Oracle® Forms Recognition
Scripting Guide
10g Release 3 (10.1.3.5.0)

October 2009

ORACLE

Oracle Forms Recognition Scripting Guide

10g Release 3 (10.1.3.5.0)

This guide contains sample code that is provided for educational purposes only and not supported by Oracle Support Services. It has been tested and works as documented. However, Oracle does not guarantee that it will work for you, so be sure to test it in your environment before relying on it.

Proofread the sample code before using it! Due to the differences in the way text editors, e-mail packages and operating systems handle text formatting (spaces, tabs and carriage returns), the sample code may not be in an executable state when you first receive it. Check over the sample code to ensure that errors of this type are corrected.

Contents

Chapter 1 Introducing Oracle Forms Recognition

1.1 About Oracle Forms Recognition

Oracle Forms Recognition is a product suite from Oracle Corporation for automatic processing of arbitrary **document** input. The system reads and analyzes classical forms, unstructured documents and logical structured documents and extracts information from the documents needed for subsequent processes. The product is based on a trainable approach that minimizes user definition tasks. New patterns are generated automatically using the neural network technology of in multiple ways for classification and knowledge-based extraction.

The product is designed to handle large volumes of documents of arbitrary origin. It serves as a portal for all incoming documents for large enterprises. The system is not only adapted to handle scanned documents or faxes using OCR, but also all types of electronic office document formats and emails. The main focus is on converting information on paper to electronic information.

1.2 Purpose of Oracle Forms Recognition

The recognition of paperbound information in fixed physical layout has reached a state of the art that it can be used routinely to capture large amounts of data. But today, most of the documents that need to be processed by an enterprise (such as in a mail room) are not forms, but logically structured or unstructured documents. Especially when workflow systems, ECM (electronic content management) systems and archive systems are used, there is a specific need for organizations to speed up the data acquisition process. To be able to process documents electronically in a workflow system, it is necessary to capture information and **images** of paper documents as early and as fast as possible.

Requirements for Oracle Forms Recognition are:

- Classification of documents as input to workflow and DMS. The ability to route a document to an appropriate task, correct classification, and extract relevant data from the document are required.
- Reduce data entry by reading contents of the document and transferring these contents with high security in a structured form to the subsequent systems.
- Easy to adapt by the customer to the changing organizational surroundings. The system is based on a neural network that must have only a limited rule base and be self-learning as far as possible.
- Easy to tune and adapt by the customer to different types of documents.
- Smooth integration of workflow and archive system environments.
- Transparency and supervision of the process by statistic and analysis tools.

One basic goal of the product is to integrate the organizational style of each customer when setting parameters and configuring the application. Only the customer knows the specifics of their documents and the way they are treated today. The mapping of organizational knowledge onto the technological definitions necessary must be simple, transparent, and intuitive. We must be able to store the knowledge of today's organization in the document knowledge base. The technology must not be visible to the end user, who should be able to customize the product. Sophisticated user interfaces and the "intelligent" algorithms make this possible.

1.3 Relationship of Oracle Forms Recognition to Other Products

With Oracle Forms Recognition, the first step in the document and information processing chain is solved: The document input process. You would use products with document management and with knowledge management of an enterprise. Oracle Forms Recognition is included as a front end to other products. Standard interfaces are provided for workflow solutions, ECM systems and archiving products. Oracle Forms Recognition provides the information and data processed in knowledge management products.

Chapter 2 Oracle Forms Recognition Overview

2.1 Oracle Forms Recognition Modules

After installation of the product, you will find three modules on your computer:

Oracle Forms Recognition Designer (DstDsr.exe) is used as an integrated definition, development, and testing tool. *Designer* is used in the project preparation phase to generate the system and tune classification and extraction. The results are stored in the project file.

Oracle Forms Recognition Runtime Server (DstAdmin.msc) is an operator-free process for automatically processing the document input based on the definitions in a project file. The input are Tiff files organized in **batches**. The output is freely configurable in the Export module, but normally consists of text files and the Tiff /PDF for archiving.

Oracle Forms Recognition Verifier (DstVer.exe) is the interactive application to correct all of the documents that could not be correctly classified or extracted. It shows all invalid documents to the user and requests the input of **valid** data.

Oracle Forms Recognition has four main user interfaces:

1) Design interface: The design interface is part of *Oracle Forms Recognition Designer*. It is used to prepare a project to define document classes, main parameters, and rules for classification and extraction. In the design environment, validation of data and export to subsequent systems is defined. A Visual Basic-compatible scripting environment (Sax basic) is provided for project-specific classification and extraction rules, validation routines, database lookup, and export formatting.
2) Training interface: The training interface is integrated into *Oracle Forms Recognition Designer*. It serves as a workbench for teaching the system new documents by using the results of manual classification and data entry. In addition, already indexed data from existing archives can be used. This mode is called “supervised training and tuning.” The results of this process are stored as samples in the Learn Set. In an explicit learning step, these samples are converted to internal patterns that are used for subsequent automatic processing.
3) Runtime interface: *Oracle Forms Recognition Runtime Server* runs on a dedicated PC as a service or program, processing documents automatically. *Runtime Server* classifies and reads documents from file input and writes the results to standard output files or to output channels like databases defined in script. The capacity of the server can be scaled arbitrarily by adding more CPU power and running several servers in parallel. An interface to full text recognition for generating PDF or full text indices is provided optionally.
4) Verifier interface: *Oracle Forms Recognition Verifier* is used by subject-matter experts to verify and organize extraction results and to interactively train tables and learn sets, the former through Table Extraction and the latter through the Supervised Learning Manager add-on.

2.2 Technical Summary

Oracle Forms Recognition can analyze entire documents, replacing and enhancing reading in a fixed zone layout (forms reading). This section gives an overview on the concept and the technical foundations of this process.

In many cases, information on paper is not structured in a physical layout called form, but has a logical structure that can be understood by humans. In fact, most data on documents give specific hints for humans to get the correct information. Examples are borders around zones, keywords, and lines. *Oracle Forms Recognition* implements technologies that use the structure that a generic document has been given to be readable by a human. The patterns for classification and extraction are predefined for a specific set of document classes in the document knowledge base and learned by the technology from given examples.

In the first step, an image is analyzed and, if necessary, converted into an internal representation called a **Workdoc**.

Technologies that are used include:
- Standard OCR technology
- Image preprocessing and enhancement
- Line analysis
- Layout analysis
- Conversion of electronic office formats (*.doc, *.xls, *.htm, *.msg) using INSO filters

The resulting representation of a document as a Workdoc is analyzed and processed through learned structures. In addition to the self-learning neural network technology, classical forms reading technologies can be applied. These technologies are combined in a simple but powerful way to create document understanding.

The product is built on components that all work on two central objects representing document and knowledge base: the **Workdoc** and the **Project**. The Workdoc, a general internal representation of any kind of document, no matter what its origin, is a scanned paper document, an email, or a fax. It also serves as a container to carry all information of subsequent classification and extraction processes. The process steps – scanning, classification, extraction, correction, and export – are all separate modules that work on the Workdoc, which contains the final results.

2.3 Process Overview

The basic process of document input is described in Figure 2-1:

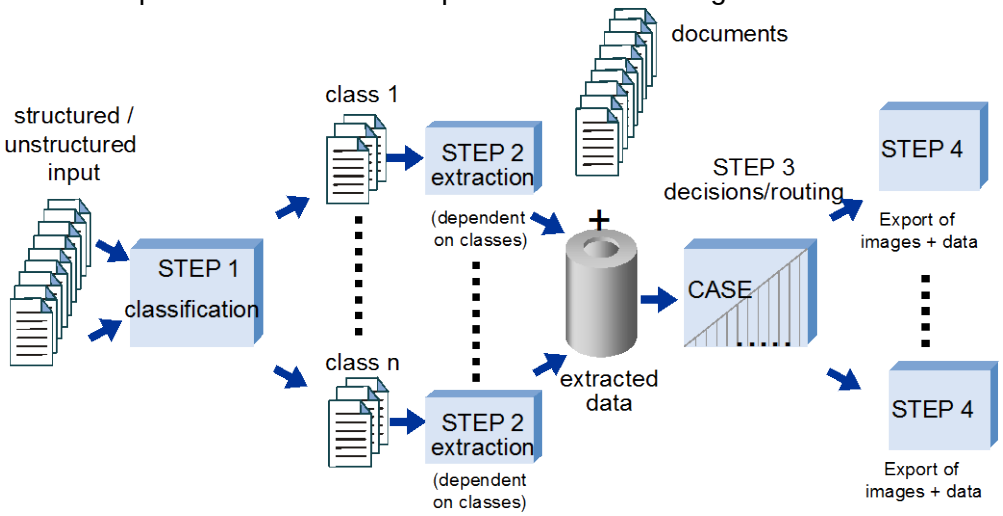


Figure 2-1: Basic process of document input

Structured or unstructured document input is produced by scanning, fax-server, or other processes. The documents are stored in file system as Tiff files or Office formats, respectively. Structure may have been imposed during scanning to organize multi-page documents into multi-Tiffs and documents that belong together into a structure that is called a **folder**. For example, a folder might contain a letter of a customer, several invoices, and several checks.

1) In the first step, documents are classified. If OCR is needed, it is done automatically or on demand.
2) In the second step, once the class of the document is known, the fields of that class are loaded and extraction for each field is performed using class-specific settings or inherited settings. The process stores extracted data and documents.
3) In the third step, a case decision can be imposed using script, sending documents to appropriate subsequent systems.
4) In the fourth step, an export is performed and documents and data are released from the system and forwarded to connected systems.

2.4 Project

The complete process of automatically analyzing incoming documents relies on a set of definitions, parameters, and trained networks that are stored in a structure called the Document Knowledge Base. This structure contains the knowledge of how to process the documents and represents the definition data storage. Internally, it is represented by the *Oracle Forms Recognition* Project, a file with the extension of *.sdp. The Project is portable, which means that a Project that has been defined in one place can be copied to another place just by copying the *.sdp. Predefined Projects can be prepared for vertical solutions and distributed along with the base software. The Project has a built-in version control system. The following is an overview of some of the objects that are stored in the Project:

- Document class names
- networks
- Pointers to Learn Set and **status** of Learn Set
- Template matching patterns
- Phrases and rules
- Analysis format definitions
- Field names and types
- Standard validation rules
- Script for Classification, Extraction & Validation and Export
- General settings and paths

A detailed description of properties and methods of the project object and the script interface is given in the object reference.

2.4.1 Field Types

There are two field types available, header fields, containing text or table fields. Text is the default field type. If you want to have a table, the field type has to be changed to table. The field type will be inherited by all derived field definitions and cannot be changed in a derived class. A text field can contain one or more lines of text. You can access the content of the field using the Text or Line property of a field. A table field contains a table object that represents a two-dimensional array of strings. The number of columns is predefined by the settings of the table analysis **engine** and the numbers of rows can vary depending on the current document. The table data can be accessed using the Table property of the field.

2.4.2 Smart Indexing

Smart indexing provides the feature of automatic data completion within a document based on database information. Therefore, you need one or more index fields that can be used to search for a record set within a database table. From the returned record set, it is possible to fill other fields of the document with data. This can be used to look up a customer ID and get all address information for that customer from the database, then copy the information to fields of the document. Smart indexing is available during automatic extraction and during manual verification.

2.5 Workdoc

The Workdoc represents the runtime data of a document. In the first step, the incoming document is converted to a Workdoc by using OCR for Images or INSO filters for Office documents. After the Workdoc is generated, all modules and algorithms work on it (hence the reason for the name), while the original document is treated as an attachment for displaying and further analysis. An overview is given in the graph in ***Figure 2-1: Basic process of document input***.

The Workdoc supplies an infrastructure for the various steps in document analysis. OCR or conversion results are stored in a structure (called **Worktext**) that preserves also the geometry of words and other parameters of each single character. The data is used in the later processes. The Workdoc can be displayed in the viewer, where words, candidates, and other elements can be highlighted. A Workdoc takes implicit account of multi-page documents. It does not matter if a word is located on the first page or any of the following pages because all words are internally mapped to one structure. Therefore, dealing with multi-page documents is natural and easy. The image with several pages is only an attachment to one Workdoc. The Workdoc also contains the fields with Runtime data gained from extraction and later correction. The process of analyzing a document is completed when the Workdoc is attached to a document class, and all of the fields are filled by the extraction step and have received a “valid” status from validation.

The Workdoc consists of a file with the extension *.wdc.

A detailed description of methods and properties of the Workdoc object and the script interface are given in the object reference.

Chapter 3 Oracle Forms Recognition Script

Oracle Forms Recognition is a general product for very different markets and applications. But it also is able to handle the complex document input processing in many different colors and shades.

Sometimes it is necessary to customize the “normal” *Oracle Forms Recognition* processing. Scripting is particularly useful for setting up the evaluation of candidates, validation, and reporting.

To this end, a script engine (SaxBasic V.6.0) is built in to provide flexibility to the classification, extraction, validation, and export processes. The script can be used to enhance the standard functionality by customizing the code. It can also be used to fit the system into the different customer organizational needs.

This section describes how script is used in *Oracle Forms Recognition* . The integrator can include custom Basic code that is executed in different events. Several objects of the Runtime environment and components are accessible from script. To use script, you first must decide which event to use and then add your custom code there, using the appropriate methods and properties of the internal or external objects.

In any case, you should try to use standard methods provided by *Oracle Forms Recognition* engines before you start to implement a script algorithm.

3.1 How to Use the Script Editor

To open the script editor, first switch to Definition Mode in *Oracle Forms Recognition Designer*. Open the script editor by clicking the button for script editing in the toolbar or using the menu “*Edit/Show DocClass/Edit Script*”.

A new dialog box opens showing the script editor. When a document class has been selected in the Definition Mode, the script of the currently selected document class is shown. Otherwise, the one for the project is shown. The script editor can simultaneously display the script code for several document classes. The code for each document class is displayed in a separate sheet that is accessible through the tabs on the left side of the editor window or by selecting the “*sheet*” command in the menu bar. The editor can display to nine sheets. The first sheet always contains the code for the Project. If a ninth document class is selected, a message box occurs that no more sheets can be opened. Then the last sheet shows the code for the last selected document class, the eighth sheet shows then the code for the document class that was selected before the last document class, and so on. That means that the code for each document class is displayed in the sheet before and the code for the document class that had been shown at the second sheet will then be skipped.

The script editor supports a multitude of methods that you can use to work with the code. An online help for all editor functions is provided. Sax Basic provides the core language definition. It is compatible with Visual Basic for Applications. Online documentation is also provided for the basic language.

An online help for the Cedar and Cairo components is available but is currently reachable only via the object browser. For each method or property of an object, a helpstring is displayed. The detailed description will be shown when the question mark is clicked. Note that the Winhelp files for the Cairo and Cedar components must have been copied to the adequate components’ directory.

3.1.1 Graphical User interfaces

All menu commands can also be activated through the context menu (right-mouse click.)

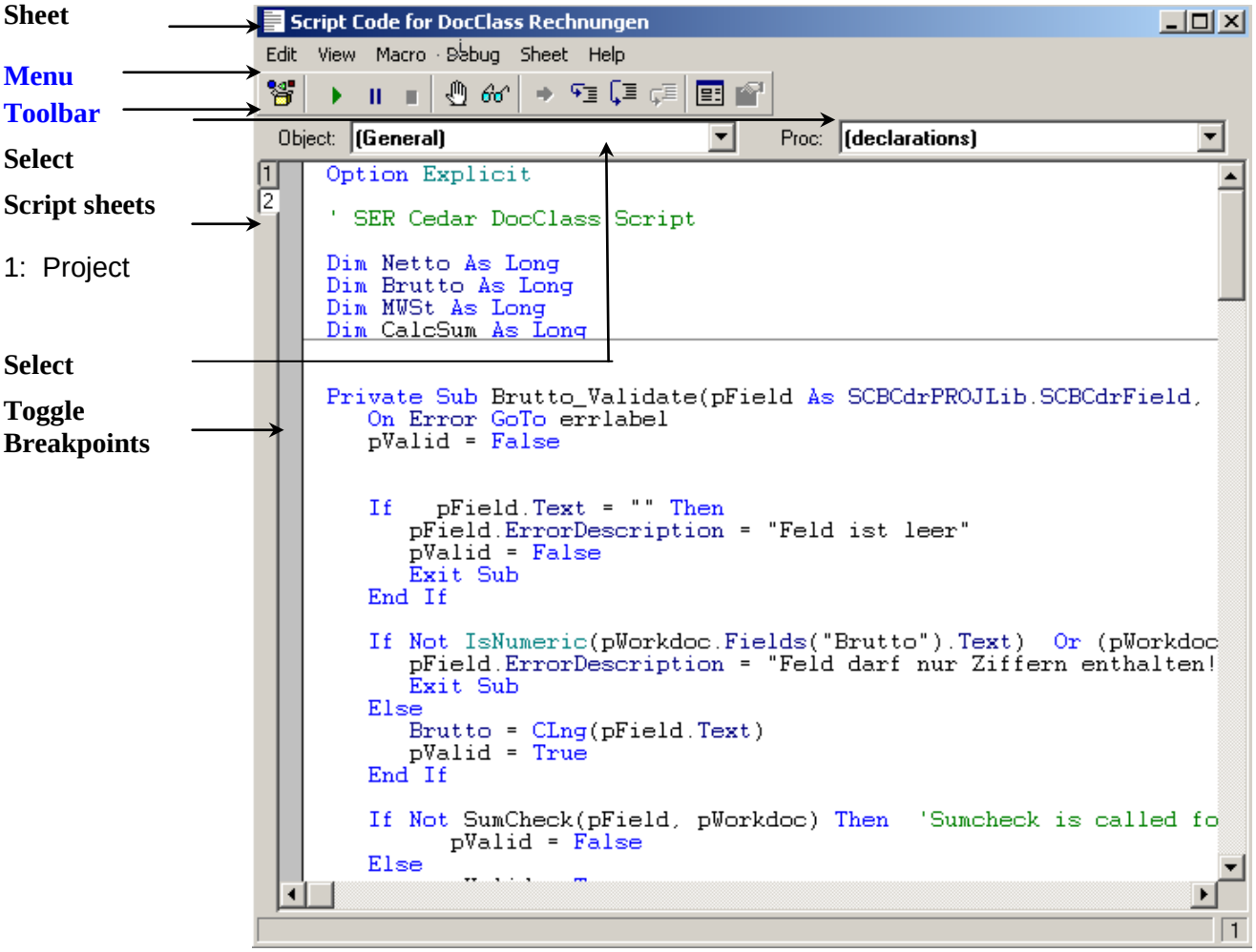


Figure 3-1: Script Editor interface

3.1.1.1 Menus

Menu Item	Description
Edit	This menu contains commands to manipulate the edited text, like copy and paste, undo, redo, find, etc., but also to select references, to edit User Dialog, and to edit the Project properties. This menu contains commands to manipulate the edited text, such as Copy, Paste, Undo, Redo, and Find. It also contains commands to elect references, to edit User Dialog, and to edit Project properties.
View	Used to view different tracing windows, toolbar, edit buttons, status bar and to change the font for the macro window.
Macro	Start, Pause and Stop the script execution.
Debug	Provides the options for debugging macros and modules like Step in, Step over, Toggle Break.
Sheet	Allows to change from the project script to others.
Help	Editor Help gives support on the GUI, and Language Help refers to Sax Basic.

3.1.1.2 Toolbar

Control	Keyboard Shortcut	Description
	CTRL + X	Cut
	CTRL+ C	Copy
	Ctrl + V	Paste
	CTRL + Z	Undo last action
	CTRL + Y	Redo last action
		Object Browser
	F5	Start Script (Run)
	ESC	Pause Script
		Stop Script
	F9	Toggle Break
		Evaluate Expression
		Show current statement
	F8	Step into subroutine
	CTRL + F8	Step out of subroutine
	SHIFT + F8	Step over subroutine
		Edit User Dialog
		Edit Module Properties

Figure 3-2: Script Editor Toolbar

3.1.2 Object Browser

In script, ActiveX objects can be registered and used via their dispatch interface. The built-in object browser displays the interfaces of the objects. See ***Figure 3-3: Object Browser***.

A number of objects from the component base (SCB) that are used in *Oracle Forms Recognition* are referenced by default and are accessible for script programmers. In the example of the object browser shown above, the methods and properties of the Workdoc component are shown. If you follow the value "*Fields,*" you are led to the methods and properties of the Fields object. The main objects are Workdoc, Worktext, and Image. A description of interfaces is in the Object Reference.

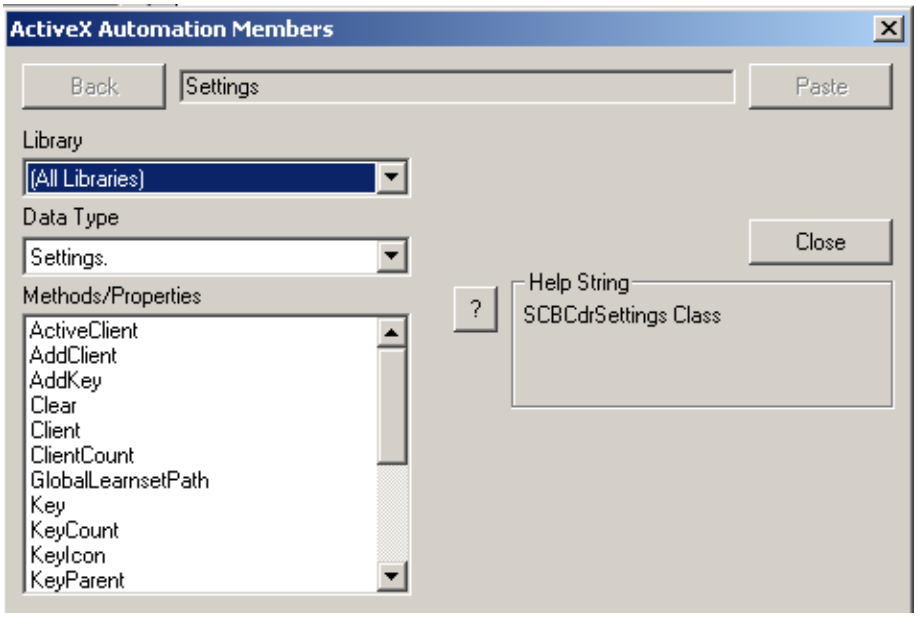


Figure 3-3: Object Browser

Any other additional objects can be referenced using the menu command "*Edit/References.*" In the dialog that is shown, any registered ActiveX component is displayed.

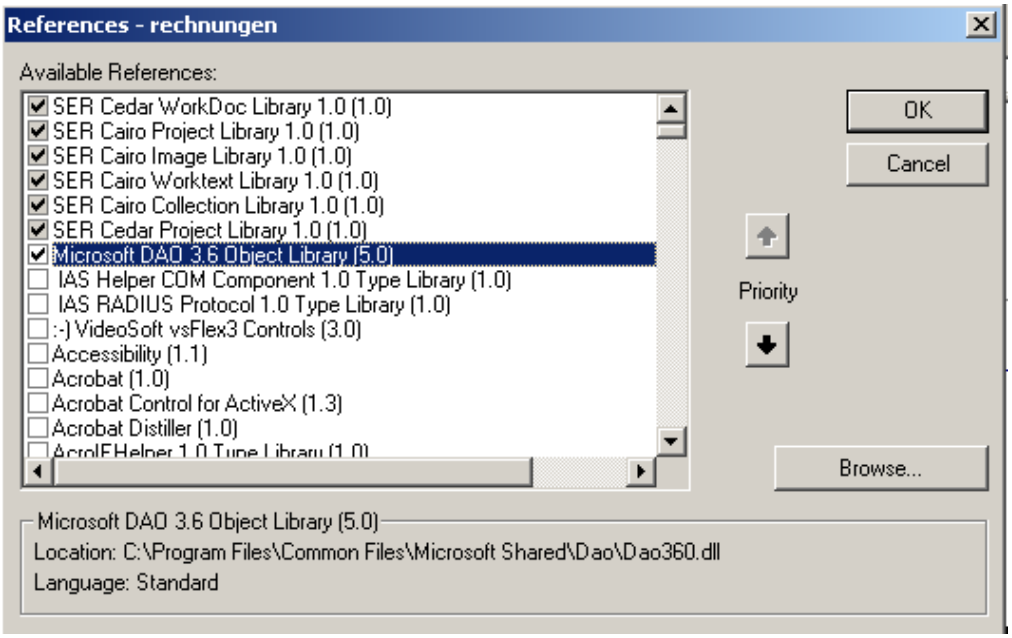


Figure 3-4: Edit/References

When you select one of the entries, the component is immediately available in script. The methods and properties of this component can be seen using the object browser.

An important application of external components is the access of databases using ODBC. To access databases using ODBC, the Microsoft DAO or ADO components have to be installed and referenced. Then it is possible to access any database from script (e.g., in Export.)

