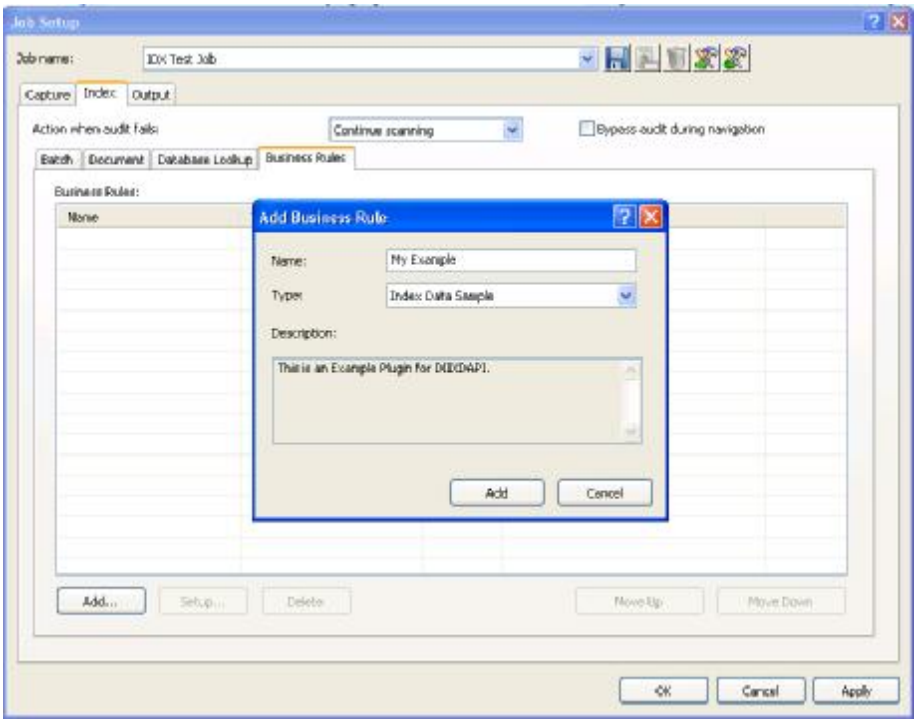
Kodak Capture Pro Index Data API Documentation

March 2010

For use with Capture Pro Version 2.5 or later



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 Data API (DIIXDAPI)

The Index Data Application Programming Interface makes it possible to apply custom business rules to index fields and data during the capture process. Business rules may include:

- Validating the contents of index fields based on the contents of other index fields
- Setting or changing the contents of index fields based on the contents of other index fields
- Hiding index fields based on the contents of other index fields
- Making index fields "Read only" based on the contents of other index fields
- Inhibiting the default Capture Pro index field input/output mask audit.

Business rules are applied in a custom Index Data Plug-in created by you. You may create one or more custom Index Data Plug-ins and specify which ones, if any, get applied in your Job Setups. If you specify more than one custom Index Data Plug-in, you may also specify the order in which the Business rules will be applied.

A custom Index Data Plug-in 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 Data 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

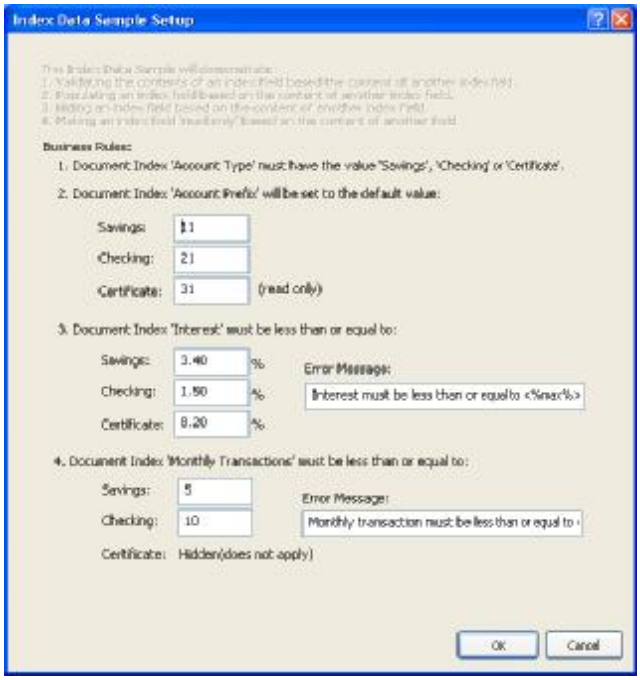
It is assumed programmers are accomplished developers who have created applications / DLL's for Microsoft Windows.

The Index Data Plug-in 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 Plug-in library to be recognized within Kodak Capture Pro.

