
```

If you are only going to include 1 help file for a single language then you can omit all the language specific entries and use only the default entry as follows

```
[General]
ReportError = 1

[Language]
default = General_help_file.chm
```

All help files and the *help.info* file should reside in the same folder as the Index Type plugin file.

5 Installing your custom Index Field Type

Capture Pro Batch Index Field Types are installed in their own folder in the following location on an English language system. On other language based systems, substitute the local folders in the path below.

C:\Program Files\Kodak\Capture Pro\Plugins\KCIDXAPI

To install your custom field type, you will need to create a folder for it within the KCIDXAPI folder.
Each field type, together with any supporting files are installed in their own folder, the folder name must adhere to the format described earlier.
E.g for the example code supplied, create a folder under the KCIDXAPI called IDX_EXM and put the IDX_EXM.DPL file in the newly created IDX_EXM folder.

If you are supplying Windows Help with your index field type, you will need to put the *.CHM and the *Help.info* file in this folder. See section 4.7 Help file information on page 20 for more information.

6 Document revision history

Version	Date	Author	Changes:
1.0	Nov 2008	IJP/RM	Initial Version (1.1)
1.1	Nov 2008	IJP/RJM	Incorporate CL additional descriptions
1.1.2	Mar 2009	RJM	Update to include Change data in ValidateValue
1.1.2a	Apr 2009	RJM	Corrected example in ValidateValue