A second way to open the script code is when a message box pops up during classification or extraction due to a script error:

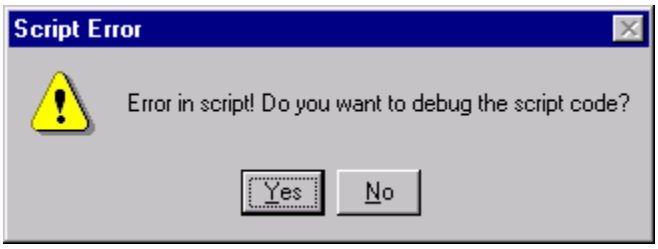


Figure 3-5: Script Error Message box

If you click *Yes,* the script editor is opened in Debug Mode. The reason for the error appears in the status bar of the script editor window. In Debug Mode, the script is still running. To edit the script, click the *Stop* button (red square) and start editing. Be sure to restart the script again before closing the window.

3.1.3 Debug Mode

A script can be started by clicking  Start. This causes a syntax check and the tracing windows are displayed:

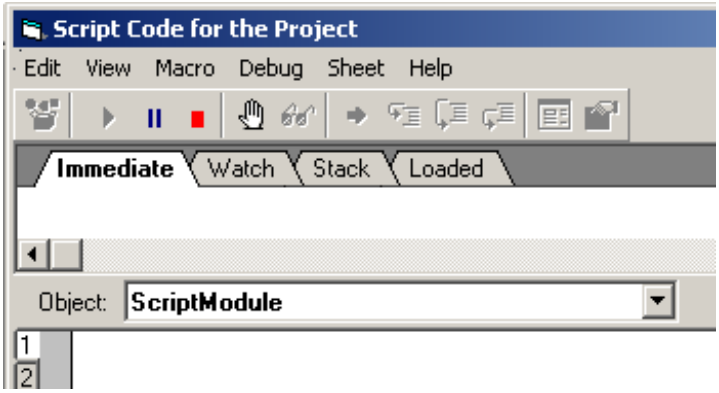


Figure 3-6: Sax Basic Tracing windows

Immediate

Similar to the Immediate window in Visual Basic. Used to type a line of code. Press ENTER RETURN to run it.

Query or change the value of a variable while running an application. While execution is halted, assign the variable a new value as you would in code.

Query or change a property value while running an application.

Call procedures as you would in code.

View debugging output while the program is running.

Watch

Similar to the Watch window in Visual Basic and used to view watch expression.
You can edit a value and then press ENTER, the UP ARROW key, the DOWN ARROW key, TAB, SHIFT+TAB, or click somewhere on the screen to validate the change. If the value is illegal, the Edit field remains active and the value is highlighted. A message box describing the error also appears. Cancel a change by pressing ESC.

Stack

Shows the already processed function calls.

Loaded

Shows all loaded scripts up to that point

A qualified debugging of the script works only when *Oracle Forms Recognition* is running. Therefore, it is useful to set breakpoints. As in Visual Basic, click left of the code windows to toggle a breakpoint.

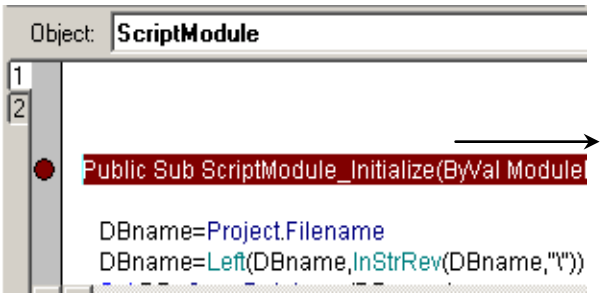


Figure 3-7: Section of the code window

In the Debug Mode, the script is still running. To edit the script, stop the execution first by clicking the Stop button .

3.2 Sequence of Events

In general, there are three types of script events:

ScriptModule

These are all events that are document class-independent, for example, the *ScriptModule_Initialize-/Terminate* event and the *ScriptModule_ExportDocument* event. For Details, refer to Section ***4.1, ScriptModule***.

Document

The script for the document class provides a sequence of Document-dependent events, such as the *Document_Validate* event. For details, refer to Section ***4.2, Document***. Those events can be customized for each document class, the parent, and all derived document classes.

When the *Document_Validate* event that is defined for the parent class and for the derived document class, and a document is assigned to the derived document class, then the event will be fired twice – once for the parent class and once for the derived document class.

On the other hand, a derived document class inherits the parent's script. This means that a *Document_Validate* event that is defined for the parent class occurs when the document is assigned to the derived decument class.

<Fields>

There are events for each field of a document class, for example *TotalPrice_Validate* event. For details, refer to Chapter ***4.3***, <Fields$_n$>. <Fields> is a substitute for the correct name of the event that depends on the field name, defined for a document class. Similar to the Document events, a derived document class inherits the events from the parent document class.

To customize the ScriptModule events, change to the first sheet in the script editor, select the *ScriptModule* from the objects' list, and from Proc drop down menu, the relevant event. To edit Document- or <Fields>- events select the relevant sheet for the document class. If the sheet is not yet available, change to the Definition Mode, select the document class from the Classtree view and select *Show Script* from the context menu or press F12 to open the script editor again.

3.2.1 Global Variables in Script

You can use the objects of all referenced libraries, but only those objects are documented within the scripting reference that should be used from script. Some of the objects also contain a list of methods and properties that must not be used. These methods are for integration purposes only and may cause undefined behavior if called from script.

To use the objects, a variable must be declared before the functionality can be used. There are some objects that are declared as global variables and can therefore be used in all script sheets. Currently, the following variables can be used without further declaration:

- **ScriptModule**: You can use this variable to retrieve the name of the current application and to execute OCR manually. For details, see also ***SCBCdrScriptModule***.
- **Project**: You can use this variable to have full access to the currently loaded project, for example to get information about the class tree and the defined document classes and other project specific information. See also ***SCBCdrProject***.
- **Settings**: You can use this variable to store information, which will change at different installation sites, such as absolute path names, database references or similar information.

3.2.2 Script Inheritance Chronology

The script event inheritance chronology describes how and in which order the script events are fired when a document is processed. This inheritance chronology is important when a document is classified to a document class that is derived from one or more parent classes. In this case, the script events for this current document class will be executed, as will the events for all parent document classes.

Inheritance Chronology of Script Events:

Document class Parent1
→ Document class Parent2
→…
→ last derived document class (senior derived class)

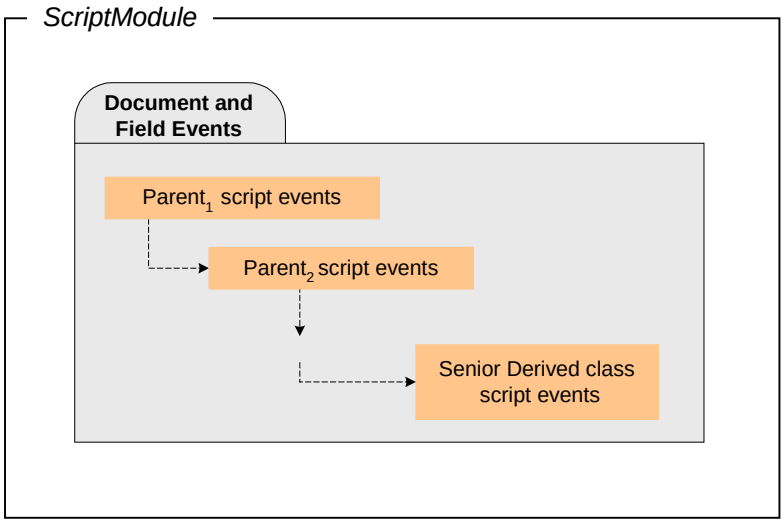


Figure 3-8: Script inheritance chronology for script events

Example:
A Project that contains the following document classes:

Parent class Invoices contains two derived classes: foreign invoices and inland invoices. The document class (DocClass) inland invoices itself is a parent class for the DocClass Smith, that contains all documents that are invoices of the company Smith. For example, for the validation of a document that is classified for the DocClass Smith, the validation process will be executed in the way that first the event Document_Validate for the parent class Invoices will be executed. In Script the variable pValid will be set. Then the Document_Validate event for the DocClass inland invoices will be performed and again pValid is set. At last the Document_Validate event for the DocClass Smith will be transacted and pValid can be set to a final value.

Similar to the Document_Validate event, all other script events are transacted. Exceptions to this rule are the ScriptModule-Events, the <Fieldn>_SmartIndex, Document_OnAction and the Document_FocusChanged, which will be fired only once.

The above described inheritance chronology is default for all Projects from Version 2.00. For the latest release, users can distinguish between two methods of processing:

- Top-down (From parent to derived class)
  or
- Bottom-up (From derived class to parent)

For **Top-down**, all events follow the same chronology. First, the events are fired for the parent document class and so on until the derived class, in which the document is classified.

For Bottom-up, document events are processed in a reversed order: first the document events of the class, in which the document is classified in, followed by the document events for all parent classes. The chronology for all other events is not be changed.

Only if you need to run script code from older projects for which the default inheritance chronology has to be changed, select the project settings – Runtime Mode tab. Switch the radio button for the Inheritance Chronology – Event sequence from Top-down to Bottom up. For all other projects you should not change the default settings.

For the above described example, this will mean that first the Document_Validate event for the DocClass *Smith* is fired, followed by the Document_Validate for the DocClass *inland voices* and last, the Document_Validate for *Invoices.*

Only the following events are effected by this reverse script chronology:

- Document_Validate
- Document_PreExtract
- Document_PostExtract
- <Fieldn>_Format
- <Fieldn>_PostAnalysis
- <Fieldn>_PostEvaluate
- <Fieldn>_PreExtract
- <Fieldn>_ValidateCell
- <Fieldn>_ValidateRow
- <Fieldn>_ValidateTable

By default, the inheritance chronology is now set to Top-down. For projects created in Oracle Forms Recognition Designer 1.31 and earlier, the default chronology was set to Bottom-up. If you want to use older projects, you must adjust the settings in the Runtime mode tab to Bottom-up.

3.2.3 Batch Processing

Batch processing starts with an initialization, runs through each document, and ends with the Termination event. (See ***Figure 3-9***) Classification, Extraction & Validation and Export events are fired when they are selected in the settings.

3.2.3.1 Classification

The PreClassify event is called before any defined classification method is executed. During this event, it is possible to apply an existing name of a document class to the Workdoc. In this case, all defined classification methods are skipped and the assigned document class is used as the result of the classification.

The PostClassify event is called after all defined classification methods are executed. During this event, it is possible to validate the classification result and if necessary, change the result to another DocClass.

3.2.3.2 Extraction and Validation

The **PreExtract** (**PostExtract**) event is called before (or after) any defined analysis or evaluation method is executed by the document class. During this event, it is possible to assign results to one or

more fields of the document and by changing the field state to skip defined analysis and evaluation methods.

The **Document_Validate** event can be used to perform validation on a document level. At this point, the validation of all fields was performed successfully. That means that all fields have a valid state. During the Document_Validate event, it is possible to implement validation rules combining several fields. When the document returned an invalid state within this event, a message box will pop up, to tell the user that the document is not valid.

Field events are specific for each field of each document class. Field events appear within the script sheet of their document class. This means that all events for the field “Number” of the document class “Invoice” are available within the script sheet of the DocClass “Invoice.”

Within the script, the name of the fields will appear as a specifier for the field. This means the validate event for the field “Number” will appear as method “Number_Validate.”

The **Field_Validate** event can be used to perform project-specific validation rules. Use the pValid parameter to return the validation decision. The default initialization of pValid is TRUE, so if the parameter remains unchanged or if the event is not implemented, the document state becomes valid.

From Version 2.00, the following procedure is implemented for the extraction of tables. If an extraction engine could not extract the table, the manual verification shows an empty table containing one row with each of its cells as ‘invalid’ (red). To validate the document the table has to be edited. To skip the validation of this table field, the script can be used. However, it is not enough just to set Table, TableValid to True. First of all, it is absolutely necessary to set Valid property of the field that might contain a table (as a parent of table object). The other part of the issue is that the system always attempts to create one invalid table row, so that a *Verifier* user can start editing tables, even if they were not extracted at all. To skip the validation, you have to set the all fields of this first row to True . Below you can find the sample of script code that helps to skip table data validation in*Verifier* (for a table with two columns):

You can use this code in Document_Validate and (or) in Document_FocusChanged events.

```
Dim theEmptyTable As SCBCdrPROJLib.SCBCdrTable
Dim theEmptyTableField As SCBCdrPROJLib.SCBCdrField

   ' Initializes table and filed references
   Set theEmptyTable =
pWorkdoc.Fields("EmptyTable").Table(pWorkdoc.Fields("EmptyTable").ActiveTableIndex)
   Set theEmptyTableField = pWorkdoc.Fields("EmptyTable")

   ' Makes table object valid
   theEmptyTable.CellValid(0,0) = True
   theEmptyTable.CellValid(1,0) = True
   theEmptyTable.RowValid(0) = True
   theEmptyTable.TableValid = True

   ' Makes table field valid (table object is a part of more generic field object)
   theEmptyTableField.Valid = True
   theEmptyTableField.Changed = False

   ' Releases references
   Set theEmptyTable = Nothing
   Set theEmptyTableField = Nothing
```

The new Table Extraction engine introduces a new aspect for the scripts validation and/or extraction correction. Scripts should now consider a new case when a table cell has multiple extracted lines. To do this, the system administrator has to use a new “RowNumber” property of *Oracle Forms Recognition* table object that returns actual row number for every table line extracted via Table Extraction engine. Below you also find a simple script sample:

```
Private Sub Tabelle_ValidateCell(pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As_
SCBCdrPROJLib.SCBCdrWorkdoc, ByVal Row As Long, ByVal Column As Long, pValid As Boolean)
   Dim nCurrentRow, nRow, nLine As Integer
   Dim strCellText As String

   nCurrentRow = pTable.RowNumber(Row)
   strCellText = pTable.CellText(Column, Row)

   nLine = Row - 1
   nRow = nCurrentRow
   While (nLine >= 0) And (nRow = nCurrentRow)
      nRow = pTable.RowNumber(nLine)
      If (nRow = nCurrentRow) And (pTable.CellText(Column, nLine) <> "") Then
         strCellText = pTable.CellText(Column, nLine) + " " + strCellText
      End If
      nLine = nLine - 1
   Wend

   nLine = Row + 1
   nRow = nCurrentRow
   While (nLine < pTable.RowCount) And (nRow = nCurrentRow)
      nRow = pTable.RowNumber(nLine)
      If (nRow = nCurrentRow) And (pTable.CellText(Column, nLine) <> "") Then
         strCellText = strCellText + " " + pTable.CellText(Column, nLine)
      End If
      nLine = nLine + 1
   Wend

   If strCellText = "" Then
      pValid = False
   Else
      pValid = True
   End If
End Sub
```

The new automated field formatting, *<Field$_n$>_FormatForExport*, occurs after firing of *<Field$_n$>_Validate* event, in case the field is completely valid, see Figure 3-10: Extraction and Validation events. The results of formatting can be then accessed in “Export” event of *Oracle Forms Recognition* script, using a new “FormattedText” property of *Oracle Forms Recognition* field object (the former “Text” property will still store unformatted text, which could be accessed as well).

3.2.3.3 Pre-Validation

There is another aspect that can influence the validation of a field: the pre-validation. This will be supported by the selected OCR engine that identifies uncertain OCR results, a minimum length, and a maximum length. The options *min. length* and *max. length* cannot be used for table fields. By default, the option *No Rejects* is set for a field within the properties, see **Chapter 7 of the Oracle Forms Recognition User Guide.** If the pre-validation results should not influence the field validation, select ‘Always valid.’

For both normal fields and tables, it is important to know that this pre-validation can be easily and silently discarded by validation script. If there is no validation script defined, the pre-validation will always work without any problems. This is not necessarily the case in a production environment. Therefore consider the following script examples:

```
1. WRONG, discards any pre-validation:
        If  IsInvoiceNumberValid(strInvoiceNumber)  Then
                pValid = True
        End If

2. WRONG, discards any pre-validation:
        pValid = IsInvoiceNumberValid(strInvoiceNumber)
```

In some cases, it makes sense to overwrite the results of the standard-validation. For example, if you use a crosswise validation of fields in script, you can overwrite the validation result:

```
TotalNet + TotalVat = TotalGross
```

and the OCR result of some characters of TotalNet is unsure, then you can set the valid-properties of all fields to valid. In this case, the unsure characters remain red but the fields are valid.

In all other cases it is recommended to use the recommended statement:

```
3. CORRECT, takes into account pre-validation results:
        If  Not  IsInvoiceNumberValid(strInvoiceNumber)  Then
                pValid = False
        End If
```

In other words, when it is important to take into account the pre-validation results, "pValid" must **never** be set to True. Moreover, a system administrator must know that code like pField.Valid = True may discard pre-validation results.

The hints above are also relevant to another pre-validation step for tables, which is called *Entry required* and can be set via "*Designer*"->"Definition Mode"->"Fields"->"Select a table field"->"Show field properties"->"Analysis property page"->"Insert/Delete columns" .

3.2.3.4 Export

The Export event provides the possibility to implement a customer-specific export of all extracted data. With the Export event, it is possible to write export files in any format or to connect a database or workflow system and write the data directly to the next system.

3.2.4 500 kb Per Script Sheet Limitation

SAX Basic includes a default limitation of 512 kb on the size of each script sheet. This includes both code and comments. One solution for getting around this problem is to optimize a script by, for example, removing any functions which are not used. (It is not recommended that one remove all comments.)

The limit on the size of each script sheet can be increased by increasing the application call stack size. To increase the limit to 1 MB per script sheet do this:

Open a text file. Type in the following on a single line:

```
editbin.exe /stack:0x200000,0x200000 DstVer.exe
```

Save the text file as “Increase Verifier Stack Size.bat”

Place this batch file into your application folder. (\Program Files\Oracle\Forms Recognition)

Double click the batch file to run it.

Note: You will need the editbin system utility in order to run this batch file.