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 with both a debug and run-time version of the example Plug-in has been made available with this document.



The Index Data Plug-in sample source code demonstrates the features using a "banking" example. To use the example, either import the Job that is contained in the "CapturePro Job" folder in the source code zip file, or create a new job having four document index fields with the following names:

- Account Type
- Account Prefix
- Interest
- Monthly Transactions

For help setting up a new Job or Importing an existing Job into Capture Pro, please refer to the Capture Pro Help or the Capture Pro Users Guide.

The example implements the following Business rules:

- The "Account Type" must be "Savings", "Checking" or "Certificate"
- The "Account Prefix" will be set to a specific value based on the "Account Type"
- If the "Account Type" is "Certificate" the "Account Prefix" shall be "read only"
- Check that the "Interest" value is less than or equal to the value specified based on the "Account Type". If the "Interest" is not correct, display the specified message.
- Check that the "Monthly Transactions" are less than or equal to the value specified based on the "Account Type".
- If the "Account Type" is "Certificate" the "Monthly Transactions" field shall be hidden.

1.3 Using the sample source code

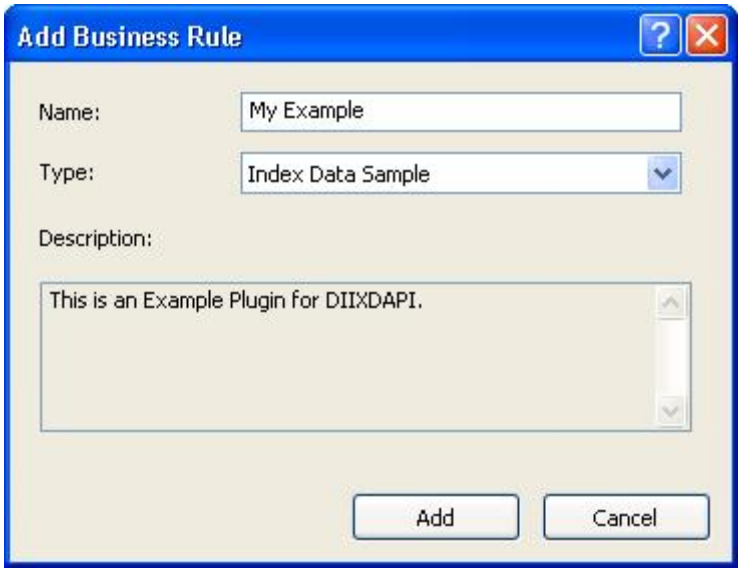
Copy the sample source files to a suitable folder on your hard disk.

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

The project is configured to build the output to the following folder.
IXD_EXM\bin\kcixdapi\ixd_exm\ixd_exm.dpl

The resulting DPL file should be placed in the folder as described in Section 5 Installing your custom Index Data Plug-in on page 23

In Job setup within Capture Pro, the "Business Rules" page will appear on the Index page. Select "Add" on the Business Rules page to get a list of all custom Index Data Plug-ins. An Index Data Plug-in may only be used once in a Job.



2 Index Data Plug-in library description

2.1 Overview

The key objective for the Index Data API is to provide a way to set, modify, audit and apply business rules to Batch and Document index data while scanning, editing and during output. The ability to programmatically control what index fields are visible to the operator and what fields may be modified by the operator is also provided.

The Index Data API needs to control the following operations:

- The setup dialog for any index data Plug-in specific configuration.
- Audit the index data applying rules as required to accept, reject or substitute the data.
- 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 versions 1 and 2 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.

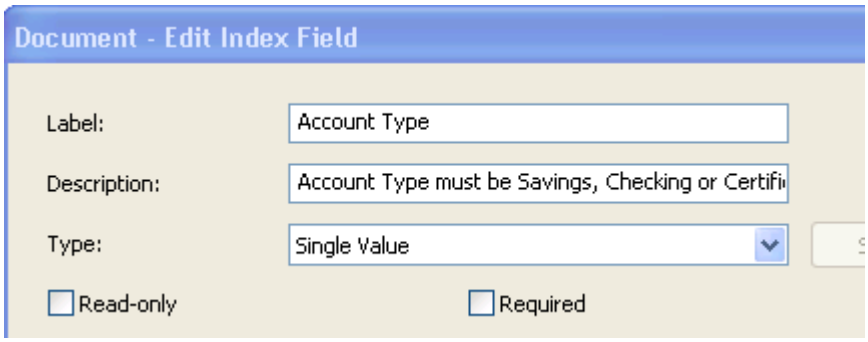
A Document is a collection of pages that are logically related to each other (e.g. a report or multi-page letter). Associated Document Level index data is related to all pages and images contained within the document.

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 Index Data Plug-in 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.



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 Data API

Please be aware that the Index Data 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.

2.3 Function overview

A Kodak Capture Pro Software compatible Index Data Plug-in library needs to support the following routines:

SetIndexList
GetIndexData
CheckIndexList

2.3.1 SetIndexList()

Capture Pro will call this routine before your Business Rules configuration UI is called. This will typically occur when your plug-in is selected and the "Setup" button is selected on the Index, Business Rules dialog in Job Setup. All Batch and Document index key pairs will be passed to the plug-in. This data should be saved and used to initialize your configuration UI.

2.3.2 GetIndexData()

Capture Pro will call this routine in response to many different events as detailed below. This is the routine that will be used to implement your business rules, validate index values or modify index values. All Batch and Document index key pairs are passed, along with the event that initiated the call. If the call was initiated while the user is in Index Edit mode, the index field that triggered the event will be identified. In Index Edit mode you will also have the ability to set the focus to any displayed index field and to display a message at the bottom of the index panel.

The full list of the events that trigger GetIndexData call are:

- During scanning, it is called on document creation after any data (i.e. default values or barcode/OCR values) is placed into the index field(s).
- Split Document is the same as the creation of a new document and the new document will inherit the original document's index values.
- When editing index data in Capture Pro's Indexing mode this call will occur when navigating between index fields when the contents of the index field have been modified. The name of the field that you just left will be indentified in the call (trigField).
- Pre Open of a new or existing batch
- Post Open of a new or existing batch
- Output of the batch
- Document navigation in Capture Pro Indexing mode: "next doc"/"previous doc"/"next invalid" in the "index" menu, entering Index Edit Mode and exiting index Edit Mode
- Document navigation in Capture Pro View mode Navigating in Batch Explorer
  Navigation in Thumbnail Viewer

2.3.3 CheckIndexList()

Capture Pro will call this function when selecting the Business Rules tab, when the OK button is selected on your plug-in configuraiton UI and when the OK/Apply button is selected on the Job setup screen. Your plug-in should verify that the configuration is correct and that all required index fields have been defined in the current Job Setup before returning. If the plug-in detects a problem, such as a required index field has not been defined or a field attribute is not correct then an error (FALSE) should be returned. Capture Pro will respond by calling GetErrorMsg(). You should provide the appropriate error message. The error will be displayed and the user will not be allowed to exit the configuration UI until the error is corrected.

3 Data structures

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

The GetIndexData function is passed the index field parameters using the structure DIIXAPI_GETINDEXDATA_PARAM.

```
struct DIIXDAPI_GETINDEXDATA_PARAM{
       DIIXDAPI_INDEX_LIST_TYPE batchIndexFieldList;
                                  //!< [in,out] batch index list
       DIIXDAPI_INDEX_LIST_TYPE documentIndexFieldList;
                                 //!< [in,out] document index list
       DIIXDAPI_INDEX_FIELD_LABEL_TYPE trigField;
                                 //!< [in] trigger field
       DIIXDAPI_INDEX_FIELD_LABEL_TYPE fieldToFocus;
                                 //!< [out] only index field name and index field
                                 // level is checked after this call returns
       DFAPI_TEXT4096_TYPE szIndexMessage;
                                 //!< [out] index description popup at the bottom of
                                 // index property grid
       DIIXDAPI_INDEX_TRIG_REASON trigReason;
                                 //!< [in] trigger reason
       HWND parentWnd;           //!< [in] parent window, used to popup model dialog
};
```

This structure provides:

- A list of all Batch index fields
- A list of all Document index fields
- The `trigField` contains the field that triggered the call. This field only contains valid data for the EditIndex trigger events (`DIIXDAPI_TRIG_EDITINDEX`) and contains the index field level, Document or Batch and the name of the field that has just been modified.
- The `fieldToFocus` is the field that should get focus next. This is typically used in Index Edit Mode to give focus to a field that may be invalid due to a change in the field that triggered the call.
- The `szIndexMessage` is typically used to display a message at the bottom of the index property window.
- The `trigReason` is an enumerated type that specifies the trigger event and is defined ddixdapi.h

```
enum DIIXDAPI_INDEX_TRIG_REASON{
       DIIXDAPI_TRIG_SCANIMAGE,          //!< during scan image
       DIIXDAPI_TRIG_SPLITDOC,           //!< split document
       DIIXDAPI_TRIG_COMMAND_NEXT_INVALID_DOC,
                                         //!< menu "Next Invalid" in index mode.
       DIIXDAPI_TRIG_EDITINDEX,          //!< edit index value in index mode
       DIIXDAPI_TRIG_PRE_OPENBATCH,      //!< pre open batch
       DIIXDAPI_TRIG_POST_OPENBATCH,     //!< post open batch
       DIIXDAPI_TRIG_OUTPUTBATCH,        //!< validate batch before output
       DIIXDAPI_TRIG_VIEWMODE_DOCNAV,    //!< navigate document in view mode
       DIIXDAPI_TRIG_INDEXMODE_DOCNAV,   //!< navigate document in index mode
   };
```