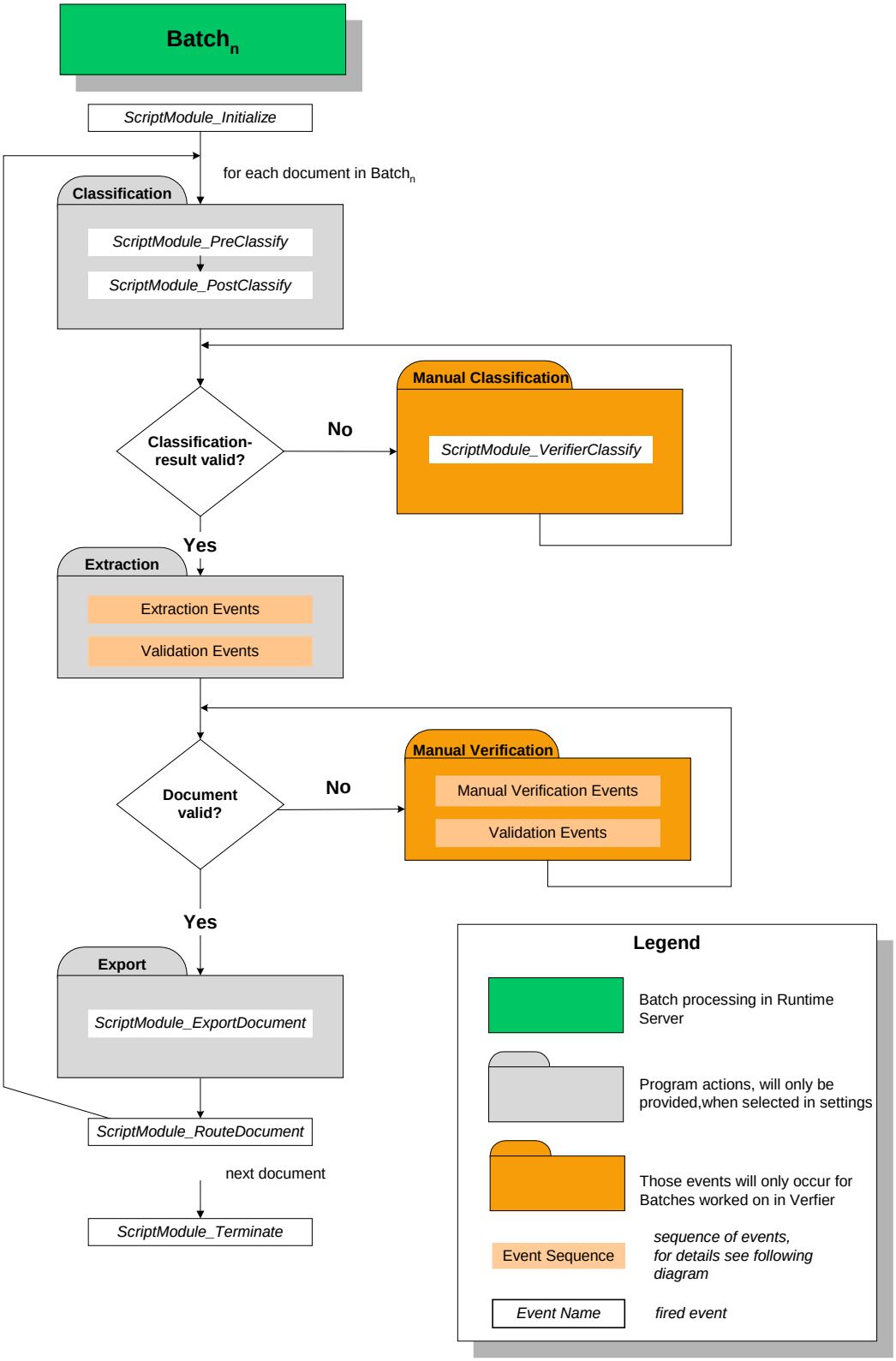


Figure 3-9: Event chronology for Batch processing

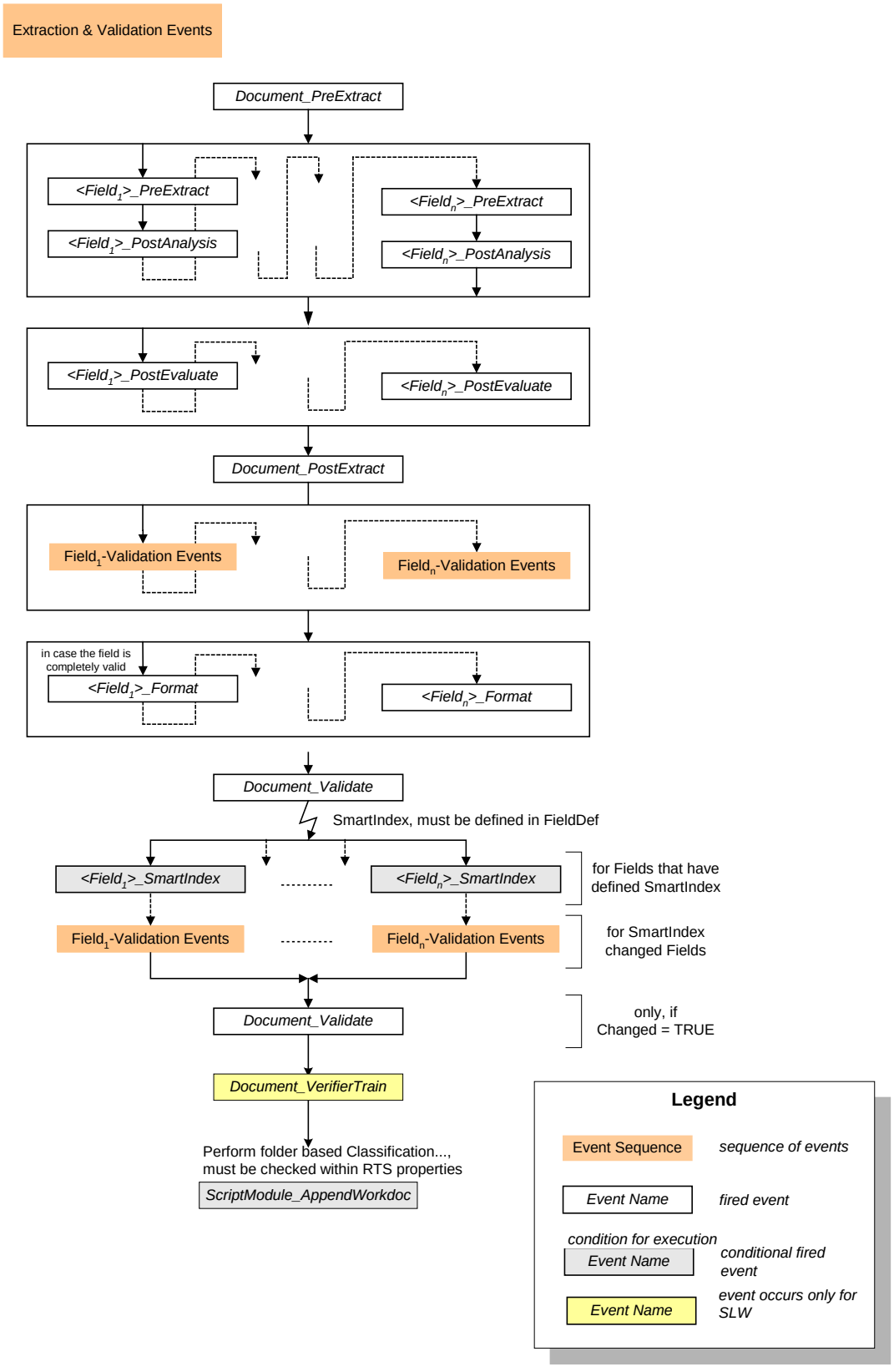


Figure 3-10: Extraction and Validation events

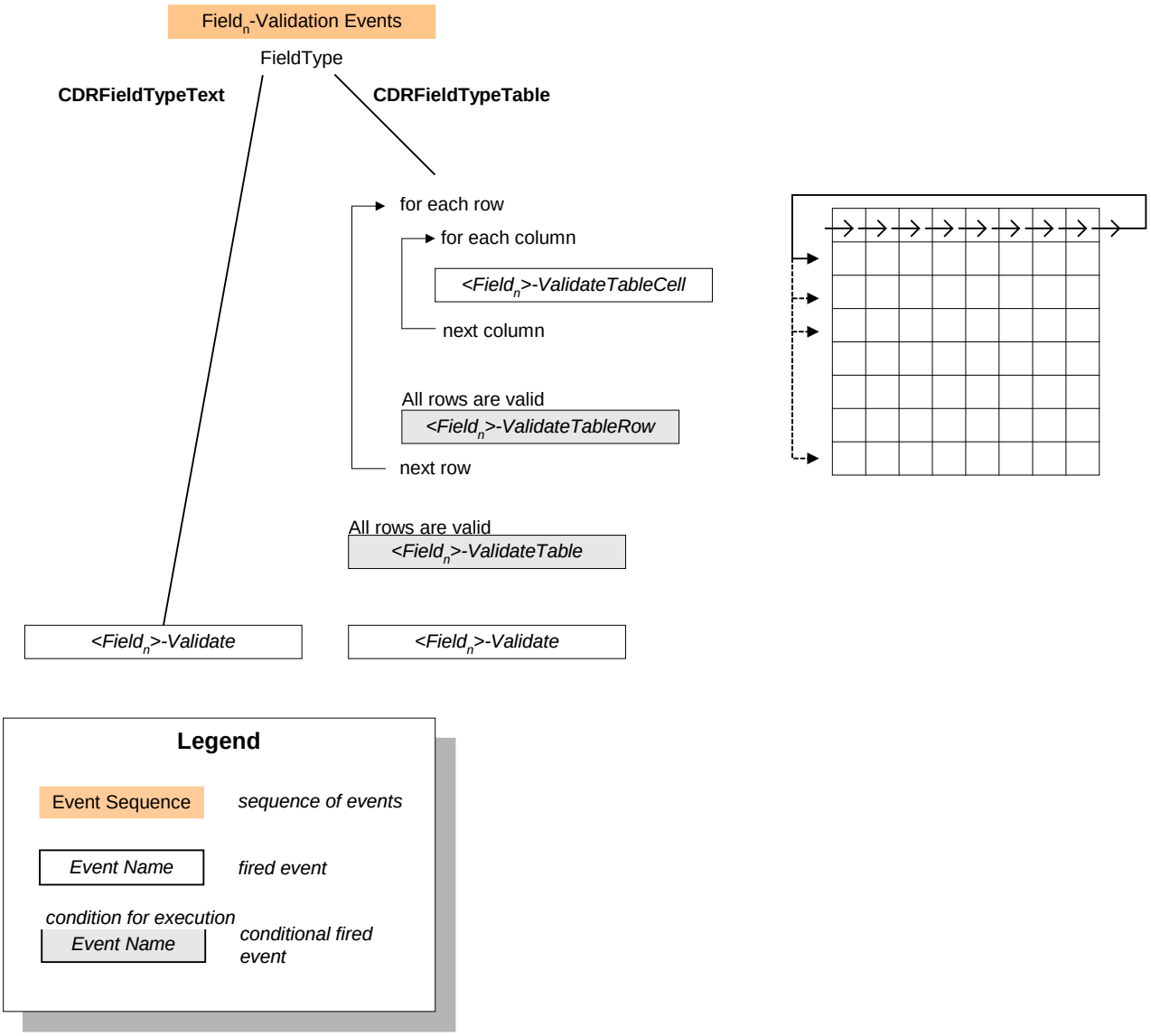


Figure 3-11: Field Validation events

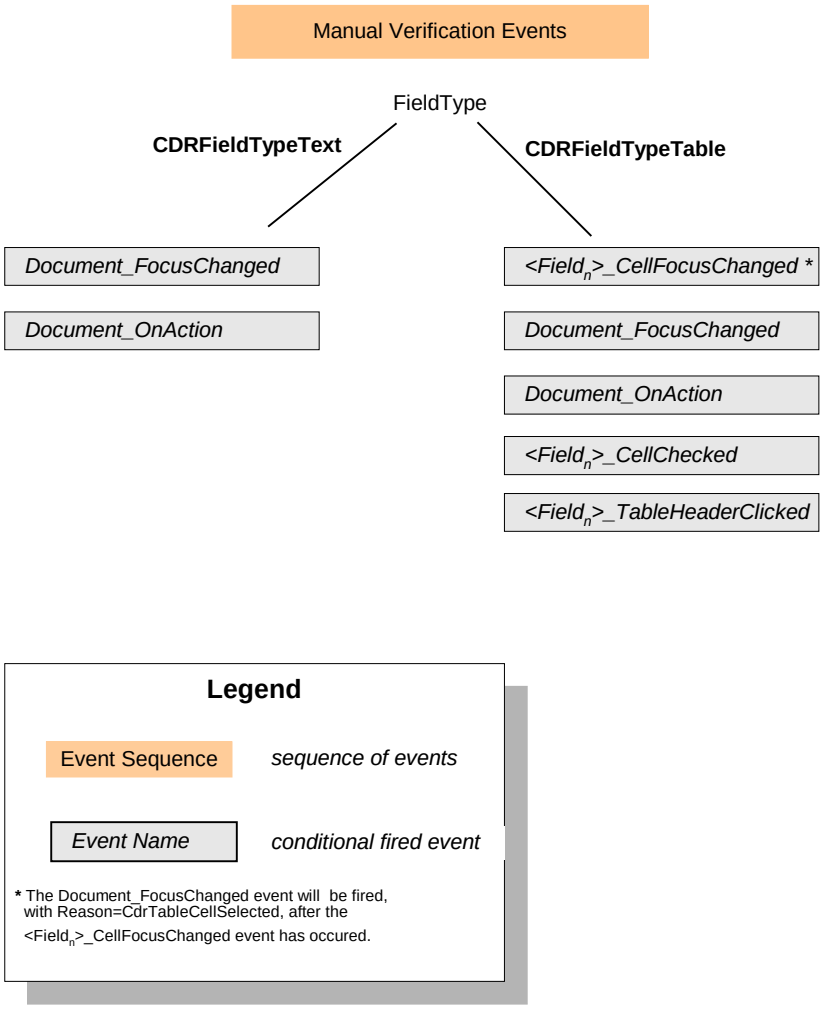


Figure 3-12: Manual Verification events

3.2.5 SCBCdrWorkdoc in Course of Batch Processing

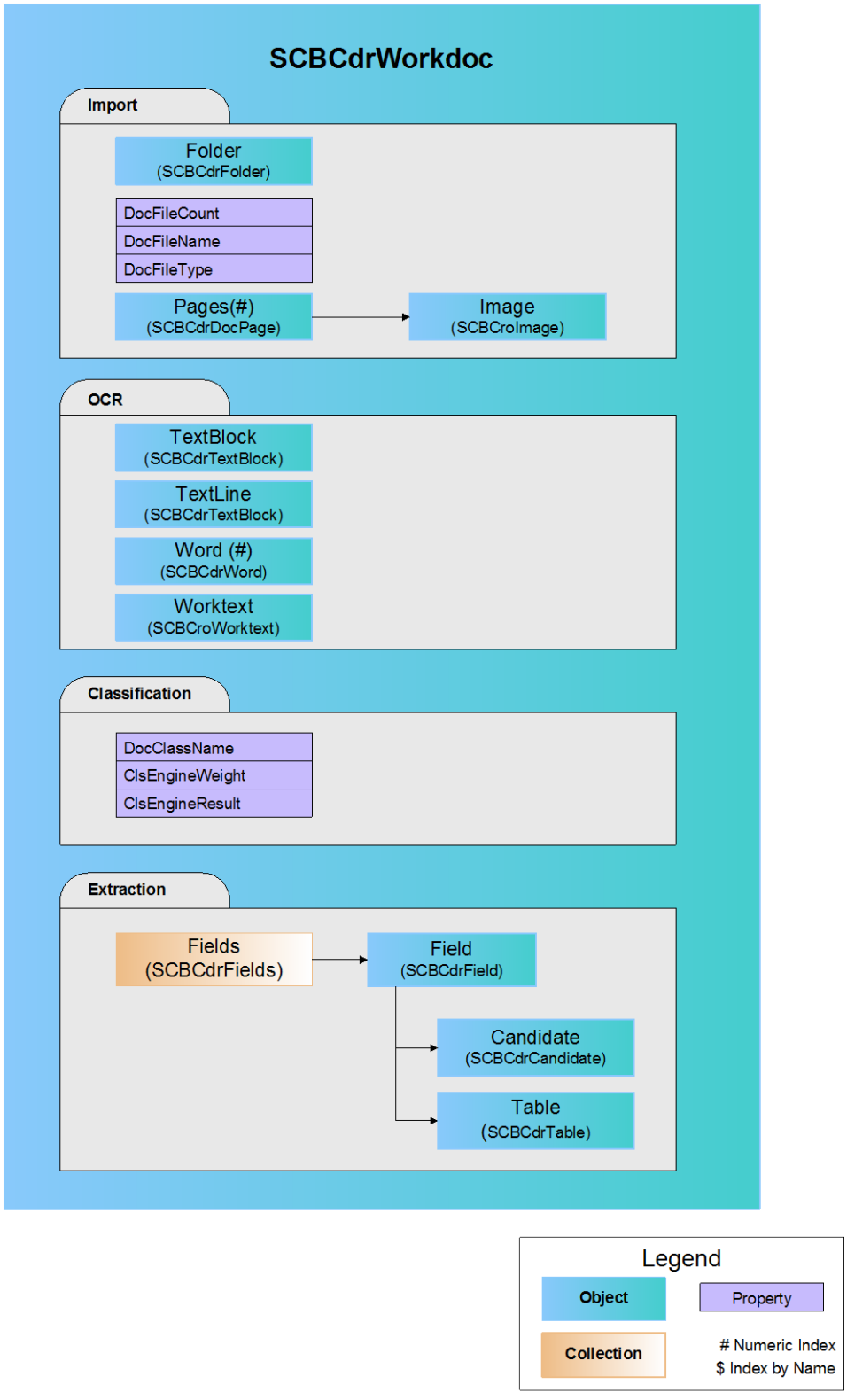


Figure 3-13: SCBCdrWorkdoc properties in course of batch processing

3.2.6 Public versus Private Variable Declaration

If you define a public variable on a parent-level script page, do not declare public variables in nested script sheets unless the variables will be defined as local variables in local functions or procedures.

If a public user type is defined on a parent-level script page and a public variable is declared on any child level as private or public, the SAX Basic Engine may crash when it unloads the script pages' code. This probably happens because SAX Basic first attempts to unload the parent-level sheet, destroying the user-defined type, and then attempts to unload the child-level sheet, trying to de-allocate private / public variables of the destroyed type. If the type definition is no longer available, the de-allocation will crash the host process.

3.3 Script Samples

When implementing script code, you must consider that not all information is distributable right from the beginning. For example, the name of the current project and the defined document classes are directly known, but the information about which document class a document belongs to is not available before the classification step is processed. Therefore, it is necessary to know which processing step will provide which information. See also Section ***3.2.5***.

Besides that, the chronology of the events is of particular importance. For details, see Chapter 3.2.3 Batch Processing, page 15. A <Fields>_Validate event such as TotalPrice_Validate is processed before the Document_Validate. When a <Fields>_Validate returns a status of invalid for a field, the status for the document will also be invalid.

The following example is conceivable for script programming: the field TotalPrice for an Invoices document class has an invalid status as the field could not be extracted from the document after classification. To validate the document although automatically, the Document_Validate could check, if the fields SinglePrice and Quantity have been extracted, assign the result of SinglePrice * Quantity to the field TotalPrice and then the status of the document to valid.

Most of the events of the document class - and field events are processed in a "Top-down" sequence, but there some exceptions that can be defined as "Bottom-up". For more details refer to ***Script Inheritance Chronology***.

3.3.1 Reporting Procedures

The following section describes three straightforward examples illustrates how script programming can be used for reporting. For simplicity reason the results are written to semicolon-delimited text files, on the C:\ drive. In a production system you should check the path where the file should be saved to and if necessary create a special directory to which the result files are saved. Currently the text file will be rewritten for each processed batch. Although the text file can be used as import for a database or directly used for an analysis tool such as Microsoft Excel, it is also conceivable to write the result directly to a database.

For clarity, the source code that is on the script sheet of the project has a light blue frame. The source code inserted to the sheet for the document classes has a yellow frame.

3.3.1.1 Write Text File Containing Project Information

This first example writes some project information to the text file C:\project.txt. It returns the name of the computer, the process ID and the name of the application and generates a class hierarchy of the document classes (*) and the number of defined fields and a list of the fields (→), that are defined for a project.

For example:

```
Computer : 'Computer-XXX'

ProcessId : '1480'

Application : 'Verifier'

* Invoices (2 Fields)
-> VendorName (1)
-> InvoiceNumber (2)
  * FRENCH ELLISON 35307 (2 Fields)
  -> VendorName (1)
  -> InvoiceNumber (2)
  * HTS 22153 (2 Fields)
  -> VendorName (1)
  -> InvoiceNumber (2)
  * 180334 CONSTRUCTION SERVICES (2 Fields)
  -> VendorName (1)
  -> InvoiceNumber (2)
  * 236043 UNIMIN (2 Fields)
  -> VendorName (1)
  -> InvoiceNumber (2)
```

The code for the above generated output must be inserted on the script sheet for the project. The **ScriptModule_Initialize** event was used to start the export:

```
Option Explicit
'  Cedar Module Script
' Script code for the project

Public Declare Function GetCurrentProcessId Lib "kernel32" () As Long
Public Declare Function apiComputerName Lib "kernel32" Alias "GetComputerNameA" (ByVal lpBuffer As
String, nSize As Long) As Long

Private Sub ScriptModule_Initialize(ByVal ModuleName As String)
' The initialize event is used to generate the text files

   Dim FNum As Integer
   Dim Classes As SCBCdrDocClasses

   On Error GoTo ErrHandler

   ' get free channel to write name of computer, processId
   ' and application name to the textfile
   FNum = FreeFile
   Open "c:\project.txt" For Output As #Fnum
   ' call priv. function GetMachineName to get the name of the computer
   Print #FNum, "Computer : '" & GetMachineName & "'"
   Print #FNum, ""
   ' call syst. function to get the processId
```

```
   Print #FNum, "ProcessId : '" & GetCurrentProcessId & "'"
   Print #FNum, ""
   ' write Modulename
   Print #FNum, "Application : '" & ModuleName & "'"
   Print #FNum, ""
   Set Classes = Project.BaseClasses
   ' call priv. sub Write Classes to output class hierarchy and fields
   WriteClasses Classes, FNum, 0
   Close FNum
Exit Sub

' Error handling
ErrHandler:
   MsgBox Err.Description
End Sub
```

```
Private Function GetMachineName() As String
'Returns the computername
   Dim lngLen As Long, lngX As Long
   Dim strCompName As String

    lngLen = 16
    strCompName = String$(lngLen, 0)
    lngX = apiComputerName(strCompName, lngLen)

    If lngX <> 0 Then
        GetMachineName = Left$(strCompName, lngLen)
    Else
        GetMachineName = ""
    End If

End Function
```

```
Private Sub WriteClasses(Classes As SCBCdrDocClasses, FNum As Integer, Level As Integer)
' Writes the document classes, no. of fields and their fields to C:\project.txt
Dim Class As SCBCdrDocClass
Dim c As Integer
Dim Fields As SCBCdrFieldDefs
Dim Field As SCBCdrFieldDef
Dim f As Integer
Dim LineStr As String
Dim zk As String
Dim l As Integer

zk = ""
For l = 1 To Level
   zk = zk & "  "
Next l

Level = Level + 1

For c = 1 To Classes.Count
   Set Class = Classes.Item(c)
   Set Fields = Class.Fields
   LineStr = zk & "* " & Class.Name & " (" & Fields.Count & " Fields)"
   Print #FNum, LineStr
   For f = 1 To Fields.Count
      Set Field = Fields.Item(f)
      Print #FNum, zk & "-> " & Field.Name & " (" & Field.FieldID & ")"
   Next f
```

```
   WriteClasses Class.DerivedDocClasses, FNum, Level
Next c

End Sub
```

3.3.1.2 Write Text File Containing Classification Results

This example is made to write some document information a text file, C:\ClassifyAnalysis.txt. It is implemented in a way that the text file will be created and all processed batch respectively all documents are appended to the text file. If you want to write the result only for one batch or for example for the batch per day, then you have to take care that several files are created.

The text file is built in a way that for each document a line is written with the name of the batch, the document belongs to, the document name , the name of document class, it was assigned to, and a status specifying whether or not the document was classified or not:

```
BatchName;Documentname;ClassName;Classifystatus
0000003;c:\projekte\demo2\batches\00000003\00000291.wdc;Rab;classified
0000003;c:\projekte\demo2\batches\00000003\00000292.wdc;Xodex;classified
0000003;c:\projekte\demo2\batches\00000003\00000293.wdc;Contac;classified
0000003;c:\projekte\demo2\batches\00000003\00000294.wdc;Contac;classified
0000003;c:\projekte\demo2\batches\00000003\00000295.wdc;Xodex;classified
:
:
```

In the sample the analysis was made with pivot tables of Microsoft Excel that allows to apply multiple representations of the same data. Therefore, the text file was used as import for the pivot table. As one possible output, a report was made that gives an overview of the processed sequence of batches (0000000; 0000001; 0000003) and to which document classes the documents were assigned to.

Batch Name	Allrauer	Bash	Computer 2001	Contac	Drv	Rab	Rubin	Schulz	Stracke	Xodex	TotalResult
0000000	1			3		2				4	10
0000001	1	1		2	2	1				3	10
0000003	10	9	10	10	10	10	10	10	8	10	97
Total Result	12	10	10	15	12	13	10	10	8	17	117

Table 3-1: Document class distribution for three processed batches

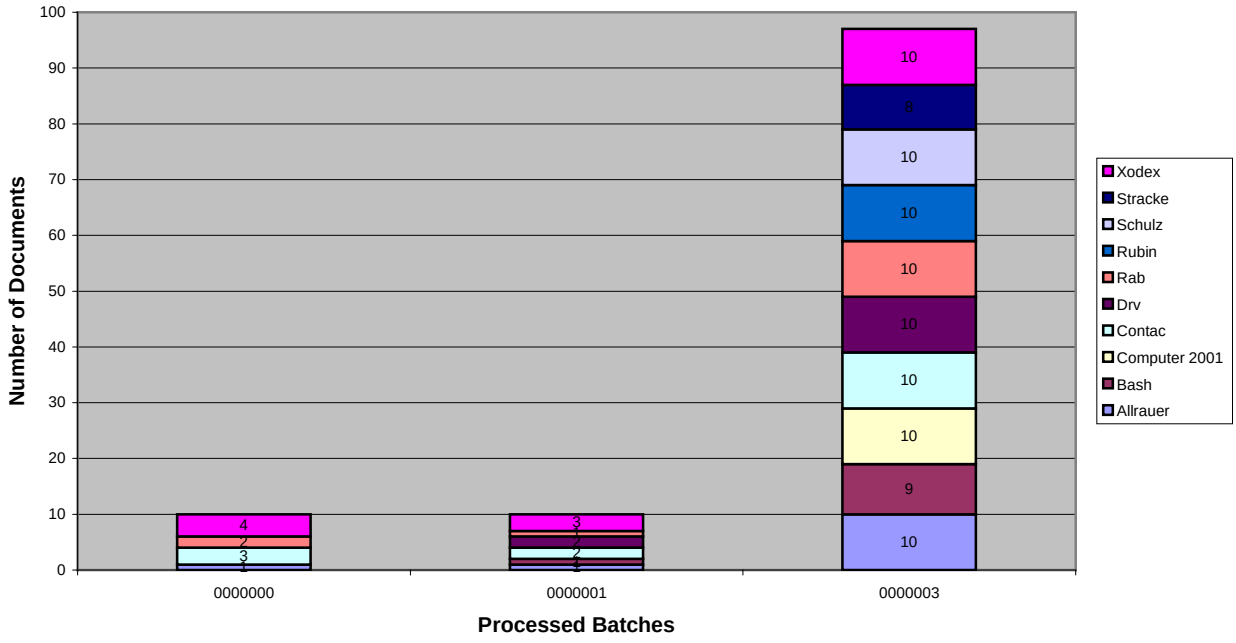


Diagram 3-1: Bar chart of the document class distribution for the processed batches

The code for the above generated output has to be inserted on the script sheet for the project. The ScriptModule_Initialize event was used to create the text file. The ScriptModule_PostClassify event is used to write the document information to the text file:

```
Option Explicit
'  Cedar Module Script
' Script code for the project

Private Sub ScriptModule_Initialize(ByVal ModuleName As String)
' The initialize event is used to generate C:\ClassifyAnalysis.txt
Dim FNum As Integer
Dim FSO As New FileSystemObject

   On Error GoTo ErrHandler

   ' get free channel and create C:\ClassifyAnalysis.txt
   FNum = FreeFile
   If Not(FSO.FileExists (C:\ClassifyAnalysis.txt)) Then
      Open "C:\ClassifyAnalysis.txt" For Output As #FNum
      Print #FNum,"BatchName;Documentname;ClassName;Classifystatus"
      Close #Fnum
   End If
   Exit Sub

' Error handling
ErrHandler:
   MsgBox Err.Description
End Sub
```

```
Private Sub ScriptModule_PostClassify(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)
' Appends the document information to the text file
```

```
'
Dim FNum As Integer
Dim ResultStr As String

On Error GoTo errhandler

   ' Check if Workdoc of document is classified, then the classifystatus
   ' is set either to “classified” or “not classified”
   ResultStr = "not classified"
   If (pWorkdoc.DocClassName <> "") And (pWorkdoc.DocClassName <> Project.DefaultClassifyResult) Then
ResultStr = "classified"

   ' Write results to text file
   FNum = FreeFile
   Open "C:\ClassifyAnalysis.txt" For Append As #FNum
   ' call priv. Func. To get the batch name as part of the file name
   Print #FNum, GetBatchName(pWorkdoc.Filename) & ";" & pWorkdoc.Filename & _ ";" &
pWorkdoc.DocClassName & ";" & ResultStr
   Close #FNum
Exit Sub

ErrHandler:
   MsgBox Err.Description
End Sub
```

```
Private Function GetBatchName (WorkdocName As String) As String
Dim FirstPart As String

On Error GoTo ErrHandler

   FirstPart= Left(WorkdocName, (InStrRev(WorkdocName, "\")-1))
   GetBatchName=Right(FirstPart,(Len(FirstPart) - InStrRev(FirstPart,"\")-1))
   Exit Function

ErrHandler:
   GetBatchName=""
   MsgBox Err.Description
End Function
```

3.3.1.3 Write Text File Containing Extraction Results

The last reporting example examines the extraction results for the processed sequence of batches. Again a text file, C:\ExtractionAnalysis.txt is written; one line field of a document, containing the Batch name, the document name , the name of document class it was assigned to, the field name, the field value, and the number of rejects:

```
BatchName;Documentname;DocumentClass;FieldName;FieldValue;ValidStatus;NumberOfRejects
0000003;c:\projekte\demo2\batches\00000003\00000291.wdc;Rab;Rechnungsnummer;183408;True;0
0000003;c:\projekte\demo2\batches\00000003\00000291.wdc;Rab;Bestellnummer;0000080;True;0
0000003;c:\projekte\demo2\batches\00000003\00000291.wdc;Rab;Netto;154200;True;0
0000003;c:\projekte\demo2\batches\00000003\00000291.wdc;Rab;MWSt;24672;True;0
0000003;c:\projekte\demo2\batches\00000003\00000291.wdc;Rab;Brutto;178872;True;0
:
```

Again the analysis was made with pivot tables Microsoft Excel and the text file was used as import data. As one possible output a report was made that gives show the distribution of fields with an invalid / valid field status per document class. Table 3-2, shows that for the document class Allrauer 72

documents were classified and only one field had an invalid field status. There were only two further document classes that contained fields with an invalid field status that are Bash and Xodex. To visualize the same details a chart was included as well; see ***Figure*** 3-14.