The list of Batch and Document index fields is actually a list of the DDIXDAPI_INDEX_FIELD_TYPE structure.

```
struct DIIXDAPI_INDEX_FIELD_TYPE{
       DIIXDAPI_INDEX_FIELD_LABEL_TYPE fieldLabel;    //!<[in] index field label
       DFAPI_TEXT4096_TYPE  szFieldValue;             //!<[in,out] index field value
       DFAPI_TEXT4096_TYPE szLastValue;               //!<reserved for future use
       DIIXDAPI_INDEX_FIELD_ATTRIBUTE_TYPE fieldAttribute;
                                 //!<[in,out] index field attribute
       BOOL bValid;              //!<[out] is index value valid?
       BOOL bDoNotValidate;      //!<[out] should we formula validate the index item
};
```

This structure contains all the information about a specific index field, including:

- The `fieldLabel`, which is a structure that contains the index level (Batch or Document) and the field name.
- The `szFieldValue` contains the current field value which may be modified by the plug-in
- The `szLastValue` is reserved for future use
- The `fieldAttribute` is a structure defined by DIIXDAPI_INDEX_FIELD_ATTRIBUTE_TYPE and it contains the field attributes as set in Job Setup.

  ```
  struct DIIXDAPI_INDEX_FIELD_ATTRIBUTE_TYPE{
         DIIXDAPI_INDEX_VALUE_TYPE valueType;    //!<[in] index type
         BOOL bReadonly;                  //!<[in,out] Readonly
         BOOL bRequired;                  //!<[in] Required
         BOOL bHidden;                    //!<[in,out] Hidden
         BOOL bCheckDuringScan;           //!<[in] check index field during scan
         DWORD dwMinLen;
               //!<[in] min length of index value if index field is "required"
         DFAPI_TEXT4096_TYPE szInputMask;        //!<[in] input format
         DFAPI_TEXT4096_TYPE szOutputMask;       //!<[in] output format
         };
  ```

- The `bValid` is a flag that is used to mark the index field as invalid (yellow warning triangle) in the index property window.
- The `bDoNotValidate` is a flag that may be used to bypass the normal index field validation performed by Capture Pro.

The SetIndexList and CheckIndexList functions only use the structure that passes the list of Batch and Document index fields.

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 Data plug-in naming and identification

The file name and folder name of the Index Data plug-in must conform to the following pattern.
IXD_XYZ for the folder and hence the filename will be **IXD_XYZ.DPL**

The **IXD_** must remain as it signifies this as an Index Date library.
The **XYZ** is a unique 3-digit identification part to identify this individual Index Data plug-in. E.g. the example supplied builds to a naming of IXD_EXM, the EXM naming was chosen because this is an EXaMple Index Data plug-in.
This name should be unique within the DIIXDAPI folder.

The Index Data plug-in needs to respond to calls from Capture Pro for the following:

- The Name to display as the Index Data plug-in name in the Business Rules plug-in selection list. This list appears when "Add" is selected on the Business Rules tab found on the Index page of Job Setup. This name is returned to Capture Pro in response to the `GetPluginInfoEx()` call.
- The description and version information for the DLL will appear on the Business Rules listing when selected and 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 Data plug-in

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 Index Data API Functions

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

4.4.1 Function SetIndexList

	Data Type	Comment
Input	DIIXDAPI_SETINDEXLIST_PARAM*	Batch/Doc index field list
return	BOOL	TRUE if no errors else FALSE

This function is used to initialize the plug-ins Batch and Document index field data before the configuration UI is displayed.
This function is called before the configuration UI is displayed.

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

	//Build a local list of index fields
	DIIXDAPI_INDEX_LIST_TYPE* srcbuf[2] = {&p->batchIndexFieldList,
	                                       &p->documentIndexFieldList};
	CStringList* tarbuff[2] = {&m_batchLabelList,&m_docLabelList};

	for (int i=0;i<2;i++)
	{
		tarbuff[i]->RemoveAll();
		POSITION pos = srcbuf[i]->GetHeadPosition();
		while (pos)
		{
			DIIXDAPI_INDEX_FIELD_TYPE* pSrcItem = srcbuf[i]
			                                      ->GetNext(pos);
			tarbuff[i]->AddTail(pSrcItem->fieldLabel.szFieldName.Get());
		}
	}
	//Save data for use in our configuration UI
	CMyCFGACIData data;
	if(!data.FromDataBlock(m_CurrentConfig)) return FALSE;
	return TRUE;
}
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, DIIXDAPI_SetIndexList)
{
	DIIXDAPI_SETINDEXLIST_PARAM* p = (DIIXDAPI_SETINDEXLIST_PARAM*) param;
	if (p == NULL) return FALSE;