DocumentClass	False	True	Number of Documents
Allrauer	1	71	72
Bash	5	55	60
Computer 2001		60	60
Contac		90	90
Drv		72	72
Rab		78	78
Rubin		60	60
Schulz		60	60
Stracke		48	48
Xodex	4	98	102
No. of documents	10	692	702

Table 3-2: Pivot table of the processed batches to visualize the number of invalid fields

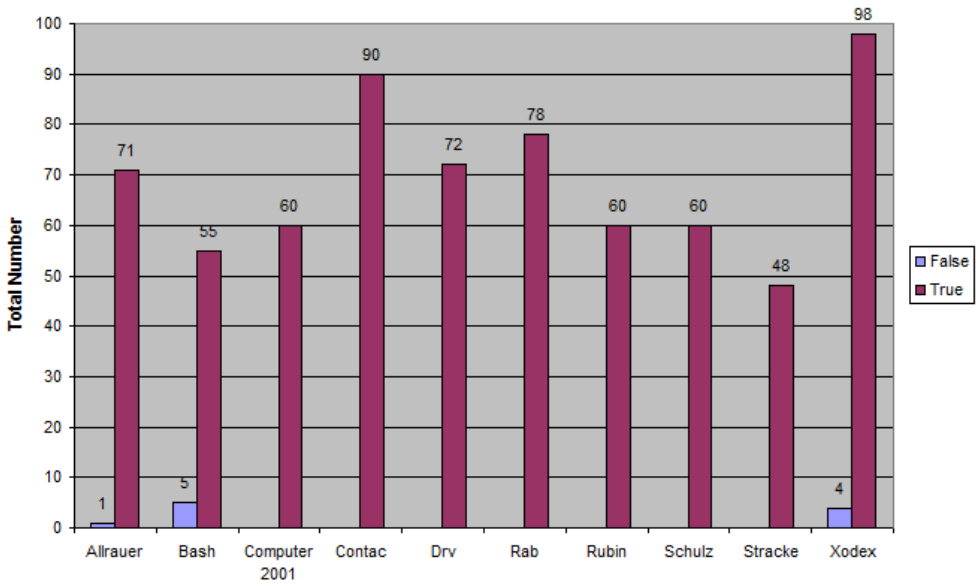


Figure 3-14: Pivot chart of to visualize the number of invalid fields

Another representation of the data is to import the text file to a Microsoft Excel worksheet and to apply an automatic filter. Then, for example, the data can be sorted to list all fields with an invalid field status, as shown in ***Figure 3-15***. Now you can easily find the fields out that were not extracted correctly.

	A	B	C	D	E	F	G	H
1	BatchNam	Documentname	DocumentClas	FieldNam	FieldVa	ValidSta	Num	rOfRejects
103	0000003	c:\projekte\demo2\batches\00000003\00000309.wdc	Bash	Brutto		False	0	
108	0000003	c:\projekte\demo2\batches\00000003\00000310.wdc	Xodex	MWSt		False	0	
126	0000003	c:\projekte\demo2\batches\00000003\00000314.wdc	Xodex	MWSt		False	0	
139	0000003	c:\projekte\demo2\batches\00000003\00000316.wdc	Bash	Brutto		False	0	
343	0000003	c:\projekte\demo2\batches\00000003\00000350.wdc	Bash	Brutto		False	0	
438	0000003	c:\projekte\demo2\batches\00000003\00000366.wdc	Xodex	MWSt		False	0	
469	0000003	c:\projekte\demo2\batches\00000003\00000371.wdc	Bash	Brutto		False	0	
516	0000003	c:\projekte\demo2\batches\00000003\00000379.wdc	Allrauer	MWSt		False	0	
685	0000001	c:\projekte\demo2\batches\00000001\00000309.wdc	Bash	Brutto		False	0	
690	0000001	c:\projekte\demo2\batches\00000001\00000310.wdc	Xodex	MWSt		False	0	

Figure 3-15: Excel table extraction data filtered for invalid field status

The code for the above generated output must be inserted on the script sheet for the project. The ScriptModule_Initialize event was used to create the text file. The public procedure WriteStatistic appends the field information to the text file. The Document_PostExtract event is used to start the reporting. It has to be defined on the sheet of the document class. For which document classes, depends on the projects' architecture. Normally, it will be sufficient to define the call for WriteStatistic on the level of the base document classes, as derived document classes inherit the parents script. When the script code is inserted for each base class, it will also be fired when a document is classified to a derived document class. If you insert the script code for the base class and the derived document class, the events will occur twice for the same document and the statistic is wrong.

```
Private Sub Document_PostExtract(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)
   ' call public sub to write the statistic
   WriteStatistic pWorkdoc
End Sub
```

```
Option Explicit
'  Cedar Module Script
' Script code for the project

Private Sub ScriptModule_Initialize(ByVal ModuleName As String)
' The initialize event is used to generate C:\ExtractionAnalysis.txt
Dim FNum As Integer
Dim FSO As New FileSystemObject

   On Error GoTo ErrHandler

   ' get free channel and create C:\ExtractionAnalysis.txt
   FNum = FreeFile
   If Not(FSO.FileExists (C:\ExtractionAnalysis.txt)) Then
      Open "C:\ExtractionAnalysis.txt" For Output As #FNum
      Print #FNum,"BatchName;Documentname;DocumentClass;FieldName;FieldValue;
ValidStatus;NumberOfRejects"
      Close #Fnum
   End If
   Exit Sub

' Error handling
ErrHandler:
   MsgBox Err.Description
End Sub
```

```
' Statistic Report
```

```
Public Sub WriteStatistic(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)
'checks for all fields if they are empty fields or counts the number of rejects

Dim NORs As Long
Dim FNum As Integer
Dim FieldCounter As Integer
Dim pField As SCBCdrPROJLib.SCBCdrField
Dim BatchName As String

On Error GoTo errorhandling

'If InStr(UCase(ScriptModule.ModuleName),"SERVER")=0 Then Exit Sub
'Run this script only on Server

For FieldCounter=1 To pWorkdoc.Fields.Count
   Set pField=pWorkdoc.Fields(FieldCounter)

   ' call priv. function to return the no. of rejects for the field
   NORs = NumberOfRejects(pField)

   'Write results To Textfile
   FNum = FreeFile
   Open "C:\ExtractionAnalysis.txt" For Append As #FNum
   Print #FNum,GetBatchName(pWorkdoc.Filename) & ";" & pWorkdoc.Filename & ";" & pWorkdoc.DocClassName
& ";" & pField.Name & ";" & pField.Text & ";" & pField.Valid & ";" & NORs
   Close #FNum
Next

Exit Sub

'Error handling
errorhandling:
   MsgBox Err.Description
End Sub
```

```
Private Function NumberOfRejects(pField As SCBCdrField) As Long
' Returns the no. of rejects for this field
Dim FieldWorkText As SCBCroWorktext
Dim MyCharCounter As Integer
Dim Anz As Long

   Set FieldWorkText=pField.Worktext
   Anz=0
   For MyCharCounter=0 To FieldWorkText.TextLength-1
      If FieldWorkText.GetCharConfidence(MyCharCounter)<100 Then Anz=Anz+1
   Next
   NumberOfRejects = Anz
End Function
```

```
Private Function GetBatchName (WorkdocName As String) As String
'Cuts the name of the Batch out of the Workdoc name and returns it
Dim FirstPart As String

On Error GoTo ErrHandler

   FirstPart= Left(WorkdocName, (InStrRev(WorkdocName, "\")-1))
   GetBatchName=Right(FirstPart,(Len(FirstPart) - InStrRev(FirstPart,"\")-1))
   Exit Function
```

```
ErrHandler:
   GetBatchName=""
   MsgBox Err.Description
End Function
```

3.3.2 Hook to Loop Over Candidates and Change Their Weights or Remove Them

In script, you have access to the candidates. Use the “Post_Evaluate” event as a hook to loop over candidates and change their weights or remove them. For example, candidates could be excluded by Check Digit calculations or other complex format restrictions. Another way is to look up each candidate in the database and decide on the correct candidate according to the lookup result. The following script code gives an example for a loop over the candidates.

In the first step, candidates that are out of a certain geometry are excluded. The candidates with rejects are excluded and candidates satisfying a criteria (not specified here) are assigned a weight of 0.78:

```
Private Sub VNr_PostEvaluate(pField As SCBCdrPROJLib.SCBCdrField, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc)
  'Loop over all Candidates
   Dim i As Long
   Dim Criteria as Boolean
   Dim myImg As SCBCroImage
   Dim myCandidate As SCBCdrCandidate
   Dim xmax, ymax, x,y,h As Long

        Set myImg= pWorkdoc.Image(0)
        xmax= myImg.Width
        ymax= myImg.Height
        i = 0

        While i<pField.CandidateCount
              If(i<pField.CandidateCount) Then
              ' exclusion of candidates by geometry
                 Set myCandidate = pField.Candidate(i)
                 x= myCandidate.Left
                 y= myCandidate.Top

            ' remove candidates from an upper or lower band
            If (y > (ymax/5*4) Or y < (ymax/4)) Then
               pField.RemoveCandidate(i)
            ' remove candidates on the right side
            ElseIf (x> xmax/3*2) Then
               pField.RemoveCandidate(i)
            ' remove candidates containing rejects
            ElseIf InStr(1,pField.Candidate(i),"?") Then
               pField.RemoveCandidate(i)
             Else
              i = i+1
              ' ask for any criteria like database, check digit or similar
              Criteria = .....
                If Criteria Then
                 pField.Candidate(i).Weight= 0.78  ' Here the weight is changed
               End If
          End If
         End If       'i<pField.CandidateCount
```

```
        Wend
End Sub
```

Script evaluation can also be used in combination with evaluation to improve the result.

3.3.3 Show Supplier Information from the Vendor Pool

This script code example provides a list of candidates for the selected header field. To show those items a dialog will pop up, see ***Figure 3-16***. To search for vendor pool entries, you can insert text in the Search text field of this dialog and start a search in the vendor pool by pressing the Search button. The button Show Best, displays the best candidates and by pressing Item Details… the details stored in the vendor pool are shown for the selected candidate, see ***Figure 3-17***.

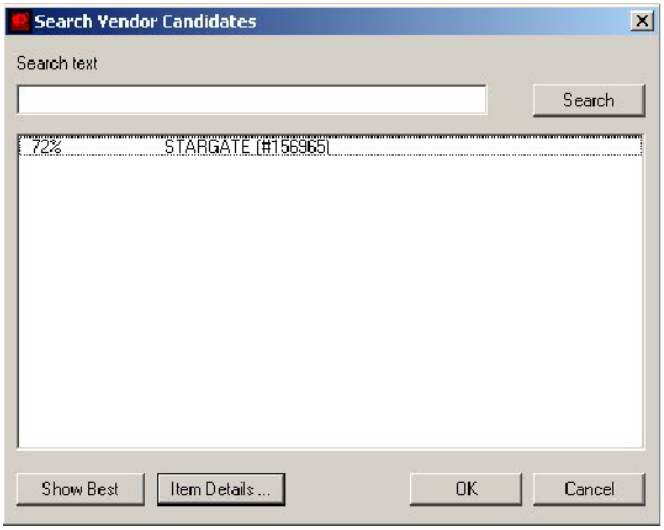


Figure 3-16: List of candidates for the header field VendorName

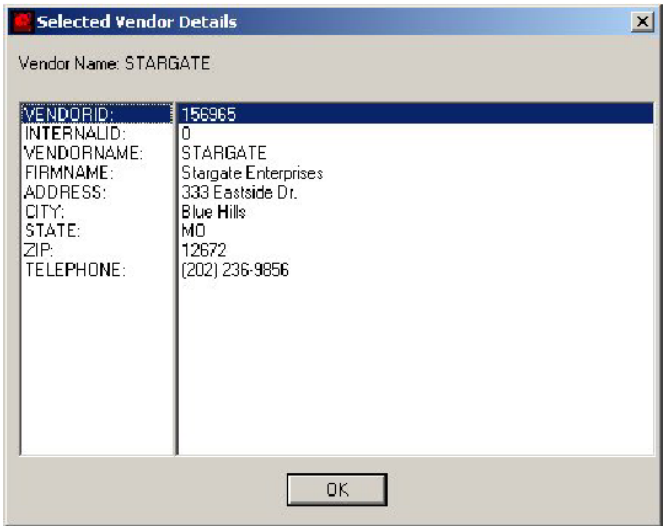


Figure 3-17: Vendor pool details for the selected candidate

To use this script code sample it is necessary that:

- A vendor pool is available and
- that your project configuration contains the following:
- There must be at least one generic document class that has a field, for example the field ‘VendorName’ for which the Associative Search Engine is selected as analysis (extraction) engine and either csv file or database imported and vendor pool has been created for it.
- There must be a button on the verification form of the document class where the buttons’ caption can be chosen by the user. The button must be associated with an action that is called ‘ShowBestSuppliers’, see Figure 3-18. If you use a different name for the action then you have to change the action name in the script code.

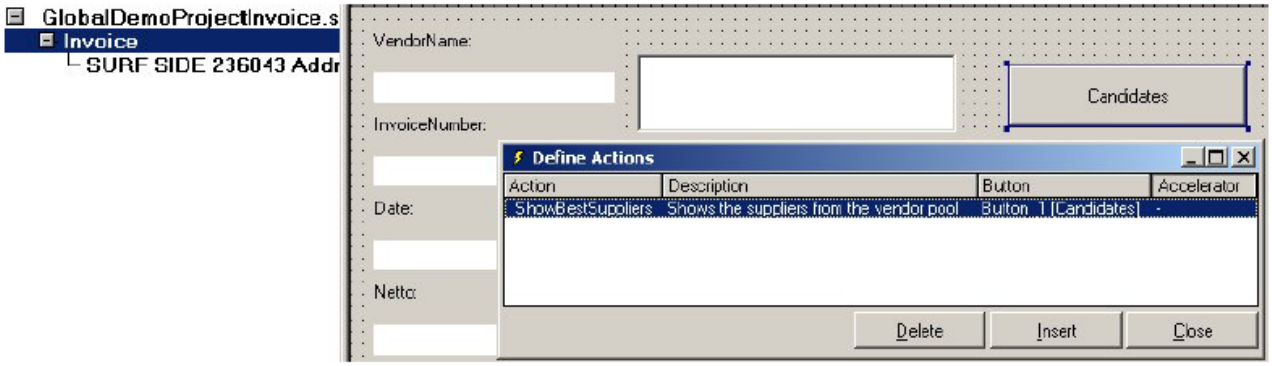


Figure 3-18: Defining the button and corresponding action event on the verification form

- Select the script editor to insert the script code that is listed below on the script sheet of the generic document class, in the example it is the class ‘Invoice.’
  The given constants ‘FIELDNAME’, ‘COLUMNNAME’ and ‘COLUMID’ have to be customized to those of the vendor pool. Be reminded that in the example below all code is copied to the sheet of the ‘Invoice’ document class, to reuse the code for other generic document classes in the same project, all code execpt the Document_OnAction event can also be copied to the scriptmodule sheet (project level). Only the constants have to be exchanged, in a way that the value for those variables are assigned for each document class in the Document_OnAction event.

```
' Project-dependent settings
' The following constants have to be adapted to the used vendor pool
' To reuse them for other generic docclasses they have to be set within
' the Document_OnAction event

‘ name of field of document class
Const FIELDNAME As String = "VendorName"
' name of vendor pool column for the supplier name
Const COLUMNNAME As String = "SupplierName"
' name of vendor pool column for the supplierID
Const COLUMNID As String = "SupplierID"

Public strModuleName As String
Public myWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc
Public mySupplierName As String
Public ListArray() As String
Public strSearchText As String
Public strSearchField As String
Public bSearchText As Boolean
Public bSearchCandidats As Boolean

'  Cedar DocClass Script
```

```
Private Sub Document_OnAction(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, _
                                                ByVal ActionName As String)
   Dim i As Integer

   If ActionName = "ShowBestSuppliers" Then
      Call AuswahlListTyp(pWorkdoc,pWorkdoc.Fields(FIELDNAME),"", "", "")
   End If
End Sub
```

```
Private Sub AuswahlListTyp(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, _
                        ByVal pField As SCBCdrPROJLib.SCBCdrField, _
                        strNummer As String, strName As String, BUKRS As String)
' This procedure retrieves the candidates for the Vendor field
' and shows the 'ShowBestSupplier' dialog
   Dim supplierID As String
   Dim strSQL As String
   Dim Combos$()
   Dim theSupplierEngine As New SCBCdrSupplierExtractionEngine
   Dim theSupplierSettings As Object
   Dim lColumnCount As Long
   Dim strColumnName As String
   Dim lIndex As Long
   Dim theDocClass As SCBCdrDocClass
   Dim theAnalysisSettings As ISCBCdrAnalysisSettings
   Dim lIDHigh As Long
   Dim lIDLow As Long
   Dim lEntryID As Long
   Dim strEntryContents As String
   Dim hilf As Long
   Dim strCandWeight As String
   Dim lUniqueId As Long
   Dim strFormattedFieldText As String
   Dim strNextLine As String
   Dim lCurrentLineIndex As Long
   Dim lStart As Long
   Dim lEnd As Long
   Dim index As Long

   Set theDocClass=Project.AllClasses.ItemByName(pWorkdoc.DocClassName)
   theDocClass.GetFieldAnalysisSettings FIELDNAME,"German",theAnalysisSettings
   Set theSupplierSettings = theAnalysisSettings
   Set myWorkdoc=pWorkdoc

   ShowBestCandidates

   lColumnCount=theSupplierSettings.ColumnCount
   ReDim Combos$(lColumnCount)

   For lIndex=0 To lColumnCount-1
      strColumnName=theSupplierSettings.ColumnName(lIndex)
      Combos$(lIndex) =strColumnName
   Next lIndex

   ReDim ListArray(pField.CandidateCount)

   For lIndex=0 To pField.CandidateCount-1
      strCandWeight=CStr(Format(pField.Candidate(lIndex).Weight*100,"00"))
      If Len(strCandWeight)<3 Then
```

```
         strCandWeight=Space(4-Len(strCandWeight))+ strCandWeight
      End If
      ListArray(lIndex) = strCandWeight + "%" + "                        "  + _
        theSupplierSettings.GetEntryByID(0,pField.Candidate(lIndex).FilterID,COLUMNNAME) + _
        "(#" + theSupplierSettings.GetEntryByID(0,
        pField.Candidate(lIndex).FilterID,COLUMNID) + ")"
   Next lIndex

   Begin Dialog UserDialog 580,300,"Search Vendor Candidates",.dlgFunction
      'Comment %GRID:10,7,1,1
      ListBox 10,60,560,200,ListArray(),.ListBox1
      Text 10,10,180,20,"Search text"
      TextBox 10,30,420,20,.Text$
      OKButton 360,270,100,21
      PushButton 470,30,100,21,"Search"
      PushButton 135,270,115,21,"Item Details ..."
      PushButton 10,270,115,21,"Show Best"
      CancelButton 470,270,100,21
   End Dialog
   Dim dlg As UserDialog

   bSearchCandidats=True
   hilf = InStrRev(ListArray(0),"                        ") + 23
   mySupplierName=Mid(ListArray(0),hilf,InStr(ListArray(0)," (#")-hilf)

   If Dialog(dlg) = -1 Then
      lUniqueId = CLng(Mid(ListArray(dlg.ListBox1),InStr(ListArray(dlg.ListBox1)," (#")+3, _
                  InStrRev(ListArray(dlg.ListBox1),")") -(InStr(ListArray(dlg.ListBox1),
                  " (#")+3)))
      theSupplierSettings.GetFormattedValueByID(0, lUniqueId, strFormattedFieldText)

      For index=0 To pField.LineCount-1
         pField.DeleteLine(0)
      Next index

      lCurrentLineIndex = 0
      lStart=1
      pField.PutUniqueEntryId(0, lUniqueId)

      For index=1 To Len(strFormattedFieldText)
         If Mid(strFormattedFieldText,index,2)=vbCrLf Or _
                                    index=Len(strFormattedFieldText) Then
            lEnd=index-lStart
            If index=Len(strFormattedFieldText) Then lEnd = lEnd + 1
            strNextLine=Mid(strFormattedFieldText,lStart,lEnd)
            lStart=index+2
            If pField.LineCount < lCurrentLineIndex + 1 Then
               pField.InsertLine(lCurrentLineIndex)
            End If
            pField.Line(lCurrentLineIndex) = strNextLine
            lCurrentLineIndex = lCurrentLineIndex + 1
         End If
      Next index

      Call ShowBestCandidates
   End If

End Sub

Private Function dlgFunction(DlgItem$, Action%, SuppValue&) As Boolean
```

```
' Sax Basic function used to code all events
Dim hilf As Long

   Select Case Action%
      Case 1 ' Dialog box initialization
      Case 2 ' Value changing or button pressed
          If DlgItem$ = "Search" Then
            Call Search(myWorkdoc,strSearchText,strSearchField)
            DlgListBoxArray "ListBox1", ListArray()
            dlgFunction = True 'do not exit the dialog
          End If
          If DlgItem$ = "ItemDetails" Then
            Call Details (myWorkdoc,mySupplierName)
            dlgFunction = True 'do not exit the dialog
          End If
          If DlgItem$ = "ShowBest" Then
            Call ShowBestCandidates
            DlgListBoxArray "ListBox1", ListArray()
            dlgFunction = True 'do not exit the dialog
          End If
          ' On list box item click
          If DlgItem$ = "ListBox1" Then
            hilf = InStrRev(ListArray(SuppValue)," ") + 23
               mySupplierName = Mid(ListArray(SuppValue),hilf, _
                                InStr(ListArray(SuppValue), " (#")-hilf)
          End If

          If DlgItem$ = "options" Then
            If DlgValue(DlgItem)=0 Then
               bSearchCandidats=True
               bSearchText=False
            Else
               bSearchCandidats=False
               bSearchText=True
            End If
          End If

         Rem dlgFunction = True 'Prevent button press from closing dialog box
      Case 3 ' TextBox or ComboBox text changed
         If DlgItem$ = "Text" Then
            strSearchText = DlgText$(DlgItem$)
          End If

         If DlgItem$ = "combo" Then
             strSearchField= DlgText$(DlgItem$)
         End If

      Case 4 ' Focus changed

      Case 5 ' Idle
         Rem dlgFunction = True ' Continue getting idle actions
      Case 6 ' Function key
         If (SuppValue = 1) Then
            Call Search(myWorkdoc,strSearchText,strSearchField)
            DlgListBoxArray "ListBox1", ListArray()
         End If
   End Select
End Function
```

```
Private Sub Details(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc,SupplierName As String)
' Shows the details for a selected candidate
   Dim supplierID As String
   Dim strSQL As String
   Dim Combos$()
   Dim theSupplierEngine As New SCBCdrSupplierExtractionEngine
   Dim theSupplierSettings As Object
   Dim lColumnCount As Long
   Dim strColumnName As String
   Dim lIndex As Long
   Dim theDocClass As SCBCdrDocClass
   Dim theAnalysisSettings As ISCBCdrAnalysisSettings
   Dim lIDHigh As Long
   Dim lIDLow As Long
   Dim i As Integer
   Dim myListArray() As String
   Dim myListArray2() As String
   Dim lEntryIndex As Long
   Dim strSupplierName As String
   Dim lEntryCount As Long
   Dim lBlank As Long


   Set theDocClass=Project.AllClasses.ItemByName(pWorkdoc.DocClassName)

   theDocClass.GetFieldAnalysisSettings FIELDNAME,"German",theAnalysisSettings
   Set theSupplierSettings = theAnalysisSettings

   lEntryCount=theSupplierSettings.EntryCount

   For lIndex = 0 To lEntryCount-1
      theSupplierSettings.GetIDByIndex (lIndex, lIDHigh, lIDLow)
      strSupplierName = theSupplierSettings.GetEntryByID(lIDHigh, lIDLow,COLUMNNAME)
      If strSupplierName = SupplierName Then
         lEntryIndex=lIDLow
         Exit For
      End If
   Next lIndex

   lColumnCount=theSupplierSettings.ColumnCount

   ReDim myListArray(lColumnCount)
   ReDim myListArray2(lColumnCount)

   For lIndex = 0 To lColumnCount-1
      strColumnName=theSupplierSettings.ColumnName(lIndex)
      myListArray(lIndex) =UCase(CStr(strColumnName)) + ": "
      If Len(CStr(theSupplierSettings.GetEntryByID(0,lEntryIndex,strColumnName)))=0 Then
         myListArray2(lIndex) = " "
      Else
         myListArray2(lIndex) =CStr(theSupplierSettings.GetEntryByID _
                                          (0,lEntryIndex,strColumnName))
      End If
   Next lIndex

   Begin Dialog UserDialog 580,300,"Selected Vendor Details",.dlgFunction
      'Comment %GRID:10,7,1,1
      Text 10,10,300,20, "Vendor Name: " + SupplierName
```

```
      ListBox 10,40,140,220,myListArray(),.ListBox2
      ListBox 150,40,430,220,myListArray2(),.ListBox3
      OKButton 250,270,90,21
   End Dialog
   Dim dlg As UserDialog
   Dialog dlg ' show dialog (wait for ok)

End Sub
```

```
Private Sub Search(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc,SearchText As _
                                                    String,SearchField As String)
' Searches for special search items in the vendor pool
   Dim theSupplierEngine As New SCBCdrSupplierExtractionEngine
   Dim theSupplierSettings As Object
   Dim lIndex As Long
   Dim theDocClass As SCBCdrDocClass
   Dim theAnalysisSettings As ISCBCdrAnalysisSettings
   Dim myListArray() As String
   Dim lCandidatenCount As Long
   Dim strCandWeight As String
   Dim pField As SCBCdrField

   Set theDocClass=Project.AllClasses.ItemByName(pWorkdoc.DocClassName)

   For lIndex=0 To myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount-1
      If lIndex >myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount-1 Then Exit For
      myWorkdoc.Fields.ItemByName(FIELDNAME).RemoveCandidate(lIndex)
      lIndex=lIndex-1
   Next lIndex

   theDocClass.GetFieldAnalysisSettings FIELDNAME,"German",theAnalysisSettings
   Set theSupplierSettings = theAnalysisSettings

   theSupplierEngine.SearchField(SearchText,pWorkdoc,theAnalysisSettings,FIELDNAME)

   lCandidatenCount=myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount

   ReDim ListArray(lCandidatenCount)

   For lIndex=0 To myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount-1


strCandWeight=CStr(Format(myWorkdoc.Fields.ItemByName(FIELDNAME).Candidate(lIndex).Weight*100,"00"))
      If Len(strCandWeight)<3 Then
         strCandWeight=Space(4-Len(strCandWeight))+ strCandWeight
      End If

      ListArray(lIndex) = strCandWeight + "%" + _
         "                       " + theSupplierSettings.GetEntryByID(0,
myWorkdoc.Fields.ItemByName(FIELDNAME).Candidate(lIndex).FilterID,COLUMNNAME) + _
         " (#" + theSupplierSettings.GetEntryByID(0,
myWorkdoc.Fields.ItemByName(FIELDNAME).Candidate(lIndex).FilterID,COLUMNID) + ")"

   Next lIndex
End Sub
```

```
Private Sub ShowBestCandidates()
' Fills the listbox of the dialog with candidates
```

```
   Dim theSupplierSettings As Object
   Dim lIndex As Long
   Dim theDocClass As SCBCdrDocClass
   Dim theAnalysisSettings As ISCBCdrAnalysisSettings
   Dim lCandidatenCount As Long
   Dim strCandWeight As String
   Dim pField As SCBCdrField

   Set theDocClass=Project.AllClasses.ItemByName(myWorkdoc.DocClassName)
   For lIndex=0 To myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount-1
      If lIndex >myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount-1 Then Exit For
      myWorkdoc.Fields.ItemByName(FIELDNAME).RemoveCandidate(lIndex)
      lIndex=lIndex-1
   Next lIndex
   theDocClass.GetFieldAnalysisSettings FIELDNAME,"German",theAnalysisSettings
   Set theSupplierSettings = theAnalysisSettings
   Set pField = myWorkdoc.Fields.ItemByName(FIELDNAME)
   pField.FieldState=CDRFieldStateReset
   theDocClass.AnalyzeField(myWorkdoc,FIELDNAME)
   lCandidatenCount=myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount
   ReDim ListArray(lCandidatenCount)

   For lIndex=0 To myWorkdoc.Fields.ItemByName(FIELDNAME).CandidateCount-1
 strCandWeight=CStr _
(Format(myWorkdoc.Fields.ItemByName(FIELDNAME).Candidate(lIndex).Weight*100,"00"))
      If Len(strCandWeight)<3 Then
         strCandWeight=Space(4-Len(strCandWeight))+ strCandWeight
      End If
      ListArray(lIndex) = strCandWeight + "%" + _
               "                       " + theSupplierSettings.GetEntryByID(0, _
myWorkdoc.Fields.ItemByName(FIELDNAME).Candidate(lIndex).FilterID,COLUMNNAME) + _
             " (#" + theSupplierSettings.GetEntryByID _
(0,myWorkdoc.Fields.ItemByName(FIELDNAME).Candidate(lIndex).FilterID,COLUMNID) + ")"
   Next lIndex
End Sub
```