	CMyCFGACIData data;
	if(!data.FromDataBlock(m_CurrentConfig)) return FALSE;
	return TRUE;
}
```

4.4.2 Function GetIndexData

	Data Type	Comment
Input	DIIXDAPI_GETINDEXDATA_PARAM*	Pointer to index field list
return	bool	TRUE if no errors, else FALSE

This function is used to implement your Business Rules. The Batch and Document index field data and the associated trigger event information is provided to the plug-in. The plug-in will typically only check business rules for specific trigger events. If specific trigger reasons are not to be processed by the plug-in, the plug-in should return ERR_OK (TRUE) for those events.
This function is called for the following events:

Event	Description	trigReason
During Scanning	following the capture of each document. Zonal barcode and OCR values will be available.	DIIXDAPI_TRIG_SCANIMAGE
Split Document	following a document split event.	DIIXDAPI_TRIG_SPLITDOC
Next Invalid	selecting the Next Invalid button in Index Edit mode.	DIIXDAPI_TRIG_COMMAND_NEXT_INVALID_DOC
Edit Index	navigating between index fields in Index Edit mode	DIIXDAPI_TRIG_EDITINDEX
Pre Open Batch	before the opening of a new or existing batch	DIIXDAPI_TRIG_PRE_OPENBATCH
Post Open Batch	following the opening of a new or existing batch	DIIXDAPI_TRIG_POST_OPENBATCH
Batch Output	before the batch is output	DIIXDAPI_TRIG_OUTPUTBATCH
Document Navigation (view mode)	navigating between documents in View mode from Batch Explorer or from Thumbnail Viewer	DIIXDAPI_TRIG_VIEWMODE_DOCNAV
Document Navigation (index edit mode)	navigation between documents in Index Edit mode, entering Index Edit Mode or exiting Index Edit Mode	DIIXDAPI_TRIG_INDEXMODE_DOCNAV

Note:

1. GetIndexData is called before the normal Capture Pro index data validation processes. Which is where Capture Pro by default audits the index data against the input and output masks as defined on the Index field property page of Job Setup. When replacing data values, ensure that the new data values will pass the normal validation process. The plug-in has the ability to inhibit Capture Pro from enforcing the default index field mask for any index fields that are defined. If the plug-in inhibits the default index field mask check, it will also have to inhibit the field check that happens during subsequent calls (e.g. during document navigation, batch output, etc.). One way to simplify this is to create a plug-in that performs the required audit During Scanning, Split Document and Edit Index, then setting the Capture Pro default index field mask for the field to accept all possible values that the plug-in will use.
2. GetIndexData is NOT called for the Document Navigation (view mode) event if “Bypass audit during navigation” is checked on the Index page of Job Setup.

See code sample on the next page for details.

```
DFAPI_PLUGIN_IMPLEMENT_STD_INTERFACE(CMyPlugin, DIIXDAPI_GetIndexData)
{
	// This is the entry point during normal audit processing such as
	// Scan, Edit, Document Navigation, etc.

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

	// get a copy of the local configuration data (if required)
	CMyCFGACIData data;
	if(!data.FromDataBlock(m_CurrentConfig)) return FALSE;

	// Call our own class, CMyController, Lookup function that does
	// the processing for this example audit.  We pass a pointer to
	// of the DIIXDAPI_GETINDEXDATA_PARAM data structure and a copy
	// the local configuration data.
	// The CMyController class contains all the code in response to
	// this call and is commented appropriately
	ErrorCode code = CMyController::Lookup(p,&data);
	// set the error level returned from Lookup
	SetLastError(code);

	// if all is well we return TRUE
	return Err_OK==code;
}
```

4.4.3 Function CheckIndexList

	Data Type	Comment
Input	DIIXDAPI_CHECKINDEXLIST_PARAM*	Pointer to field parameter list
return	bool	TRUE if no errors else FALSE

This function is called when the OK button is selected on the Index Data plug-in configuration UI. The plug-in will typically verify that the configuration is correct before returning. If the plug-in detects a problem, such as a required index field has not been defined or a field attribute is not correct, an error code is returned. The error will be displayed and the user will not be allowed to exit the configuration UI until the error is corrected.
This function is also called when the "Business Rules" tab is selected. If an error is returned, the error message will not be displayed, however the name of the plug-in will be displayed in red.
This function will be called when the "OK" or "Apply" buttons on the Job Setup dialog are selected and any index field configurations have changed.

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

      //get a copy of the local configuration data (if required)
      CMyCFGACIData data;
      if(!data.FromDataBlock(m_CurrentConfig)) return FALSE;

      // This is where we get the option to validate the configuration and
      // Job setup.  We use our CMyController class to do this validation
      // and pass a pointer to the IIXDAPI_CHECKINDEXLIST_PARAM data structure
      ErrorCode code = CMyController::Check(p);
      // set the error based on the return
      SetLastError(code);
      return Err_OK==code;
}
```