3.3.4 Quick Memory Search Pools Generation via Associative Database Search Engine in Custom Script

3.3.4.1 Description

The Associative Database Search engine is now extended to support dynamic generation of pool in memory without the necessity to store the pool on the disk. The search pool's rows can now be imported from any information source without the necessity to configure a dedicated Oracle Forms Recognition field.

The following custom script sample shows how a pool can be created from within the Oracle Forms Recognition script. Copy this example as a template when creating your own script implementations, for example, those that are based on external database queries:

```
Private Function fnCreatePoolInMemory(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)

On Error GoTo Errorlabel

Dim myColumns(4) As String
```

```
myColumns(0) = "SupplierID"
myColumns(1) = "SupplierIndex"
myColumns(2) = "SupplierName"
myColumns(3) = "Field4"

Dim myArray(10,4) As String

myArray(0,0) = "4500000000100001"
myArray(0,1) = "4500000001"
myArray(0,2) = "00001"
myArray(0,3) = "Packets of Penguins"

myArray(1,0) = "4500000000100002"
myArray(1,1) = "4500000001"
myArray(1,2) = "00002"
myArray(1,3) = "Big bar of Galaxy chocolate"

myArray(2,0) = "4500000000100003"
myArray(2,1) = "4500000001"
myArray(2,2) = "00003"
myArray(2,3) = "Cream Eggs - 20KG Case"

myArray(3,0) = "4500000000100004"
myArray(3,1) = "4500000001"
myArray(3,2) = "00004"
myArray(3,3) = "Cadbury's Fruit & Nut - 1 box"

myArray(4,0) = "4500000000100005"
myArray(4,1) = "4500000001"
myArray(4,2) = "00005"
myArray(4,3) = "Case of Dime Bars"

myArray(5,0) = "4500000000100006"
myArray(5,1) = "4500000001"
myArray(5,2) = "00006"
myArray(5,3) = "Case of tubes of smarties"

myArray(6,0) = "4500000000100007"
myArray(6,1) = "4500000001"
myArray(6,2) = "00007"
myArray(6,3) = "Mars Bars - 10KG Box"

myArray(7,0) = "4500000000100008"
myArray(7,1) = "4500000001"
myArray(7,2) = "00008"
myArray(7,3) = "Kit Kats"

myArray(8,0) = "4500000000100009"
myArray(8,1) = "4500000001"
myArray(8,2) = "00009"
myArray(8,3) = "Double Deckers"

myArray(9,0) = "4500000000100010"
myArray(9,1) = "4500000001"
myArray(9,2) = "00010"
myArray(9,3) = "Case of Snickers"

Dim lInd As Long
Dim oField As New SCBCdrField
```

```
Dim strTempField As String
Let strTempField = "TempAssaPoolInMemory"
pWorkdoc.Fields.Add(oField,strTempField)

Dim cdrADS As New SCBCdrSupExSettings
cdrADS.StartMemoryPool()
'-------------------------------------------- Create Columns
Dim j As Long
For j = 0 To UBound(myColumns) - 2
   cdrADS.AddColumn(myColumns(j), False, lInd)
Next
cdrADS.AddColumn(myColumns(j), True, lInd)  'Last column is search column

cdrADS.IdentityColumn = myColumns(0)                ' SupplierID
cdrADS.FieldContentsFormat = "[" & myColumns(2) & "]" ' [SupplierName]

'-------------------------------------------- Fill Pool
Dim i As Long
For i = 0 To UBound(myArray) - 1

   cdrADS.StartAddEntry()

   For j = 0 To UBound(myArray, 2) - 1
      cdrADS.ChangeEntry(myColumns(j), myArray(i,j))
   Next

   cdrADS.CommitAddEntry(lInd)

Next

cdrADS.CommitMemoryPool()
'-------------------------------------------------------------------------------------

Dim pTable As SCBCdrTable
Dim lngRow As Long
Dim lngCandidate As Long
Dim SupplierExtractionEngine As New SCBCdrSupplierExtractionEngine

Set pTable = pWorkdoc.Fields("LineItems").Table(pWorkdoc.Fields("LineItems").ActiveTableIndex)

For lngRow = 0 To pTable.RowCount-1

 For lngCandidate = oField.CandidateCount-1 To 0 Step -1
   oField.RemoveCandidate(lngCandidate)
 Next lngCandidate

SupplierExtractionEngine.SearchField(pTable.CellText("Description", lngRow), pWorkdoc, cdrADS, strTempField)

 If oField.CandidateCount > 0 Then

   pTable.CellText("PO", lngRow) = pWorkdoc.Fields("PONumber").Text
   pTable.CellText("Line", lngRow) = oField.Candidate(0).Text
   pTable.CellText("Weighting", lngRow) = Format(CStr(oField.Candidate(0).Weight), "0.000")

 Else

   pTable.CellText("PO", lngRow) = ""
   pTable.CellText("Line", lngRow) = ""
   pTable.CellText("Weighting", lngRow) = ""
```

```
 End If

Next lngRow

GoTo ContinueLabel

ErrorLabel:
   MsgBox Err.Description

ContinueLabel:

pWorkdoc.Fields.Remove("TempAssaPoolInMemory")

End Function
```

3.3.4.2 Usage

This feature can be used for rapid implementation of any custom search functionality in Oracle Forms Recognition script. For example, in context of typical project, for searching for the best matching items in a customer's database that match each row's line items' description.

But, in principle, any information queried from a customer database or any other source can be quickly packaged as ASD memory pool and then searched using all the powerful attributes of the product's context search technologies.

3.3.5 Automatically Adjusting of the Page Separation Engine's Decision

In some cases it may appear to very advantageous to increase accuracy or reduce amount of so-called false positives cases of the automatic page separation from within Oracle Forms Recognition custom script. For this purpose the system administrator can implement a handler of the "AppendWorkdoc" event:

```
Private Sub ScriptModule_AppendWorkdoc(pLastWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, pCurrentWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, pAppendType As SCBCdrPROJLib.CdrMPType)
   pAppendType = CdrSubseqPage
End Sub
```

Via analyzing and comparing of the preceding document ("pLastWorkdoc" parameter) and the currently processed document ("pCurrentWorkdoc" parameter) the script can change (or accept) the automatic engine's decision by setting the "pAppendType" variable to either "CdrSubseqPage" (for "merge" decision) or "CdrFirstPage" (for split decision).

One very common use case to introduce such a script can be about taking advantage of the Associative Search vendor and/or address extraction. Truly, if the Associative Search engine tells (with high probability) that the preceding document belongs to vendor type A and the current document – to a different vendor type B, this may appear (but not in all the projects) a very strong condition for a split decision, no matter what kind of result was previously detected by the Page Separation engine. Such an extra script rule can, beyond doubt, significantly increase precision and reliability of the auto-processing procedure.

Chapter 4 Reference of Script Events

4.1 ScriptModule

Cedar ScriptModule Event Interface

Project events are specific for one Oracle Forms Recognition Project, but within an Oracle Forms Recognition Project, all documents and fields share the same implementation of these events. This means that they are document class (DocClass) independent. As the Project events belong to the "sheet" ScriptModule, all events start with the prefix ScriptModule.

4.1.1 AppendWorkdoc

ScriptModule_AppendWorkdoc (pLastWorkdoc As ISCBCdrWorkdoc, pCurrentWorkdoc As ISCBCdrWorkdoc, pAppendType As CdrMPType)

pLastWorkdoc

Last Workdoc

pCurrentWorkdoc

Current Workdoc

pAppendType

An enumeration type based on the definition of the relationship of the Last and Current Workdoc. In other words, whether the current Workdoc is to be treated as a new document or appended to the last Workdoc. The user can change this parameter using script to influence the decision.

CdrMPType	Value	Description
CdrFirstPage	1	The classified page is the first page of a new document and does not belong to the previous page
CdrLastPage	3	The classified page is the last page of the current document. The next page will start a new document
CdrPageUndefined	0	There was no classification result
CdrSinglePage	4	The classified page belongs to a single page document
CdrSubseqPage	2	The classified page belongs to the previous page. More pages can be appended to the current document

This method is used to append a given Workdoc after last Workdoc on the base of CdrMPType.

4.1.2 ExportDocument

ScriptModule_ExportDocument (pWorkdoc As ISCBCdrWorkdoc, ExportPath As String, pCancel As Boolean)

pWorkdoc

Workdoc, which should be exported

ExportPath

Export path, which was configured within the Oracle Forms Recognition Runtime Server settings (no changes possible)

pCancel

Set this variable to TRUE to cancel the export

The Export event provides the ability to implement a customer specific export of all extracted data. Here it is possible to write export files in any format or to connect a database or workflow system and write the data directly to the next system.

Example:

```
Private Sub ScriptModule_ExportDocument(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, ByVal ExportPath As
String, pCancel As Boolean)

End Sub
```

4.1.3 Initialize

ScriptModule_Initialize (ModuleName As String)

ModuleName

Name of the current module, values: "Server", "Designer" or "Verifier"

The Initialize event is called when a batch is opened for processing. The content of ModuleName depends on the current application. The following values are possible:

"Server" - Script was called from Oracle Forms Recognition Designer DesignMode or RunTimeMode

“Designer” - Script was called from Oracle Forms Recognition Designer TrainMode

“Verifier” - Script was called from Oracle Forms Recognition Designer VerifierTestMode

"Verifier" - Script was called from Oracle Forms Recognition Verifier

"Server" - Script was called from Oracle Forms Recognition Runtime Server.

Example:

```
Public Sub ScriptModule_Initialize(ByVal ModuleName As String)
   DBname=Project.Filename
   DBname=Left(DBname,InStrRev(DBname,''\'')) & ''InvoiceBestellNo.mdb''
   Set DB=OpenDatabase(DBname)
End Sub
```

4.1.4 PostClassify

ScriptModule_PostClassify (pWorkdoc As ISCBCdrWorkdoc)

pWorkdoc

Workdoc object which has been classified

The PostClassify event will be called after all defined classification methods have been executed by the Cedar Project. During this event it is possible to validate the classification result and, if necessary, also to change the result to another or non-DocClass.

Use the DocClassName property of the Workdoc to see the current classification result and to change DocClass of the Workdoc.

Example:

```
Private Sub ScriptModule_PostClassify(pWorkdoc As SCBCdrWorkdoc)
    if (MyValidation(pWorkdoc) = TRUE) then
      ' classification was ok, do nothing
    else
      ' classification was wrong, change it
      if (DoSomeMagic(pWorkdoc) = TRUE) then
        ' assign “Invoice”  as result of the classification
        pWorkdoc.DocClassName = ''Invoice''
      else
        ' don't know what kind of DocClass it is
        ' force classification correction
        pWorkdoc.DocClassName = ''''
      end if
    end if
End Sub
```

4.1.5 PreClassify

ScriptModule_PreClassify (pWorkdoc As ISCBCdrWorkdoc)

pWorkdoc

Workdoc object, which should be classified

The PreClassify event will be called before any defined classification method is executed by the Cedar Project. During this event it is possible to apply an existing name of a DocClass to the WorkDoc. In this case, all defined classification methods will be skipped and the assigned DocClass will be used as the result of the classification.

Use the DocClassName property of the Workdoc to apply DocClass to the Workdoc.

Example:

```
Private Sub ScriptModule_PreClassify(pWorkdoc As SCBCdrWorkdoc)
    if ( DoSomeMagic(pWorkdoc) = TRUE ) then
        ' assign “Invoice”  as result of the classification
        pWorkdoc.DocClassName = ''Invoice''
    else
        ' do nothing and continue with normal classification
    end if
End Sub
```

4.1.6 RouteDocument

ScriptModule_RouteDocument (pWorkdoc As ISCBCdrWorkdoc, State As Single)

pWorkdoc

Workdoc object, which was classified and extracted

State

This parameter contains the current default state, which will be assigned to the Workdoc. Value can be changed from a script

This script event is called in Oracle Forms Recognition Runtime Server after the classification and extraction step and in Oracle Forms Recognition Verifier before saving of a document. It can be used to route a document to a special state, depending on the data of the current WorkDoc.

```
Example:
Private Sub ScriptModule_RouteDocument(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, State As Integer)
If pWorkdoc.Fields(“Field1”).Valid = FALSE then
    ' route to 500 if Field1 is not valid
    State = 500
    Exit sub
End if
If pWorkdoc.Fields(“Field2”).Valid = FALSE then
    ' route to 520 if Field2 is not valid
    State = 520
    Exit sub
End if
'  else use default state
End Sub
```

4.1.7 Terminate

ScriptModule_Terminate (ModuleName As String)

ModuleName

Name of the current module, values: "Designer,', "Verifier" or "Server"

The Terminate event is called before a batch is closed after processing. The content of ModuleName depends on the current application. The following values are possible:

"Server" - Script was called from Oracle Forms Recognition Designer DesignMode or RunTimeMode

"Designer" - Script was called from Oracle Forms Recognition Designer TrainMode

"Verifier" - Script was called from Oracle Forms Recognition Designer Verifier TestMode

"Verifier" - Script was called from Oracle Forms Recognition Verifier

"Server" - Script was called from Oracle Forms Recognition Runtime Server.

```
Example:
Private Sub ScriptModule_Terminate(ByVal ModuleName As String)
       DB.Close
       Set DB = nothing
End Sub
```

4.1.8 VerifierClassify

ScriptModule_VerifierClassify (pWorkdoc As ISCBCdrWorkdoc, Reason As CdrVerifierClassifyReason, ClassName As String)

pWorkdoc

Reference to the currently processed document.

Reason

The reason why the script routine decided to reject or accept the document.

ClassName

The name of the document class to which it is classified manually.

This event occurs only inVerifier when a document is manually classified.

4.1.9 VerifierFormLoad

ScriptModule_VerifierFormLoad (pWorkdoc As ISCBCdrWorkdoc, FormName As String, FormClassName As String)

pWorkdoc

Reference to the currently processed document.

FormName

A string value that contains the current form name that Verifier application is going to load. The name can be modified in the custom script to initiate loading of a different form when required.

FormClassName

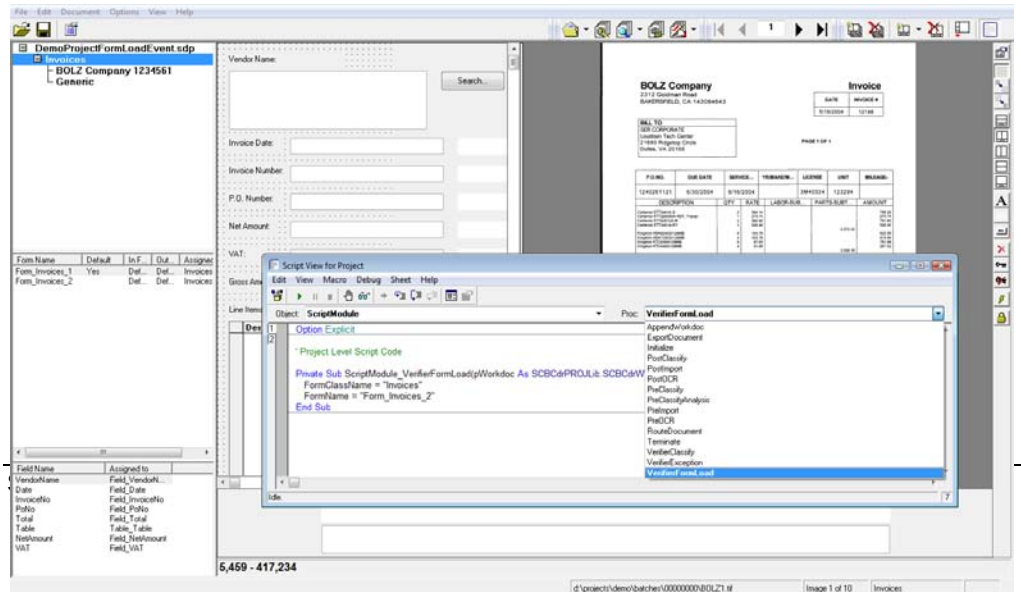
A string variable that contains the current class name of the verification form is to be loaded from. This name can be changed from within the Oracle Forms Recognition custom script to point to a different document class, in case the desired verification form is located in this different class.

The event is fired right before the Verifier application starts loading the indexing verification form for the next document to process.

The new “VerifierFormLoad” event is now available for dynamic substitution of verification forms in Oracle Forms Recognition Verifier application.

4.1.9.1 Description

In order to implement the script handler of this event, start the Oracle Forms Recognition Designer application, load the desired project file, select the project node in Definition mode, open the Script Editor, select the “Script Module” object and click on the new “VerifierFormLoad” item in “Proc” drop

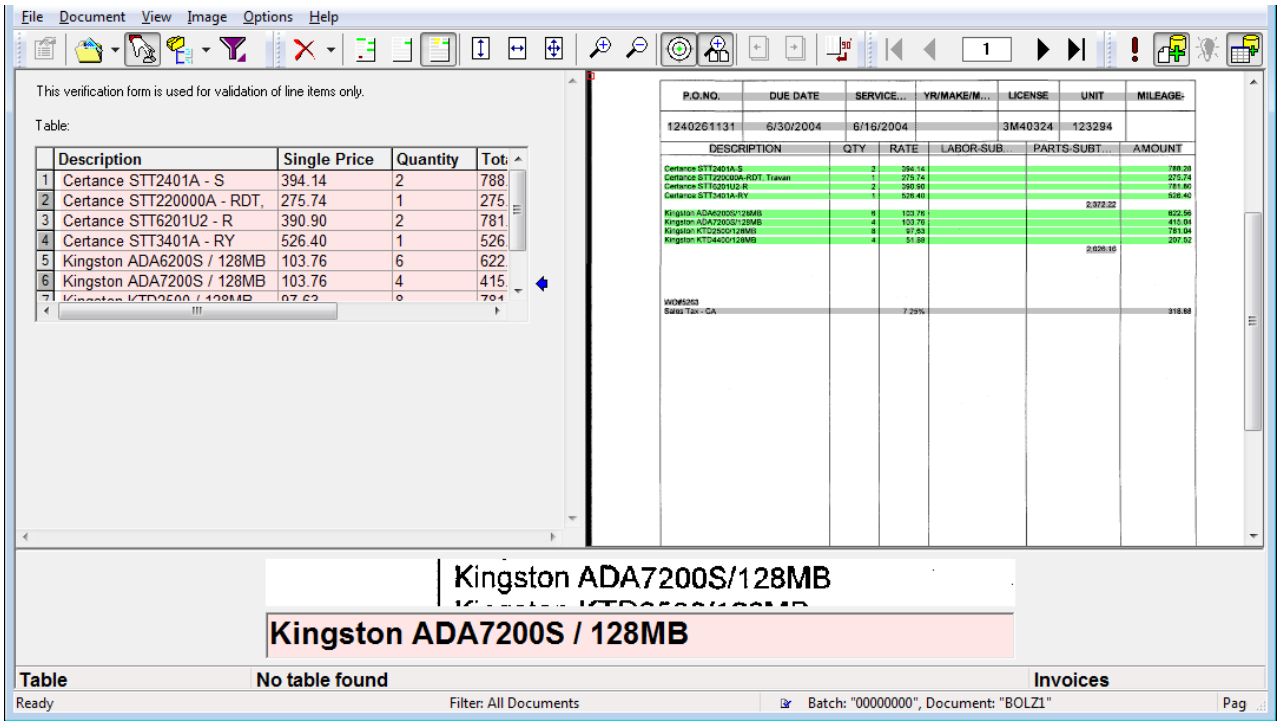


down list:

For example, the following simple implementation of the “VerifierFormLoad” event is going to (in this simple case non-optionally) replace the standard form “Form_Invoices_1” with a custom one “Form_Invoices_2” defined for the same document class:

```
Option Explicit

' Project Level Script Code

Private Sub ScriptModule_VerifierFormLoad(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, FormClassName As String,
FormName As String)

   FormClassName = "Invoices"
   FormName = "Form_Invoices_2"
End Sub
```

As a result, Verifier application will always load the simple second form, specified in the script:

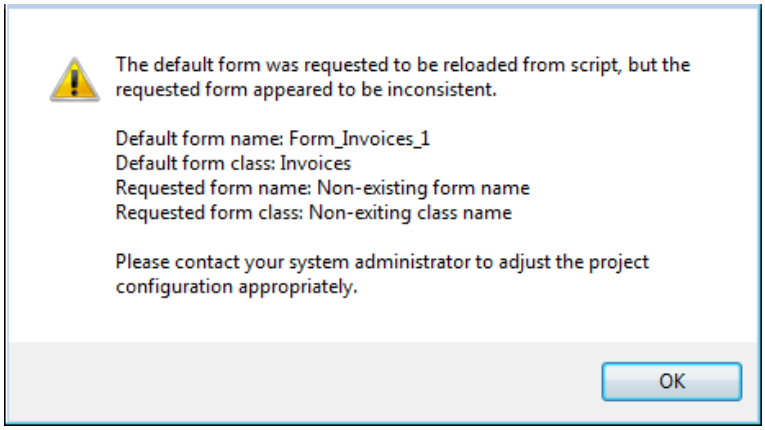


In case the script modifies the form and form’s class references incorrectly (for example, referring to a non-existing verification form of a class, or in case the form does not exist in the specified class, and so on), a warning message is displayed to the Verifier user.

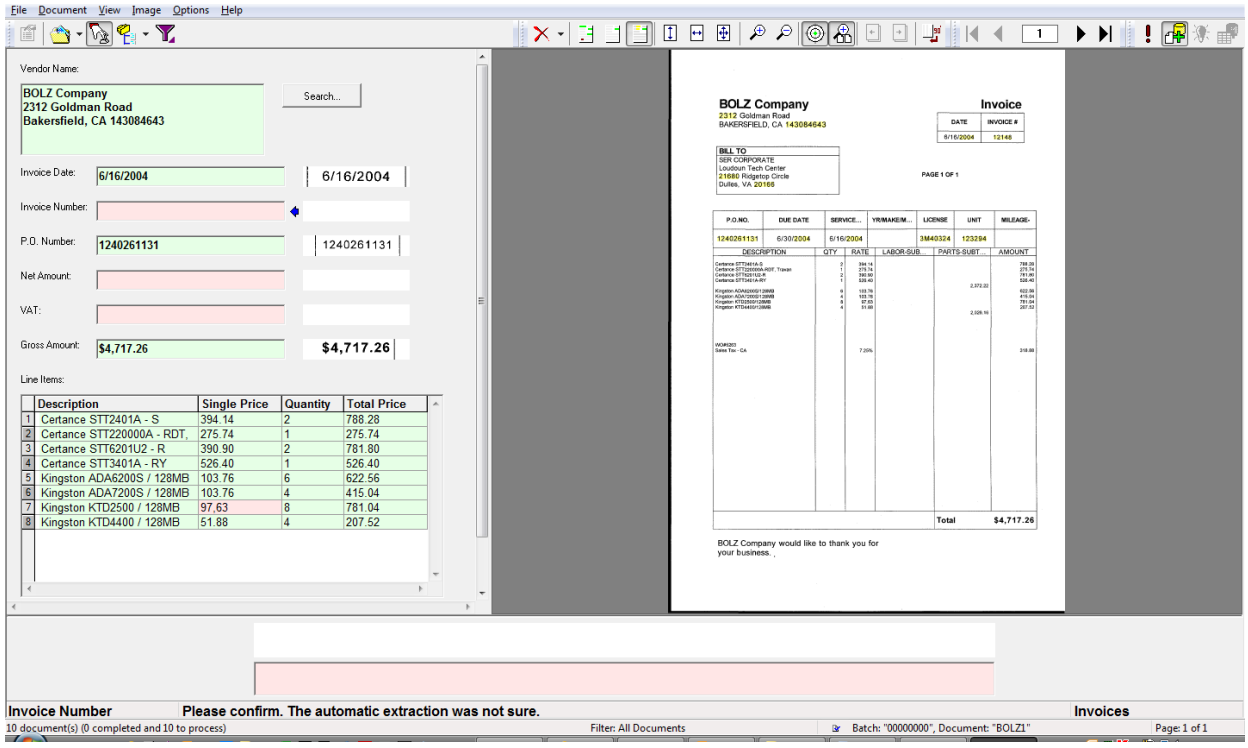
For example, in case of the wrong script like this:

```
Private Sub ScriptModule_VerifierFormLoad(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, FormClassName As String,
FormName As String)

   FormClassName = "Non-exiting class name"
   FormName = "Non-existing form name"

End Sub
```

The Verifier application is going to show the following warning message:



And then load the standard verification form (i.e., the one that the application would be loading anyway if the script handler of “VerifierFormLoad” event did not exist) instead of the wrong one proposed the by custom script:



Note: the new event is fired from within the Oracle Forms Recognition Verifier application only and cannot be tested in Oracle Forms Recognition Designer application.

As another relevant extension, the former document class level “FocusChaned” event has been extended with a new “Reason” called “CdrBeforeFormLoaded”. The event is now also fired right

before the desired verification form is about to be loaded but after the “VerifierFormLoad” event described above.

Below is a script sample that shows how the handler of this extended reason can be implemented in the Oracle Forms Recognition custom script:

```
Private Sub Document_FocusChanged(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, ByVal Reason As
SCBCdrPROJLib.CdrFocusChangeReason, ByVal OldFieldIndex As Long, pNewFieldIndex As Long)

   If Reason = CdrBeforeFormLoaded Then
      MsgBox "The form has not been loaded yet"
   End If

End Sub
```

4.1.9.2 Usage

The features described in the present section can be used for many different purposes, for example:

- To optionally load non-standard verification form(s) in accordance with some parameters of the processed document.
- To dynamically translate the content of verification form into, e.g., a different language or simply load the required verification form in accordance with the current system Regional settings.

4.2 Document

Cedar DocClass Event Interface

Document events are specific for each Cedar DocClass instance. Each DocClass has its own script module and implementation of script events.

4.2.1 FocusChanged

Document_FocusChanged (pWorkdoc As ISCBCdrWorkdoc, Reason As CdrFocusChangeReason, OldFieldIndex As Long, pNewFieldIndex As Long)

pWorkdoc

Reference to the currently displayed workdoc.

Reason

Reason of the current focus change, which can be either Tab key, Enter key, mouse click, or initial loading.

OldFieldIndex

Index of the current select field. In case of initial loading this will be -1.

pNewFieldIndex

Index of the field which should be selected now. Can be modified during the script event to keep the focus in the previous field or set it to another field.

This event will be fired each time before the focus inside the verification form is changed. It is possible to influence the focus change by modifying the pNewFieldIndex parameter. It is possible to write a different field index into that parameter, which causes the Verifier to change to specified field instead

to the originally selected field. Special care has to be taken when implementing this event, since a wrong implementation can prevent the user from selecting a certain field.

Example:

```
Private Sub Document_FocusChanged(pWorkdoc As SCBCdrWorkdoc, Reason As CdrFocusChangeReason,
OldFieldIndex As Long, pNewFieldIndex As Long)
' Below you can find the sample of script code that helps to skip table
' data validation in Verifier (for a table with 2 columns):
 Dim theEmptyTable As SCBCdrPROJLib.SCBCdrTable
 Dim theEmptyTableField As SCBCdrPROJLib.SCBCdrField

   ' Initializes table and field references
   Set theEmptyTable = _
pWorkdoc.Fields("EmptyTable").Table(pWorkdoc.Fields("EmptyTable").ActiveTableIndex)
   Set theEmptyTableField = pWorkdoc.Fields("EmptyTable")

   ' Makes table object valid
   theEmptyTable.CellValid(0,0) = True
   theEmptyTable.CellValid(1,0) = True
   theEmptyTable.RowValid(0) = True
   theEmptyTable.TableValid = True

' Makes table field valid
' (table object is a part of more generic field object)
   theEmptyTableField.Valid = True
   theEmptyTableField.Changed = False

   ' Releases references
   Set theEmptyTable = Nothing
   Set theEmptyTableField = Nothing
End Sub
```

4.2.2 OnAction

Document_OnAction (pWorkdoc As ISCBCdrWorkdoc, ActionName As String)

pWorkdoc

Reference to the currently displayed workdoc.

ActionName

Name of the action which was assigned to the pressed button or short cut key.

This event will be fired if any of the configured actions was caused by the user. Actions have to be configured in the Verifier design mode. Actions can either caused if a user pressed a button or any of the configured keyboard short cuts.

4.2.3 PostExtract

Document_PostExtract (pWorkdoc As ISCBCdrWorkdoc)

pWorkdoc

Current Workdoc object

The PostExtract event will be called after all defined analysis or evaluation methods have been executed by the Cedar DocClass. During this event, it is possible to examine and change the results of one or more fields of the document.

This event can also be used in combination with generic Designer settings to establish multiple classification. In Designer, establish a default classification result. Then set "pWorkdoc.DocClassName" to a different class in this event. This technique enables you to keep the generic extraction pointed toward the default class, while moving the validation script a different class.

Example:

```
Private Sub Document_PostExtract(pWorkdoc As SCBCdrWorkdoc)
   Dim Number as string
   Dim Name as string

   ' get fields name and number and make a database lookup
   Number = pWorkdoc.Fields(“Number”)
   Name = pWorkdoc.Fields(“Name”)

   if len(Name) = 0 and len(number) > 0 then
     ' try to get Name from a database using the found number
     Name = GetNameFromDB(Number)
     pWorkdoc.Fields(“Name”) = name
   else
     if len(Name) > 0 and len(number) = 0 then
       Number = GetNumberFromDB(Name)
       pWorkdoc.Fields(“Number”) = Number
        end if
   end if
End Sub
```

4.2.4 PreExtract

Document_PreExtract (pWorkdoc As ISCBCdrWorkdoc)

pWorkdoc

Current Workdoc object

The PreExtract event will be called before any defined analysis or evaluation method will be executed by the Cedar DocClass. During this event it is possible to assign results to one or more Fields of the document, and by changing the Field state, to skip defined analysis and evaluation methods.

Example:

```
Private Sub Document_PreExtract(pWorkdoc As SCBCdrWorkdoc)
   Dim MyResult as string
   MyResult = DoSomeMagic(pWorkdoc)
   if (len(MyResult) > 0) then
      ' assign result to a single field
      pWorkdoc.Fields(“Number”) = MyResult;
      ' skip defined analysis and evaluation methods
      pWorkdoc.Fields(“Number”).FieldState
                           = CDRFieldStateEvaluated
   end if
end Sub
```

4.2.5 Validate

Document_Validate (pWorkdoc As ISCBCdrWorkdoc, pValid As Boolean)

pWorkdoc

Current Workdoc object

pValid

Parameter containing the current valid state of the Workdoc

The Validate event can be used to perform validation on document level. At this point the validation of all single Fields has been executed. If one of the Fields is still invalid, pValid will be FALSE. During the Document_Validate event, it is possible to implement validation rules combining several Fields. This may cause some Fields to be invalid again. Please do not make the document invalid if all Fields are valid because the Verifier needs an invalid Field for focus control. If you want to keep the document invalid, always set at least one Field to an invalid state.

It is also possible to make invalid Fields valid during document validation. Therefore, you must set the Valid property of the appropriate fields to TRUE.

Example:

```
Private Sub Document_Validate(pWorkdoc As SCBCdrWorkdoc, pValid As Boolean)
Dim Number as string
Dim Name as string

   ' get fields name and number and make a database lookup
   Number = pWorkdoc.Fields("Number")
   Name = pWorkdoc.Fields("Name")

   if LookupDBEntry(Name, Number) = FALSE then
      ' the Name/Number pair is NOT in the database
      ' set the document state to invalid
      pValid = FALSE
        ' make both fields invalid and provide an error description
       pWorkdoc.Fields("Number").Valid = FALSE
       pWorkdoc.Fields("Number").ErrorDescription = "Not in database"
       pWorkdoc.Fields("Name").Valid = FALSE
       pWorkdoc.Fields("Name").ErrorDescription = "Not in database"
   end if
End Sub
```

4.2.6 VerifierTrain

Document_VerifierTrain (pWorkdoc As ISCBCdrWorkdoc, ProposedClassName As String, WillTrain As Boolean, VerifierReason As CdrLocalTrainingReason, ScriptReason As String)

pWorkdoc

Contains the reference to the currently processed document.

WillTrain

Boolean value for the current learning state. True, when the document is going to be learnt and False when it will not be learnt.

VerifierReason

Contains the reason why the document was taken for training or why it was rejected. The reason parameter should be one of the predefined enumerated values for CdrLocalTrainingReason.

ScriptReason

Contains the reason why the script routine decided to reject or accept the document.

After a document processed in self-learning Verifier has been checked whether it is supposed to be automatically trained for the local project, the Verifier has to fire an event that adds a document to the local learnset.

4.3 <Field$_n$> (Cedar FieldDef Event Interface)

Field events are specific for each Cedar field of each DocClass. Field events appear within the script sheet of their DocClass. That means all events for the field “Number” of the document class Invoice must be implemented within the script sheet of the DocClass Invoice.

Within the script the name of the fields will appear as specifier for the field. That means the Validate event for the field “Number” will appear as method "Number_Validate." During this documentation, <Fieldn> will be used as a placeholder for the name of the field. The Validate event will be named here as <Fieldn>_Validate.

4.3.1 CellChecked

<Field$_n$>_CellChecked (pTable As ISCBCdrTable, pWorkdoc As ISCBCdrWorkdoc, Row As Long, Column As Long, Checked As Boolean)

pTable

Current Table object.

pWorkdoc

Current Workdoc object.

Row

This parameter contains the value of the current row on which the user clicked.

Column

This parameter contains the value of the current column on which the user clicked.

Checked

Boolean value that is TRUE when the cell is checked, otherwise its value is FALSE.

Occurs when a check-box cell of the table has been checked or unchecked by the user. Alternatively the CellFocusChanged event can be used where the variable Reason has the value CdrTfcrCellBitmapClicked.

Example:

```
Private Sub Table_CellChecked(pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, ByVal Row As Long, ByVal Column As Long, ByVal Checked As Boolean)
   If Checked = True Then
      ' The cell (Row, Column) has been checked
      ...
```

```
   End If
End Sub
```

4.3.2 CellFocusChanged

<Field$_n$>_CellFocusChanged (pTable As ISCBCdrTable, pWorkdoc As ISCBCdrWorkdoc, Reason As CdrTableFocusChangeReason, OldRow As Long, OldColumn As Long, pNewRow As Long, pNewColumn As Long)

pTable

Current Table object.

pWorkdoc

Current Workdoc object.

Reason

Parameter that contains the kind of focus change that has occurred. Its possible values are described below:

CdrTableFocusChangeReason	Value	Description
CdrTfcrCellBitmapClicked	5	The cell focus in a table has changed because the cell bitmap was clicked
CdrTfcrCellDoubleClicked	13	The cell focus in a table has changed because a cell was double clicked
CdrTfcrCellLocationClicked	14	The cell focus in a table has changed because the location of this cell was clicked in the active Cairo Viewer
CdrTfcrColumnMapped	9	The cell focus in a table has changed because a column was mapped
CdrTfcrColumnsSwapped	11	The cell focus in a table has changed because some columns were swapped
CdrTfcrColumnUnmapped	10	The cell focus in a table has changed because a column was unmapped
CdrTfcrEnterPressed	2	The cell focus in a table has changed because validation of a cell was invoked
CdrTfcrFocusRefreshed	12	The cell focus in a table has changed because the table had to be refreshed
CdrTfcrFormLoaded	1	The cell focus in a table has initialized because a form was loaded
CdrTfcrMouseClicked	4	The cell focus in a table has changed because a cell was selected by mouse click
CdrTfcrRowsMerged	8	The cell focus in a table has changed because some rows were merged
CdrTfcrRowsRemoved	7	The cell focus in a table has changed because one or several table rows were removed
CdrTfcrTableCandidateChanged	6	The cell focus in a table has changed because a new table candidate was selected by a user
CdrTfcrTabPressed	3	The cell focus in a table has been changed since a user pressed TAB or arrow keys
CdrTfcrUnknownReason	0	The cell focus in a table has changed because unknown cause

OldRow

This parameter contains the value of the derivation row.

OldColumn

This parameter contains the value of the derivation column.

pNewRow

This parameter contains the value of the destination row. This value can be changed, e.g., set back to OldRow value, to forbid, for example, a double-click on the special column.

pNewColumn

This parameter contains the value of the destination column. This value can be changed, e.g., set back to OldColumn value, to forbid, for example, a double-click on the special column.

This event occurs each time the focus inside the verification table is going to be changed or can be changed potentially. For example, the “CdrTfcrSelectionCopied” event may occur if a user drags an area of text into the viewer and drops it in one of verification fields. In principle, the currently selected field can be the destination field so that the focus will not be changed. Or, the event is fired anyway so that you can set this specific type of user action in the script.

In other words, this event can be used not only to control changing of input focus, but also to control specific user actions. For more details, please check description of the “Reason” parameter in the section below.

Example:

```
Private Sub Table_CellFocusChanged(pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, ByVal Reason As SCBCdrPROJLib.CdrTableFocusChangeReason, ByVal OldRow As
Long, ByVal OldColumn As Long, pNewRow As Long, pNewColumn As Long)


   Select Case Reason
      Case CdrTfcrCellBitmapClicked
         ' Occurs when a user clicks on cell's picture, e.g., on check-box image of a check-box cell.
      Case CdrTfcrCellDoubleClicked
         ' Occurs if a user double clicks on a table cell. Could be useful if it ' is designed to
         ' implement a kind of database look-up, etc by double clicking on a cell.
      Case CdrTfcrCellLocationClicked
         ' Occurs when a user clicks on a word that is linked to one of the cells in image viewer.
         ' This will cause setting of keyboard focus to the corresponding table cell.
      Case CdrTfcrColumnMapped
         ' Occurs when a user maps a column.
      Case CdrTfcrColumnsSwapped
         ' Occurs when a user swaps two columns.
      Case CdrTfcrColumnUnmapped
         ' Occurs when a user unmaps a column.
      Case CdrTfcrEnterPressed
         ' Occurs when "Enter" key is pressed, i.e. cell (table) validation is activated.
      Case CdrTfcrFocusRefreshed
         ' Occurs when the application refreshes a table.
      Case CdrTfcrFormLoaded
         ' Occurs right after a new document to verify is loaded.
      Case CdrTfcrMouseClicked
         ' Occurs when a cell is selected by mouse click.
      Case CdrTfcrRowsMerged
         ' Occurs when rows were merged to one row.
      Case CdrTfcrRowsRemoved
         ' Occurs when a user removes a row.
      Case CdrTfcrTableCandidateChanged
         ' Occurs when a user changes current table candidate.
      Case CdrTfcrTabPressed
         ' Occurs when the focus is changed to another cell by arrow keys or TAB keys.
      Case CdrTfcrUnknownReason
         ' Focus is changed due to unknown reason.
   End Select
   ' Example of changing cell focus from the script:
   ' when document is opened, set focus to the first cell
```

```
   If Reason = CdrTfcrFormLoaded Then
      pNewRow = 0
      pNewColumn = 0
   End If
   ' Example of changing cell focus from the script: do not allow selection of first cell by mouse
   If OldRow = 0 And OldColumn = 0 And Reason = CdrTfcrMouseClicked Then
      pNewRow = 1
      pNewColumn = 1
   End If
End Sub
```

4.3.3 Format

<Field$_n$>_Format (pField As ISCBCdrField)

pField

Field object

The Format event can be used to reformat the content of a Field, for example to unify a date or amount format or removing prefixes and suffixes. This event can be used to prepare the field data for validation. Be reminded that the content of pField.Text is normally used for learning within the Oracle Forms Recognition engines. If the users wants to change the output format for the fields' content rather use the script event FormatForExport.

Example:

```
Private Sub Amount_Format(pField As SCBCdrField)
   Dim NewAmount as string
   if MyReformatAmount(pField, NewAmount) = TRUE then
      ' reformatting of the text field is successful to prepare a field for validation
      pField.Text = NewAmount
   end if
End Sub
```

4.3.4 FormatForExport

<Field$_n$>_FormatForExport (pField As ISCBCdrField)

pField

Current field.

The FormatForExport event can be used to reformat the content of a Field, for example to unify a date or amount format or removing prefixes and suffixes and to keep this additional information within pField.FormattedText rather than to change pField.Text. This text is normally used for learning within the Oracle Forms Recognition engines. This formatted text can also be used for Export.

Example:

```
Private Sub Amount_Format(pField As SCBCdrField)
   Dim NewAmount as string
   if MyReformatAmount(pField, NewAmount) = TRUE then
      ' reformatting is successful to generate a unified output format for the fields' content.
      ' Use the pField.FormattedText to save the reformatted information.
      ' You should then use pField.FormattedText also for the Export, instead of pField.Text
      pField.FormattedText = NewAmount
   end if
End Sub
```

4.3.5 PostAnalysis

<Field$_n$>_PostAnalysis (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)

pField

Object containing the Field

pWorkdoc

Current Workdoc object

The PostAnalysis event will be called after the analysis step has been performed. It is possible to examine the list of all candidates and to add further candidates to the Field.

Example:

```
Private Sub MyField_PostAnalysis(pField As SCBCdrField,
                                 pWorkdoc As SCBCdrWorkdoc)
   Dim cindex as long, count as long, id as long
   ' add a new candidate to the field
   if pWorkdoc.Wordcount > 42 then
      ' use the 42th word as new candidate
      count = 1  ' wordcount of new candidate
      id = 0     ' rule-id for later backtracing
      pField.AddCandidate 42, count, id, cindex
      ' cindex is the new index of the candidate
   end if
End Sub
```

4.3.6 PostEvaluate

<Field$_n$>_PostEvaluate (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)

pField

Object containing the Field

pWorkdoc

Current Workdoc object

The PostEvaluate event will be called after the evaluation step has been performed. It is possible to examine the list of all candidates and to change their weights. After this event is processed, the candidate with the highest weight will be chosen as the result for this Field.

Example:

```
Private Sub MyField_PostEvaluate(pField As SCBCdrField,
                                 pWorkdoc As SCBCdrWorkdoc)
   ' set the weight of the first candidate to 1
   if pField.CandidateCount > 0 then
      pField.Candidate(0).Weight = 1
   end if
End Sub
```

4.3.7 PreExtract

<Field$_n$>_PreExtract (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)

pField

Object containing the Field

pWorkdoc

Current Workdoc object

The PreExtract event will be called before any defined analysis or evaluation method for this Field is executed by the Cedar DocClass. During this event, it is possible to assign results to this Field and by changing the Field state, to skip defined analysis and evaluation methods.

Example:

```
Private Sub Today_PreExtract(pField As SCBCdrField,
pWorkdoc As SCBCdrWorkdoc)
' the field Today should contain the processing date of the document
   Dim today as date
   today = Date
   pField = Format(date, “yyyymmdd”)
End Sub
```

4.3.8 SmartIndex

<Field$_n$>_SmartIndex (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)

pField

Object containing the current Field

pWorkdoc

Current Workdoc objectThe smart index event is called each time after smart indexing can be performed for a certain Field. The event will be called for the Field where the smart indexing was defined. This field usually provides the key for the select statement. After the smart indexing has copied data to Fields, the validation step will be repeated for the modified Fields.

Example:

```
Private Sub CustomerNo_SmartIndex(pField As SCBCdrPROJLib.SCBCdrField, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc)
     ' avoid validation for the Name field if filled by smart indexing
     pWorkdoc.Fields("Name").Valid = TRUE
End Sub
```

4.3.9 TableHeaderClicked

<Field$_n$>_TableHeaderClicked (pTable As ISCBCdrTable, pWorkdoc As ISCBCdrWorkdoc, ClickType As CdrTableHeaderClickType, Row As Long, Column As Long, pSkipDefaultHandler As Boolean)

pTable

Current Table object

pWorkdoc

Current Workdoc object

ClickType

The click type of the mouse depend on the place where the click occurred either for the Column Header, Row Header or Table Header and which kind of click occurred either clicked, double-clicked, or right button clicked. Its values are described below:

CdrTableHeaderClickType	Value	Description
Description		
CdrColumnHeaderClicked	3	A column header button has been clicked
CdrColumnHeaderDoubleClicked	4	A column header button has been double-clicked
CdrColumnHeaderRightButtonClicked	5	The right mouse button has been pressed on a column header button
CdrRowHeaderClicked	0	A row header button has been clicked
CdrRowHeaderDoubleClicked	1	A row header button has been double-clicked
CdrRowHeaderRightButtonClicked	2	The right mouse button has been pressed on a row header button
CdrTableHeaderClicked	6	The table header button has been clicked
CdrTableHeaderDoubleClicked	7	The table header button has been double-clicked
CdrTableHeaderRightButtonClicked	8	The right mouse button has been pressed on the table header button

Error! It is not possible, through the processing of field functions to create objects.