4.4.4 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.
As an example, the function `CMyController::Check(…)` will return `Err_TypeNotFound` if the Document level index field "Account Type" has not been defined. Capture Pro will then call `GetErrorMsg( …)` and show the error message: **`Document level index "Account Type" not defined.`** as a MessageBox.
There are two opportunities to display an error to the user. The first is when `SetIndexList(…)` is called when the plug-in configuration UI is displayed. This provides an opportunity to check if the required Batch and Document index fields have been defined to implement the desired business rules. The second opportunity is when the "OK" or "Apply" buttons are selected to close the Job Setup dialog. After the plug-in has been configured, a user may have added, modified or deleted one or more batch or document index fields. The plug-in has the opportunity when `CheckIndexList(…)` is called to check that the required index fields have be defined.

See code sample below and on the next page for details.

```
BOOL CMyPlugin::GetErrorMsg(LPCTSTR lpszLanguage, DWORD dwErrCode,
                                                  CString& strMsg)
{     // In an error (FALSE) was returned in any of the function calls
      // above, Capture Pro will call this function with the error code
      // you returned.
      // You should provide the appropriate error message to be displayed
      // by Capture Pro.

      // Note.  You should NOT display a messagebox in response to this
      // function call

      //Use our class to get the appropriate error message for the error code
      CMyController::ToErrorMessage(dwErrCode,strMsg);
      return TRUE;
}
```

```
BOOL CMyController::ToErrorMessage(DWORD dwErrCode, CString& strMsg)
{
	// When CMyPlugin returns an error, the plugin should return
	// an error message to be displayed in response to the error
	// passed in the dwErrCode parameter

	// Note.  You should NOT show a messagebox to display an error message
	// in response to this call, the error message must be returned in the
	// strMsg parameter which will be displayed by Capture Pro
	switch(dwErrCode)
	{
	case Err_OK:
		strMsg = _T("No error found.");
		break;
	case Err_InvalidParam:
		strMsg = _T("Invalid param passed to plug-in");
		break;
	case Err_TypeNotFound:
		strMsg = _T("Document level index \"Account Type\" not defined.");
		break;
	case Err_PrefixNotFound:
		strMsg = _T("Document level index \"Account Prefix\" not
                                                        defined.");
		break;
	case Err_InterestNotFound:
		strMsg = _T("Document level index \"Interest\" not defined.");
		break;
	case Err_MonthyNotFound:
		strMsg = _T("Document level index \"Monthly Transactions\" not
                                                                defined.");
		break;
	default:
		strMsg = _T("Unknown error.");
		break;
	}
	return TRUE;
}
```

4.5 Configuration and support Functions

To access your Index Data plug-in 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);
```

4.5.1 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 Data plug-in to display in the Type list on the Add Business Rule dialog that is displayed when the 'Add' button is selected on the Job Setup, Index, Business Rules dialog. It is also used to populate the Help menu ->About Kodak Capture Pro ->System Info… details.

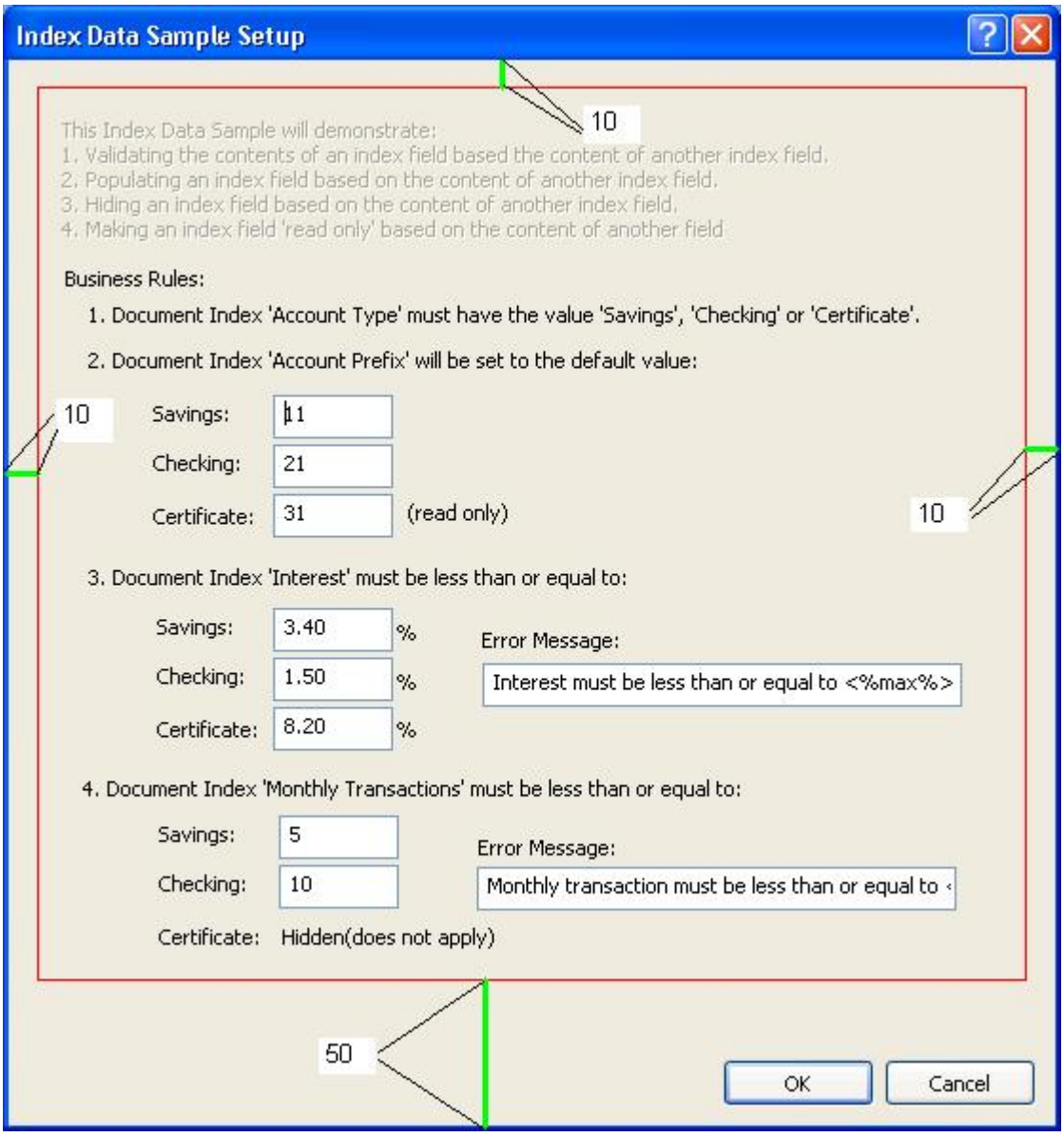
```
void CMyPlugin::GetPluginInfoEx(LPCTSTR lpszLanguage, CString& strName,
                        CString& strDescription, CString& strCopyright)
{
      //_This is where we give the Plugin its name shown in the UI
      strName = _T("Index Data Sample");
      strDescription = _T("This is an Example Plugin for DIIXDAPI.");
      strCopyright = _T("Copyright (C) 1998-2010 Eastman Kodak Company.
                                                  All rights reserved.");
#ifdef      _DEBUG
      strName += _T("(Debug)");
#endif      //_DEBUG
}
```

4.5.2 Function GetUISize()

As a developer you will need to provide a setup dialog to allow the user to configure the Index Data plug-in specific settings.

Capture Pro handles the display of the plug-in setup dialog. Capture Pro displays the setup dialog, you as an Index Data plug-in programmer need to provide the template. To enable Capture Pro to size the dialog 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 Data setup dialog to display the plug-in's setup information.
To find this out Capture Pro will call the **`GetUISize()`** function. The Index Data plug-in 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 Data setup screen area within it as shown below.



The red box outlines the template screen area provided by the Index Data plug-in (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 Data plug-in 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 Data plug-in returns an area too small to display the template screen area you define, scroll bars will be added automatically within the red box.

4.6 Help file information

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

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

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

To include help information with your Index Data plug-in 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 Data plug-in 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
cs-cz = help_file_cs_cz.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
ru-ru = helf_file_ru.chm
sv-se = help_file_sv_se.chm
tr-tr = help_file_tr.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 Data plug-in file.

5 Installing your custom Index Data Plug-in

Capture Pro Index Data plug-ins 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\DIIXDAPI

To install your custom field type, you will need to create a folder for it within the DIIXDAPI folder.
Each plug-in, together with any supporting files are installed in their own separate folder. The folder name must adhere to the format described in Section 4.2 Index Data plug-in naming and identification on page 12.
E.g. for the example code supplied, create a folder under the DIIXDAPI folder called IXD_EXM and put the IXD_EXM.DPL file in the newly created IXD_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 4.6 Help file information on page 22 for more information.

6 High level flow of the example API source code

6.1 Manually indexing a document

Enter Index data into a field, tab to move to the next index field.

`CMyplugin::GetIndexData` is called.

Determine which field has called the function by interrogating the `trgField` member of the `DIIXDAPI_GETINDEXDATA_PARAM` structure. Ensure the field is a document level index (check the index field type – Batch or Document - `DIIXDAPI_INDEX_LEVEL`).

If the plug-in is to audit this field it will typically note the index value. Based on the value you may want to either audit or set the value of another field.

Return `Err_OK` to indicate no error

Focus is then set to the next field in the Index screen unless the plug-in has set the focus to another field

6.1.1 Example

The sample Index Data plug-in source code is built to only trigger (TrigReason) during the folowing events;