Row

This parameter contains the value of the current row on which the user clicked.

Column

This parameter contains the value of the current column on which the user clicked.

pSkipDefaultHandler

The default value is FALSE. When the user wants to skip the default handling it has to be set to True.

For the code example the default handling is skipped, generally. When the user wants to have another handling for a special column, the parameter has to be set within the select statement to check, for example, the row number to disable the default handling or the force and additional handling.

This event occurs when a user clicks on one of the table header buttons. There are three different table header buttons: Row Header button, the Column Header button, or Table Header button.

Example:

```
Private Sub Table_TableHeaderClicked(pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, ByVal ClickType As SCBCdrPROJLib.CdrTableHeaderClickType, ByVal Row As
Long, ByVal Column As Long, pSkipDefaultHandler As Boolean)

   Select Case ClickType
      Case CdrColumnHeaderClicked
         ' Table column header button has been clicked -
         ' define your message handler here
      Case CdrColumnHeaderDoubleClicked
         ' Table column header button has been double clicked -
         ' define your message handler here
      Case CdrColumnHeaderRightButtonClicked
         ' Right mouse button has been clicked on table column header -
         ' define your message handler here
      Case CdrRowHeaderClicked
         ' Table row header button has been clicked -
         ' define your message handler here
      Case CdrRowHeaderDoubleClicked
         ' Table row header button has been double clicked -
         ' define your message handler here
      Case CdrRowHeaderRightButtonClicked
         ' Right mouse button has been clicked on table row header -
         ' define your message handler here
      Case CdrTableHeaderClicked
```

```
         ' Table header button has been clicked -
         ' define your message handler here
      Case CdrTableHeaderDoubleClicked
         ' Table header button has been double clicked -
         ' define your message handler here
      Case CdrTableHeaderRightButtonClicked
         ' Right mouse button has been clicked on table header -
         ' define your message handler here
   End Select

   ' Skip default handler of the table header clicked event
   ' (handler implemented in the Verifier component)
   pSkipDefaultHandler = True

End Sub
```

4.3.10 Validate

<Field$_n$>_Validate (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc, pValid As Boolean)

pField

Object containing the current Field

pWorkdoc

Current Workdoc object

pValid

Parameter containing the current valid state of the Field

The field Validate event can be used to perform project specific validation rules. Use the pValid parameter to return the validation decision. The default initialization of pValid is TRUE, so if the parameter remains unchanged or if the event is not implemented, the document state gets valid.

In case of using standard validation, the pValid parameter contains the result of the standard validation. Do not set pValid to TRUE if you want to keep the result of the standard validation.

Example:

```
Private Sub Number_Validate(pField As SCBCdrField,
                pWorkdoc As SCBCdrWorkdoc, pValid As Boolean)
   ' check result of standard validation
   if pValid = FALSE then
        ' standard validation returns invalid, stop here
        exit sub
   end if
   ' perform additional check for number format
   if IsValidNumber(pField) = FALSE then
        pValid = FALSE
        pField.ErrorDescription = "Field is not a valid number"
   end if
End Sub
```

4.3.11 ValidateCell

<Field$_n$>_ValidateCell (pTable As ISCBCdrTable, pWorkdoc As ISCBCdrWorkdoc, Row As Long, Column As Long, pValid As Boolean)

pTable

Current Table object

pWorkdoc

Current Workdoc object

Row

Given Row of the Table

Column

Given column of the Table

pValid

Parameter containing the current valid state of the Table cell.

This event method is called for each cell of the Table. Here you can implement validation checks specific for a single cell.

Example:

```
Private Sub MyTableField_ValidateCell(pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, ByVal Row As Long, ByVal Column As Long, pValid As Boolean)

   Select Case Column
   Case 0:
      ' check date in column 0
      if CheckDate(pTable.CellText(Column, Row)) = FALSE then
         pValid = FALSE
         pTable. CellValidationErrorDescription(Column, Row) = “Invalid date”
      end if
   Case 2:
      ' check order number in column 2
      if CheckOrderNumber(pTable.CellText(Column, Row)) = FALSE then
         pValid = FALSE
         pTable. CellValidationErrorDescription(Column, Row) = “Invalid order number”
      end if
   End Select
End Sub
```

4.3.12 ValidateRow

<Field$_n$>_ValidateRow (pTable As ISCBCdrTable, pWorkdoc As ISCBCdrWorkdoc, Row As Long, pValid As Boolean)

pTable

Table Object for which row is to be validated

pWorkdoc

Current Workdoc object

Row

Given row of the Table to be validated

pValid

Parameter containing the current valid state of the row

This event is called for each row of the Table if all of the cells of the row became valid beforehand. If there are still invalid cells inside a row the event will not be called until all cells becomes valid. You can use this event to implement validation rules, which combines two or more cells of a row.

Example:

```
Private Sub MyTableField_ValidateRow(pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, ByVal Row As Long, pValid As Boolean)
' check if quantity * single price = total price
Dim quantity as long
Dim s_price as double, t_price as double

' all cells must already have a valid format
   quantity = CLng(pTable.CellText("Quantity", Row))
   s_price = CLng(pTable.CellText("Single Price", Row))
   t_price = CLng(pTable.CellText("Total Price", Row))
   if quantity*s_price = t_price then
      pValid = TRUE
   else
      pValid = FALSE
      pTable.RowValidationErrorDescription(Row) = "Invalid quantity or amounts"
   end if
End Sub
```

4.3.13 ValidateTable

<FieldN>_ValidateTable (pTable As ISCBCdrTable, pWorkdoc As ISCBCdrWorkdoc, pValid As Boolean)

pTable

Table object

pWorkdoc

Current Workdoc object

pValid

Parameter containing the current valid state of the Table

This event is called for the Table when all rows are valid. It can be used to implement a validation rule for the entire Table.

Example:

```
Private Sub MyTableField_ValidateTable (pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, pValid As Boolean)
' calculate the sum of all amounts and compare with the net amount fields
Dim tablesum as double, netamount as double
Dim cellamount as double
Dim row as long
   For row = 0 to pTabler.RowCount-1
      cellamount = CLng(pTable.CellText("Total Price", Row))
      tablesum = tablesum + cellamount
   Next row
' now compare sum with the content of the net amount field
```

```
   netamount = CDbl(pWorkdoc.Fields(“NetAmount”).Text
   if netamount = tablesum then
      pValid = TRUE
   else
      pValid = FALSE
      pTable.TableValidationErrorDescription
      = “Sum of table amounts and field net amount are different”
   end if
End Sub
```

Chapter 5 Oracle Forms Recognition Workdoc Object Reference (SCBCdrWkDocLib)

5.1 SCBCdrWorkdoc

5.1.1 Interface of SCBCdrWorkdoc

Cedar Workdoc object

The Cedar Workdoc object stores all data of one Document. The amount of data grows during the processing steps of OCR, classification and extraction.

5.1.1.1 Type Definitions

CDREdgeSide

Enumeration containing the type of alignment/edges

CDREdgeLeft	Chooses left alignment (left edges) in analysis
CDREdgeRight	Chooses right alignment (right edges) in analysis

CDRHighlightMode

Enumeration containing the mode of highlighting

CDRHighlightAttractors	Attractor highlighting
CDRHighlightBlocks	Block highlighting
CDRHighlightCandidates	Candidates highlighting
CDRHighlightCandidatesAdvanced	Highlights only candidates but according to their advanced highlighting type, also fires all mouse events for all words
CDRHighlightCheckedWords	Verified words highlighting
CDRHighlightCheckedWordsAndCandidates	Verified words and candidate highlighting
CDRHighlightCheckedWordsAndField	Verified words and field highlighting
CDRHighlightCheckedWordsAndFields	Verified words and fields highlighting
CDRHighlightFields	Fields highlighting
CDRHighlightNothing	No highlighting
CDRHighlightParagraphs	Paragraph higlighting
CDRHighlightRectangles	Variable rectangle highlighting
CDRHighlightTables	Table highlighting
CDRHighlightTablesAdvanced	Highlights checked words and selected table cell, also shows tool-tips for all words and fires all mouse events for all words
CDRHighlightTextLines	Text lines highlighting
CDRHighlightTextLinesAdvanced	Highlights text lines according their block number, show tool-tips with line confidences, also fires all mouse events
CDRHighlightTrainedFields	Trained fields highlighting
CDRHighlightVerticalEdgesLeft	Left aligned edges highlighting
CDRHighlightVerticalEdgesRight	Right aligned edges highlighting
CDRHighlightWords	Word highlighting

CDRClassifyResult

Enumeration specifying the result of classification for a specific document class and specific classification engine. This is a cell inside the classification matrix.

CDRClassifyMaybe	Document may belong to DocClass but weights are not available
CDRClassifyNo	Document does not belong to this DocClass
CDRClassifyNotApplied	Classification engine is not applied to this DocClass
CDRClassifyWeighted	Classifiaction weight property has valid content
CDRClassifyYes	For sure document belongs to this DocClass

CDRDocState

Enumeration containing the state of the document

CDRDocStateAnalyzed	Document is analyzed
CDRDocStateBlocks	Blocks are analyzed in document
CDRDocStateClassified	Document is classified
CDRDocStateDeleted	Document is deleted
CDRDocStateEvaluated	Document is evaluated
CDRDocStateExported	Document is exported
CDRDocStateHaveDocs	Images or CIDocs are assigned to documents
CDRDocStateLanguage	Language detection executed
CDRDocStateReset	Intial state of document
CDRDocStateValid	Validity state of document
CDRDocStateWorktext	Worktext is assigned to document

CDRPageAssignment

Enumeration specifying how DocPages are assigned to the Workdoc.

CDRPageAssignAllPages	Assign all DocPages of Image or CIDoc to Workdoc
CDRPageAssignNewPage	First Page of Image or CIDoc appended as last DocPage to Workdoc
CDRPageAssignNoPage	No DocPages assigned to Workdoc

CDRPDFExportStyle

Enumeration containing the export type of PDF.

CDRPDF_ImgOnly	Export only Image to PDF
CDRPDF_ImgOnTxt	Export Image on top of text to PDF
CDRPDF_NoExport	No Export for single DocPage
CDRPDF_NoThumbnails	No thumbnail generated for DocPage
CDRPDF_TxtOnly	Export only text to PDF

CDRDocFileType

Enumeration containing the type of input file.

CDRDocFileTypeCroCIDoc	Cairo CIDocument
CDRDocFileTypeCroImage	Cairo Image object
CDRDocFileTypeRawText	Created from plain text without document
CDRDocFileTypeUnknown	Unknown file type, maybe attachment

5.1.1.2 List of methods and properties

AddDocFile
Adds file into Workdoc

AddField
Adds Field to Workdoc

AddHighlightRectangle
Adds new highlight rectangle

AnalyzeAlignedBlocks
Finds blocks that are left or right aligned

AnalyzeBlocks
Examines TextBlocks in Workdoc

AnalyzeEdges
Analyzes a document set of words

AnalyzeEdges2
Same as AnalyzeEdges method, but applies the processing for visible text lines only

AnalyzeParagraphs
Examines text paragraphs in Workdoc

AppendWorkdoc
Appends Workdoc at end of existing Workdoc

AssignDocToPage
Assign Page of Image or CIDoc to DocPage of Workdoc, zero-based indexing

AttractorColor
Sets / returns color used for attractor highlighting

BlockColor
Sets / returns color used for block highlighting

BlockCount
Returns number of TextBlocks of Workdoc

CandidateColor
Sets / returns color used for Candidate highlighting

Clear
Clear all stuff from Workdoc

ClearHighlightRectangles

Removes all highlight rectangles

ClsEngineConfidence

Sets / returns confidence level for a classification engine specified by its zero-based index

ClsEngineDistance

Sets / returns distance value for a classification engine specified by its index

ClsEngineResult

Sets / returns result values of Classification Result Matrix

ClsEngineWeight

Sets / returns classification weights within Classification Result Matrix

CreateFromWorktext

Creates Workdoc from text

CutPage

Cuts current Workdoc and generates new Workdoc from DocPages present after given zero-based PageIndex

DeleteFile

Deletes all *.wdc and corresponding *.tif files of Workdoc

DisplayPage

Sets / returns displayed DocPage by zero-based index of Workdoc in Viewer

DocClassName

Sets / returns name of DocClass to which the document was classified

DocFileCount

Returns number of documents Workdoc built from

DocFileName

Returns full pathname by zero-based index of document (image or text file) Workdoc built from

DocFileType

Returns FileType of document by zero-based index

DocState

Sets / returns current state of document

EdgeCount

Returns number of edges

ErrorDescription

Sets / returns error description

FieldColor

Sets / returns color used for highlighting of valid and invalid Fields

Fields
Returns Field Collection of document

Filename
Returns filename of Workdoc

Folder
Returns Folder Workdoc belongs to

FolderIndex
Returns FolderIndex of Folder Workdoc belongs to

GetEdge
Returns coordinates left, top and bottom of corners for an edge interpreted as a rectangle

HighlightCandidate
Sets / returns position of highlighted Candidate

HighlightField
Sets / returns position of highlighted Field

HighlightMode
Sets / returns mode of highlighting

Image
Returns Image object for specified zero-based index of Workdoc

IsPlainText
When setting indicates that worktext is plain text without geometry information

Language
Sets / returns language of document

LineColor
Sets / returns Color for line highlighting

Load
Loads file from given root path

PageCount
Returns number of displayable DocPages of Workdoc

Pages
Returns single DocPages of Workdoc by zero-based PageIndex

Paragraph

Returns paragraph at index

ParagraphCount
Returns number of paragraphs

PDFExport
Exports Workdoc to PDF

PDFGetInfoType
Returns type of PDF file exported

PDFSetInfoType
Sets type of export to PDF for specified zero-based PageIdx

ReadZone
Executes OCR on DocPage (zero-based index) for optionally specified Zone (offsets in percent)

Refresh
Refreshes Workdoc's DocPage currently shown in Viewer

RenameDocFile
Renames CIDoc or Image specified by zero-based DocIndex by NewName

ReplaceFirstImage
Replaces first image in Workdoc

Save
Saves Workdoc with given filename, corresponding DocFiles are saved relatively defined by ImageRootPath

ShowTooltips
Sets / returns if tool tips will be displayed when moving mouse pointer over displayed Workdoc

SkipTrainingWithEngine
Sets / returns whether the specified trainable engine has to skip this document in the training process

Table
Returns Table by zero-based index of Workdoc

TableCount
Returns number of Table objects stored within Workdoc

TextBlock
Returns TextBlock by zero-based index of Workdoc

Textline
Returns text line by zero-based index of Workdoc

TextlineCount

Returns number of text lines present in Workdoc

TrainedWithEngine

Indicates whether this document is trained with specified engine

UnloadDocs

Unloads all Images and CIDoc from Workdoc

Word

Returns Word specified by zero-based index from Workdoc

WordColor

Sets / returns color used for Word highlighting

WordCount

Returns number of Words of Workdoc

WordSegmentationChars

Sets / returns string containing characters used for Word segmentation

Worktext

Returns SCBCroWorktext object representing raw OCR results

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- AnalyzeLines
- GetVersionInfo
- GetVersionInfo2
- PrepareClassification
- SelectField
- SetPageCount
- VersionCount

5.1.2 Methods and properties of Interface SCBCdrWorkdoc

AddDocFile

```
AddDocFile (Path As String, FileType As CDRDocFileType,
         Assignment As CDRPageAssignment)
```

This method is used to add a given type of file (CIDoc, Image, raw text) into the Workdoc. How to assign Pages of a file are assigned to the Workdoc is judged by third parameter.

Path

Path of file

FileType

Type of file viz. CIDoc, Image etc.

Assignment

This parameter specifies how DocPages are assigned to the Workdoc

AddField

```
AddField (Name As String)
```

This method is used to add a Field to the Workdoc.

Be reminded that Fields that are added manually to the Workdoc must be defined in the Document class of the Project (->Insert Fielddefinition), otherwise this causes a script error.

Name

Name of the Field

AddHighlightRectangle

```
AddHighlightRectangle (Left As Long, Top As Long, Width As Long,
          Height As Long, PageNr As Long, Color As OLE_COLOR)
```

Adds a new highlight rectangle. Set HighlightMode SCBCDRHighlightRectangles to highlight all rectangles.

Left

Left of highlight rectangle

Top

Top of highlight rectangle

Width

Width of highlight rectangle

Height

Height of highlight rectangle

PageNr

Document page number of highlight rectangle

Color

Color of highlight rectangle

AnalyzeAlignedBlocks

```
AnalyzeAlignedBlocks (edgeSide As CDREdgeSide, leftAlignTolerance As Long, XDist
          As Double, YDist As Double, Join As Boolean,
          minDistance As Double)
```

This method splits the document into blocks that contains only left (or right) aligned lines. Using this method on a document with centered lines only will usually result in one block per line.

edgeSide

Determines whether left or right aligned blocks are to be found

leftAlignTolerance

The distance (in mm) that aligned lines might differ. Useful if document was scanned slightly tilted.

XDist

A value, depending on the font size of a word, which specifies, how far off an existing block a word may be to belong to that block. If its horizontal distance from the block is greater that XDist, then a new block is created

YDist

This value specifies (in mm) the maximum vertical distance for a word from a block. If its distance is greater that YDist, a new block is generated

Join

Specifies whether overlapping blocks are to be joined. Set to TRUE if you want to join them.

minDistance

This parameter is a factor to be multiplied with leftAlignTolerance. It specifies the minimal horizontal distance of two edges. Set this value 0 to ignore its effect.

AnalyzeBlocks

```
AnalyzeBlocks (XDist As Double, YDist As Double)
```

This method is used to determine all the TextBlocks of text present in a Workdoc which are minimum XDist apart from each other on X-axis and YDist apart from each other on Y-axis.

XDist

Minimum X distance between two TextBlocks

YDist

Minimum Y distance between two TextBlocks

AnalyzeEdges

```
AnalyzeEdges (edgeSide As CDREdgeSide, AlignTolerance As Double,
          YDist As Double, MinNoOfWords As Long, minDistance As Double, [pageNr As
          Long = TRUE])
```

Analyzes a document set of words that are, within a certain tolerance, aligned either right or left. Use Highlight mode (SCBCDRHighlightVerticalEdgesLeft or SCBCDRHighlightVerticalEdgesRight) to make the results visible.

edgeSide

Set this parameter to either CDREdgeLeft or CDREdgeRight to specify if you want edges that contain left or right aligned words.

AlignTolerance

This value (in mm) specifies how far the left (right) values of a words bounding rectangle may differ in order forit to still be considered aligned.

YDist

Specifies (in mm) how far two words may be apart vertically and still belong to the same edge.

MinNoOfWords

Specifies how many words have to belong to a valid edge. Edges, that contain less than MinNoOfWords after analyzing the document are deleted.

minDistance

This parameter is a factor to be multiplied with AlignTolerance. It specifies the minimal horizontal distance of two edges. Set this value 0 to ignore its effect.

pageNr

[optional,defaultvalue(-1)] Specifies the page to be analyzed for edges. Set to -1 (default) if analysis is needed for all pages.

AnalyzeEdges2

```
AnalyzeEdges2 (edgeSide As CDREdgeSide, AlignTolerance As Double,
          YDist As Double, MinNoOfWords As Long, minDistance As Double, PageNr As
          Long, vbCheckedOnly As Boolean)
```

Same as AnalyzeEdges method, but applies the processing for visible text lines only (in case 'vbCheckedOnly' parameter is set to TRUE, otherwise it works exactly like AnalyzeEdges).

edgeSide

Set this parameter to either CDREdgeLeft or CDREdgeRight to specify if you want edges that contain left or right aligned words.

AlignTolerance

This value (in mm) specifies how far the left (right) values of a words bounding rectangle may differ in order for it to still be considered aligned.

YDist

Specifies (in mm) how far two words may be apart vertically and still belong to the same edge.

minDistance

This parameter is a factor to be multiplied with AlignTolerance. It specifies the minimal horizontal distance of two edges. Set this value 0 to ignore its effect.

PageNr

Specifies the page to be analyzed for edges. Set to -1 (default) if analysis is needed for all pages.

vbCheckedOnly

If set to TRUE, the method applies processing for visible text lines only, otherwise this function works exactly like AnalyzeEdges.

AnalyzeParagraphs

```
AnalyzeParagraphs ()
```

This method is used to determine all the paragraphs present in a Workdoc.

AppendWorkdoc

```
AppendWorkdoc (pWorkdoc As ISCBCdrWorkdoc)
```

This method is used to append a given Workdoc to the existing Workdoc.

pWorkdoc

Workdoc that is to be appended

AssignDocToPage

```
AssignDocToPage (DocIndex As Long, DocPage As Long, WorkdocPage As Long)
```

This method should be used to assign a Page of an Image or CIDoc to a certain DocPage of the Workdoc. This method requires that there are already documents inserted to the Workdoc using the AddDocFile function and the SetPageCount function must be called before.

DocIndex

Zero-based CIDoc or Image Index

DocPage

Zero-based DocPage inside the Image or CIDoc

WorkdocPage

Zero-based DocPage inside the Workdoc

AttractorColor

```
AttractorColor As OLE_COLOR (read/write)
```

Sets / returns the color that will be used for attractor highlighting.

BlockColor

```
BlockColor As OLE_COLOR (read/write)
```

Sets / returns the color which will be used for block highlighting.

BlockCount

```
BlockCount As Long (read only)
```

This read only property returns the number of TextBlocks of the Workdoc. Use this property before accessing the TextBlock property where an index is required. The range of valid indices for TextBlocks is from 0 to BlockCount –1.

CandidateColor

```
CandidateColor As OLE_COLOR (read/write)
```

Sets / returns the color which will be used for Candidate highlighting.

Clear

```
Clear ()
```

This method is used to clear all the memories and to remove all the documents from Workdoc. This will leave the Workdoc in an initial state.

ClearHighlightRectangles

```
ClearHighlightRectangles ()
```

Removes all highlight rectangles.

ClsEngineConfidence

```
ClsEngineConfidence (lMethodIndex As Long) As Long (read/write)
```

Sets / returns confidence level for a classification engine specified by its index in collection of classification engines.

lMethodIndex

Zero-based engine index in collection of classification engines.

ClsEngineDistance

```
ClsEngineDistance (lMethodIndex As Long) As Long (read/write)
```

Sets / returns distance value for a classification engine specified by its index in collection of classification engines.

lMethodIndex

Zero-based engine index in collection of classification engines.

ClsEngineResult

```
ClsEngineResult (MethodIndex As Long, DocClassIndex As Long)
         As CDRClassifyResult (read/write)
```

Provides access to classification result matrix. This matrix will be used during the classification step to store the results of each used classification method for each document class (DocClass) of the project. The matrix has one column for each classification method and one column for the combined result of all methods. A row contains the results for a single DocClass, therefore there will be one row for each DocClass in the classification matrix. The matrix will be created during the classification step, but not saved to disk. After reloading the Workdoc, the matrix is no longer available.

See *CDRClassifyResult* type definition for further information.

MethodIndex

MethodIndex = 0 can be used to access the voted result of all classification methods. A MethodIndex of 1 - n can be used to access the results of the single classification methods. The sorting of the classification methods within the array is determined by the Collection of classification settings of the Project. You can access this Collection from the script as Project.ClassifySettings which has a type of SCBCroCollection. Use the Count property to get the number of used classification engines or use the ItemIndex / ItemName property to find the index of classification method or the name for an index.

DocClassIndex

The DocClassIndex is determined by the Collection of all DocClasses. You can access this Collection from script as Project.AllClasses which has a type of SCBCroCollection. Use the Count property to get the number of DocClasses or use the ItemIndex / ItemName property to find the index of DocClass or the name for an index.

ClsEngineWeight

```
ClsEngineWeight (MethodIndex As Long, DocClassIndex As Long)
         As Double (read/write)
```

Provides access to the classification weights within the Classification Result Matrix. The classification weight is valid if the ClsEngineResult property of the same cell is 'CDRClassifyWeighted'. The Classification Matrix will be used during the classification step to store the results of each used classification method for each document class (DocClass) of the project. It has one column for each classification method and one column for the combined result of all methods. The matrix will be created during classification step, but not saved to disk. After reloading the Workdoc, the matrix is no longer available.

A row contains the result for a single DocClass, therefore there will be one row for each DocClass in the classification matrix.

MethodIndex

MethodIndex = 0 can be used to access the voted result of all classification methods. A MethodIndex of 1 - n can be used to access the results of the single classification methods. The sorting of the classification methods within the array is determined by the Collection of classification settings of the Project. You can access this Collection from the script as Project.ClassifySettings which has a type of SCBCroCollection. Use the Count property to get the number of used classification engines or use the ItemIndex / ItemName property to find the index of classification method or the name for an index.