Edit Index (`DIIXDAPI_TRIG_EDITINDEX`)
Document Navigation in index edit mode (`DIIXDAPI_TRIG_INDEXMODE_DOCNAV`)
Next Invalid (`DIIXDAPI_TRIG_COMMAND_NEXT_INVALID_DOC`)

In Indexing mode, the user enters ‘Savings’ in the field “Account Type” and presses tab to move to the next field.

The `CMyplugin::GetIndexData` is called. This passes control to a local class `CMyController` which does the work in our example. The `AdjustGUI` function is then called to optionally set field attributes based on the defined business rules as described in section 1.2 Sample source code on page 5. In our example, because we have entered ‘Savings’ as the account type, we set the field “Monthly Transactions” and “Account Prefix” to be Not Read Only and Not Hidden.

The Example plugin then determins that the API was called as a result of an index edit and verifies that we are on a document level index field.

The code checks / sets the Account Prefix, the Interest rate and the Monthly Transaction in the following order.

Set “Account prefix” value:

The plug-in gets pointers to the “Account Prefix” and ”Account Type” data structures. We check / set the prefix value in the `AdjustAccountPrefixItem` function. The account type is Savings (AT_SAVINGS) so the plug-in sets the “Account Prefix” value to that defined by the business rules.
In our example the value is set to 11.

Verify the “Interest” value:

After getting the account type (AT_SAVINGS) it checks to see if the value in the interest field is greater than that defined in the business rules. If it is, `fieldToFocus` is set to the “Interest” field, the invlaid flag is set and the error message is built before returning `FALSE`. This action inhibits further processing so that the first error encountered is reported to Capture Pro.

If the “Interest” value is within the rules, the function returns `TRUE`.

Verify the “Monthly Transactions” value

After getting the account type (AT_SAVINGS) it checks to see if the value entered in the “Monthly Transactions” field is greater that than defined in the business rules. If it is, Focus is set to the “Monthly Transaction” field, the invalid flag is set and the error message is built before returning `FALSE`.

If the “Monthly Transactions” value is within the rules, the function returns `TRUE`.

If none of the above functions returned `FALSE`, when we return `Err_OK` the field focus is set by Capture Pro to the next field in the Job. In the case that an error was encountered in the field value checks above, Capture Pro sets the field focus and displays any error message based on the field attributes set.

6.2 Setting an index value during scanning

Although not required by the Business Rules example, the sample source code reacts to the ScanImage event to show the process required to invoke Business Rules whilst scanning.

Any default values (barcode data, OCR data, etc.) as defined in the Capture Pro Job setup will be placed in the index fields by Capture Pro before calling the plug-in.

The SCANIMAGE trigger event does not provide information related to which field triggered the call, as this detail is not relevant for this call. It is down to the Business Rules Plug-in programmer to know the fields and get or set the data appropriately

The code in the example sets the Interest value based on the ‘Account type’ field data value. To achieve this, you will need to either set a default value for the index field, or setup a barcode / OCR extract to populate the index field so that the index value is set by Capture Pro during scanning. The process is as follows:

Capture Pro puts any available Default Values into the index fields.
In response to `DIIXDAPI_TRIG_SCANIMAGE param->trigReason`

Get a pointer to the data structure for the ‘Account Type’ Field
Get a pointer to the ‘Interest’ field

Read the `szFieldValue` member of the Account Type field pointer
If the data value is “Savings”
    Set the `szFieldValue` member variable of the Interest field pointer to “1.23”
Else
    Set the `szFieldValue` member variable of the Interest field pointer to “3.21”

Return `Err_OK` to indicate no error

7 Document revision history

Version	Date	Author	Changes:
1.0	Jan 2010	JDM	Initial Version (1.0)
1.1	Mar 2010	RJM/IJP	Document example source code