DocClassIndex

The DocClassIndex is determined by the collection of all document classes. You can access this Collection from script as Project.AllClasses that are a type of SCBCroCollection. Use the Count property to get the number of DocClasses or use the ItemIndex / ItemName property to find the index of DocClass or the name for an index.

CreateFromWorktext

```
CreateFromWorktext (pWorktext As ISCBCroWorktext)
```

This method is used to create Workdoc from the OCRed text of an Image.

pWorktext

Object pointer of Worktext.

CutPage

```
CutPage (PageIndex As Long, ppNewWorkdoc As ISCBCdrWorkdoc)
```

This method cuts the current Workdoc and generates a new Workdoc from DocPages present after the given PageIndex.

PageIndex

[in] Zero-based index of DocPage after which the Workdoc has to be cut

ppNewWorkdoc

[out] New Workdoc generated as part of the current Workdoc

DeleteFile

```
DeleteFile (DeleteDocFiles As Boolean)
```

This method is used to delete all wdcs and corresponding tifs of the Workdoc.

DeleteDocFiles

Flag to inform whether to delete files or not

DisplayPage

```
DisplayPage As Long (read/write)
```

Sets / returns the displayed DocPage specified by zero-based index of the Workdoc in the Viewer.

DocClassName

```
DocClassName As String (read/write)
```

Sets / returns the name of the DocClass to which the document was classified. During the classification events PreClassify and PostClassify the property can be used to assign a DocClass to the current processed document by writing the name of a DocClass into the property.

If the classification could not be performed and no default class is defined for the Project the property will be empty.

DocFileCount

```
DocFileCount As Long (read only)
```

Returns the number of documents from which the Workdoc was built from. Typically a document is created from one multi-image document or from several single-page documents.

DocFileName

```
DocFileName (index As Long) As String (read only)
```

Returns the full pathname of a document (image or text file) the Workdoc was built from. The mapping from DocPages of a Workdoc to the attached documents can be retrieved using the DocIndex and DocPageIndex properties of the SCBCdrDocPage entries of the Workdoc.

If a Workdoc was created from a single document (e.g., Multi Tiff), the name of the document file can be retrieved accessing the index 0.

```
Path = pWorkdoc.DocFileName(0)
```

index

The index parameter has a valid range from 0 to DocFileCount-1.

DocFileType

```
DocFileType (index As Long) As CDRDocFileType (read only)
```

This property returns the file type of the document by the specified index.

index

The index parameter has a valid range from 0 to DocFileCount-1.

DocState

```
DocState As CDRDocState (read/write)
```

Sets / returns the current state of the document. For example after analyzing the TextBlocks in a document. The document state is changed to CDRDocStateBlocks.

EdgeCount

```
EdgeCount (edgeSide As CDREdgeSide) As Long (read only)
```

Returns the number of vertical edges found in a document.

edgeSide

Flag to distinguish between left and right edges.

ErrorDescription

```
ErrorDescription As String (read/write)
```

Sets / returns an error description.

FieldColor

```
FieldColor (FieldValid As Boolean) As OLE_COLOR (read/write)
```

Sets / returns the color which will be used for highlighting of valid and invalid Fields.

FieldValid

If set to TRUE it specifies the color for valid Fields or it specifies the color for invalid Fields.

Fields

```
Fields As ISCBCdrFields (read only)
```

The Fields Collection provides access to all Fields of a document. See the definition of SCBCdrFields for details of the Fields Collection and the definition of SCBCdrField for details for accessing the properties of a Field.

To read the text content of a simple Field use the following command:

```
Dim FieldContent as string
FieldContent = pWorkdoc.Fields.Item("MyField").Text
```

Because Item is the default property of the Fields Collection and Text is the default property of the Field the following shorter line has the same effect:

```
FieldContent = pWorkdoc.Fields("MyField")
```

Filename

```
Filename As String (read only)
```

This property contains the name of the file where the Workdoc itself was loaded from. This is typically filename with *.wdc extension. This file does not contain the Image or text-based document the

Workdoc was build from. Use the DocFileName property to access the Image or text files that the Workdoc was created from.

Folder

```
Folder As ISCBCdrFolder (read only)
```

Workdocs can be grouped to Folders by the Batch import program. This property can be used to access the Folder to which the Workdoc belongs. It is possible to store application-specific data inside the folder or even access other Workdocs in the Folder.

FolderIndex

```
FolderIndex As Long (read only)
```

The FolderIndex property provides the index of Folder a Workdoc belongs to.

GetEdge

```
GetEdge (edgeSide As CDREdgeSide, edgeIndex As Long, pLeft As Long,
         pTop As Long, pBottom As Long, pPageNr As Long)
```

Returns the coordinates left, top and bottom of the corners for an edge, which is interpreted as a rectangle.

edgeSide

Set this parameter to either CDREdgeLeft or CDREdgeRight to specify if you want edges that contain left or right aligned words.

edgeIndex

Index of the edge to be returned, valid indices are from 0 to the result of EdgeCount – 1

pLeft

Contains left coordinate of the edge.

pTop

Contains top coordinate of the edge.

pBottom

Contains bottom coordinate of the edge.

pPageNr

Contains page number of the edge.

HighlightCandidate

```
HighlightCandidate As Long (read/write)
```

Set / returns the position of highlighted Candidate.

HighlightField

```
HighlightField As Long (read/write)
```

Sets / returns the position of the highlighted Field.

HighlightMode

```
HighlightMode As CDRHighlightMode (read/write)
```

This property is used to set / return the current mode of highlighting. If TextBlocks are highlighted in a Workdoc then CDRHighlightBlocks will be returned by this property.

Image

```
Image (index As Long) As ISCBCroImage (read only)
```

Returns an Image object for the specified DocPage of the Workdoc. If the specified DocPage was based on a text based document, the text will be rendered to a new created SCBCroImage or a reference to an Image will be provided.

index

Index of the DocPage which is valid from 0 to PageCount - 1.

IsPlainText

```
IsPlainText As Boolean (read/write)
```

This boolean property is used to set or return if worktext is plain text or not. True will indicates that the worktext contains no geometry information.

Language

```
Language As String (read/write)
```

Sets / returns the language of the document, as it was specified by the language detection or the default language of the Project.

LineColor

```
LineColor As OLE_COLOR (read/write)
```

Sets / returns the Color which will be used for line highlighting.

Load

```
Load (Filename As String, ImageRootPath As String)
```

This method is used to load a file from given root path and this root path is not the absolute path of the file.

Filename

Name of the file.

ImageRootPath

Relative path of the file.

PageCount

```
PageCount As Long (read only)
```

Returns the number of displayable DocPages of the Workdoc.

Pages

```
Pages (PageIndex As Long) As ISCBCdrDocPage (read only)
```

Returns the single DocPages of the Workdoc. See the definition of SCBCdrDocPage for details for accessing the properties of a DocPage.

PageIndex

Index of the DocPage to access, which is valid from 0 to PageCount-1.

Paragraph

```
Paragraph (index As Long) As ISCBCdrTextBlock (read only)
```

This method provides access to the paragraph array of the Workdoc. This method should only be applied to documents like business letters or anything that actually has paragraphs.

index

Specifies the index of the paragraph. Valid indexes are from 0 to the result of ParagraphCount – 1

ParagraphCount

```
ParagraphCount As Long (read only)
```

Returns the number of paragraphs in the document. Use get_paragraph to retrieve any one of them. This method should only be applied to documents like business letters or anything that actually has paragraphs.

PDFExport

```
PDFExport (FileName As String)
```

This method is used to generate a PDF file from Workdoc based on the given export type.

FileName

Name of the PDF file exported.

PDFGetInfoType

```
PDFGetInfoType (PageIdx As Long, pExportStyle As CDRPDFExportStyle)
```

This method returns the export type of given Page of PDF file.

PageIdx

Page number of PDF.

pExportStyle

Type of Export.

PDFSetInfoType

```
PDFSetInfoType (PageIdx As Long, ExportStyle As CDRPDFExportStyle)
```

This method is used to set the type of export of PDF. You can export only Image to PDF, text to PDF, or both Image and text to PDF.

PageIdx

Zero-based DocPage Number.

ExportStyle

Type of export

ReadZone

```
ReadZone (PageIndex As Long, [left As Double = FALSE],
          [top As Double = FALSE], [right As Double = 1],
          [bottom As Double = 1])
```

The ReadZone method is a part of the OCR-on-demand concept. Due to the high runtime costs of Optical Character Recognition, the execution of OCR can be delayed until the entire text of the document is really required. After execution, all text information for the specified region is available.

PageIndex

Specifies the DocPage where the OCR or text conversion should be executed. Valid indices are 0 to PageCount - 1 for working on single pages or -1 for executing OCR on all DocPages.

left

[in,optional,defaultvalue(0)] Specifies a left offset for the OCR region in percent. Use 0 here to read from the left border.

top

[in,optional,defaultvalue(0)] Specifies the top offset for the OCR region in percent. Use 0 here to read from the top border.

right

[in,optional,defaultvalue(1)] Specifies the right border of the OCR region in percent. Use 100 here to read until the right border.

bottom

[in,optional,defaultvalue(1)] Specifies the bottom line of the OCR region in percent. Use 100 here to read until the bottom border.

Refresh

```
Refresh ()
```

Refreshes the Workdoc's DocPage which is currently shown in the Viewer.

RenameDocFile

```
RenameDocFile (DocIndex As Long, NewName As String)
```

This method is used to change the name of the CIDoc or Image at given DocIndex by the given new name.

DocIndex

Specifies the zero-based CIDoc or Image Index.

NewName

New name given to the document at DocIndex.

ReplaceFirstImage

```
ReplaceFirstImage (Path As String)
```

This method replaces first image in Workdoc.

Path

Image path to replace the existing workdoc's image with.

Save

```
Save (Filename As String, ImageRootPath As String)
```

This method is used to save a Workdoc with given filename and its DocFiles relatively at given ImageRootPath. If the files are saved in the same directory give an empty string as ImageRootPath

Filename

Filename of Workdoc

ImageRootPath

Relative path where all corresponding DocFiles are saved, empty if files are saved in the same directory as the Workdoc.

ShowTooltips

```
ShowTooltips As Boolean (read/write)
```

Sets / returns if tool tips will be displayed when moving the mouse pointer over the displayed Workdoc.

SkipTrainingWithEngine

```
SkipTrainingWithEngine (bstrEngineName As String) As Boolean (read/write)
```

This property identifies whether the specified trainable engine has to skip this document in the training process.

bstrEngineName

Name of classification engine.

Table

```
Table (index As Long) As ISCBCdrTable (read only)
```

Returns a Table for given index of the Workdoc.

index

Specifies the index of the Table. Valid indices are from 0 to TableCount-1.

TableCount

```
TableCount As Long (read only)
```

Returns the number of Table objects stored within the Workdoc.

TextBlock

```
TextBlock (index As Long) As ISCBCdrTextBlock (read only)
```

Returns TextBlock by index of the Workdoc. The range of valid indices for TextBlocks is from 0 to BlockCount -1.

index

[in] Specifies the index of the TextBlock. Valid indices are from 0 to BlockCount-1.

Textline

```
Textline (index As Long) As ISCBCdrTextBlock (read only)
```

Returns text line by index of the Workdoc. The range of valid indices for TextLine is from 0 to LineCount -1.

index

Zero-based index.

TextlineCount

```
TextlineCount As Long (read only)
```

This method is used to retrieve the number of text lines present in a Workdoc

TrainedWithEngine

```
TrainedWithEngine (bstrEngineName As String) As Boolean (read only)
```

This property indicates whether this document is trained with the specified engine.

bstrEngineName

Name of engine.

UnloadDocs

```
UnloadDocs ()
```

This method is used to release all the Images and CIDocs which belong to this Workdoc.

Word

```
Word (index As Long) As ISCBCdrWord (read only)
```

This property provides access to the Word array of the Workdoc. The Words are implemented as SCBCdrWord objects which contain the text, the geometric position and DocPage of the Word.

index

[in] Index of the requested Word. Valid indices are from 0 to WordCount-1.

WordColor

```
WordColor As OLE_COLOR (read/write)
```

Sets / returns the color that will be used for Word highlighting.

WordCount

```
WordCount As Long (read only)
```

Returns the number of Words of the Workdoc. This value can be 0 if no OCR was performed before. Be sure to have the OCR performed before reading this property. Typically OCR will be requested from most classification and extraction methods. To ensure the availability of OCR results, call the ReadZone method before reading the WordCount property.

WordSegmentationChars

```
WordSegmentationChars As String (read/write)
```

Sets / returns a string which contains the characters used for the segmentation of Words, for example: '-;/'. These characters are used during OCR for Word segmentation.

Worktext

Worktext As ISCBCroWorktext (read only)

This property provides access to the raw OCR results represented by the SCBCroWorktext object. See the description of the SCBCroWorktext object for more details.
Be sure to have the OCR performed before reading this property. Typically OCR will be requested from most classification and extraction methods. To ensure the availability of OCR results, call the ReadZone method before reading the Worktext property.

5.1.3 E-mail Importing Interface

When importing a MSG file into a Workdoc, the most important properties of the e-mail are stored in the Workdoc and available in the custom script via a new "**ISCBCdrEmailProperties**" interface that can be queried from the SCBCdrWorkdoc interface.

5.1.3.1 List of method and properties

Subject

E-mail subject.

From

List of e-mail senders.

To

List of e-mail recipients.

CC

List of carbon copy recipients.

Sent

Date and time the e-mail was sent at.

Received

Date and time the e-mail was received at.

Priority

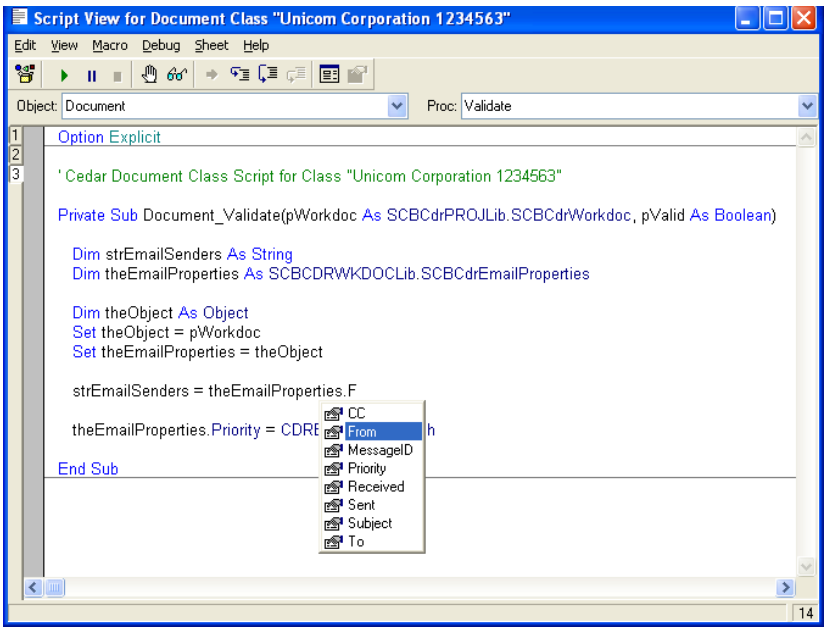
Priority the e-mail was sent with.

MessageID

Unique message identifier.

5.1.3.2 Usage and Script Sample

The screenshot below shows how the new “SCBCdrEmailProperties” interface can be used in script.



Picture 5.1.3.2. Usage of new E-mail Properties interface in [WL PN] custom script.

Here is also a script sample you could use for testing. Please make a note of the way the new “E-mail Properties” interface is queried from the “Workdoc” interface. The point is both old SAX and new WinWrap scripting engines do not support “.Object” property, i.e. the usual Visual Basic way to query an interface does not work for these scripting engines. The script sample below shows a workaround that is applicable for both SAX and WinWrap Basic engines.

```
Private Sub ScriptModule_PreClassify(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)

   Dim strEmailSenders As String
```

```
    Dim theEmailProperties As SCBCDRWKDOCLib.SCBCdrEmailProperties

    ' Queries "Email Properties" interface from the main "Workdoc" interface
    Dim theObject As Object
    Set theObject = pWorkdoc
    Set theEmailProperties = theObject

    ' Accessing of e-mail properties stored in Workdoc
    strEmailSenders = theEmailProperties.From

End Sub
```

5.2 SCBCdrField

5.2.1 SCBCdrField Interface (Cedar Field Object)

This object contains the data that are evaluated and that should be extracted from the Document.

5.2.1.1 Type Definitions

CDRFieldState

Enumeration containing the state of the Field.

CDRFieldStateAnalyzed	Field is analyzed
CDRFieldStateEvaluated	Field is evaluated
CDRFieldStateFormated	Field is formatted
CDRFieldStateReset	Initial state of a Field
CDRFieldStateValid	Validity state of Field

5.2.1.2 List of methods and properties

ActiveTableIndex

Activates / deactivates Table by given zero-based index.

AddCandidate

Adds new Candidate to Field based on specified zero-based WordNr

AddCandidate2

Adds new Candidate to Field based on specified Worktext

AddTable

Adds Table

Candidate

Returns Candidate by zero-based index of Field

CandidateCount

Returns number of Candidates of Field

Changed

Returns if Field changed

DeleteLine

Deletes a line specified by zero-based LineIndex

DeleteTable

Deletes Table at specified zero-based TableIndex

ErrorDescription

Sets / returns error description why previous validation fails

FieldID

Returns internally used FieldID

FieldState

Sets / returns current execution state of Field

FindCandidate

Searches for zero-based WordID inside list of Candidates

FormattedText

Sets / returns Formatted text

GetUniqueEntryId

Unique ID (64 bit) for the field content returned from associative search pool

Height

Sets / returns height of Field in pixel

InsertLine

Inserts line in Field, zero-based LineIndex

Left

Sets / returns left border of Field in pixel

Line

Sets / returns text of a single line by zero-based index

LineCaption

Sets / returns caption to line by zero-based index

LineCount

Sets / returns number of lines

LineWorktext

Sets / returns Worktext object of each single line of field by zero-based index

MultilineText

Sets / returns Multiline text

Name
Returns name of Field

PageNr
Sets / returns DocPage number of Field location

PutUniqueEntryId
Sets unique ID (64 bit) for the field content from associative search pool

RemoveCandidate
Removes Candidate by zero-based index from Candidate array

SkipTrainingWithEngine
Sets / returns whether the specified trainable engine has to skip this field in the training process

Table
Returns Table object from Table array by zero-based index

TableCount
Returns number of Tables of Field

Tag
Used to store / receive arbitrary variant in Field

Text
Sets / returns text of Field

Top
Sets / returns top border of Field in pixel

TrainedWithEngine
Returns if this field is trained with the specified engine

Valid
Sets / returns valid state of Field

Width
Sets / returns width of Field in pixel

Worktext
Sets / returns Worktext object of Field

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- FieldVersion
- SelectCharacter

5.2.2 Methods and properties of Interface SCBCdrField

ActiveTableIndex

```
ActiveTableIndex As Long (read/write)
```

This property can read the position where the Table is activated or activate the Table at given zero-based index.

AddCandidate

```
AddCandidate (WordNr As Long, WordCount As Long, FilterID As Long, pIndex As Long)
```

Adds a new Candidate to the Field based on the specified Word ID.

WordNr

Specifies the Word index within the Word array of the Workdoc. Must be within 0 to pWorkdoc.WordCount - 1.

WordCount

[in] Specifies the number of Words to use for the Candidate. If WordCount is greater than 1 the second word for the Candidate is defined with WordNr + 1, the third with WordNr + 2.

FilterID

[in] This parameter can be used to store a filter identifier inside the Candidate. So later it is possible to see which filter expression has created the Candidate.

pIndex

[out] Returns the index of the new Candidate within the Candidate array.

AddCandidate2

```
AddCandidate2 (pWorktext As ISCBCroWorktext, pIndex As Long)
```

Adds a new Candidate to the Field based on the specified Worktext.

pWorktext

[in] Must be an initialized Worktext as it was created calling a SCBCroZone.Recognize method.

pIndex

[out] Returns the index of the new Candidate within the Candidate array.

AddTable

```
AddTable ()
```

This method is used to add a Table into the Table array of this Field.

Candidate

```
Candidate (index As Long) As ISCBCdrCandidate (read only)
```

Returns a Candidate of the Field.

index

Index of the Candidate. Valid indices are 0 to CandidateCount-1.

CandidateCount

```
CandidateCount As Long (read only)
```

Returns the number of Candidates of the Field.

Changed

```
Changed As Boolean (read/write)
```

Returns the changed state of the Field. If the changed state becomes TRUE the field must be validated even if it was already validated before. The Changed property becomes FALSE after the validation method of a Field was executed. The Changed property becomes TRUE if the text or Worktext property was modified.

DeleteLine

```
DeleteLine (LineIndex As Long)
```

This method is used to delete a line from specific index position.

LineIndex

Index of Line, zero-based indexing

DeleteTable

```
DeleteTable (TableIndex As Long)
```

Deletes a Table from the Table array of this Field.

TableIndex

Zero-based Index of the Table.

ErrorDescription

```
ErrorDescription As String (read/write)
```

This string contains a possible error description as to why a previous validation fails. Use this property to store the reason if a script validation could not be performed successfully.

FieldID

```
FieldID As Long (read only)
```

This read-only property returns the internally used FieldID.

FieldState

```
FieldState As CDRFieldState (read/write)
```

Sets / returns the current execution state of the Field.

FindCandidate

```
FindCandidate (WordID As Long, pCandIndex As Long)
```

This method searches inside the list of Candidates if there is a Candidate based on the specified WordID. This method can be used to avoid the insertion of duplicate Candidates by similar search expressions.

WordID

[in] Specifies a WordID inside the Word array of the Workdoc searched for.

pCandIndex

[out] Contains the index of the Candidate if someone was found or -1 if no Candidate was found.

FormattedText

```
FormattedText As String (read/write)
```

This property sets or returns formatted text.

Example:

```
Private Sub Amount_Format(pField As SCBCdrField)
   Dim NewAmount as string
   if MyReformatAmount(pField, NewAmount) = TRUE then
      ' reformatting was successful, change the field
      ' writing to pField.FormattedText, then the information read
      ' within the document will be kept. You should then use
      ' pField.FormattedText also for the Export, instead of
      ' pField.Text
      pField.FormattedText = NewAmount
   end if
End Sub
```

GetUniqueEntryId

```
GetUniqueEntryId (IdHigh As Long, IdLow As Long)
```

Unique ID (64 bit) for the field content returned from associative search pool. This unique ID can also be used to retrieve other column values for the specified pool entry.

IdHigh

[out] Upper part of the 64-bit unique ID.

IdLow

[out] Lower part of the 64-bit unique ID.

Height

```
Height As Long (read/write)
```

Sets / returns the height of the Field in pixel.

InsertLine

```
InsertLine (LineIndex As Long)
```

This method is used to insert a line at given LineIndex in a Field.

LineIndex

Zero-based LineIndex at which position line has to be inserted

Left

```
Left As Long (read/write)
```

Sets / returns the left border of the Field in pixel.

Line

```
Line (index As Long) As String (read/write)
```

Sets / returns the text of a single line. A Field can consist of one or more lines. Typically multi-line Fields will be used for address search and other extraction methods. This property allows you to read or write single lines of the Field.

index

Index of the line must be from 0 to LineCount-1.

LineCaption

```
LineCaption (index As Long) As String (read/write)
```

If a Field has more than one line, it is possible to assign a caption to each line to provide information about the content of the line.

index

Index of the line, must be from 0 to LineCount-1.

LineCount

```
LineCount As Long (read/write)
```

Returns the number of lines of the Field or can be used to set the number of lines of a Field.

LineWorktext

```
LineWorktext (index As Long) As ISCBCroWorktext (read/write)
```

This property provides access to the Worktext of each single line of the Field.

index

Index of the line, must be from 0 to LineCount-1.

MultilineText

```
MultilineText As String (read/write)
```

This property sets or returns multiline text.

Name

```
Name As String (read only)
```

Returns the name of the Field as it was defined within the design environment.

PageNr

```
PageNr As Long (read/write)
```

Sets / returns the DocPage number where the Field is located.

PutUniqueEntryId

```
PutUniqueEntryId (IdHigh As Long, IdLow As Long)
```

Sets the unique ID (64 bit) for the field content from associative search pool.

IdHigh

[in] Upper part of the 64-bit unique ID.

IdLow

[in] Lower part of the 64-bit unique ID.

RemoveCandidate

```
RemoveCandidate (CandIndex As Long)
```

This method can be used to remove a Candidate from the Candidate array. The CandidateCount property will be decremented during the method call.

CandIndex

Zero-based Candidate Index.

SkipTrainingWithEngine

```
SkipTrainingWithEngine (bstrEngineName As String) As Boolean (read/write)
```

This property identifies whether the specified trainable engine has to skip this field in the training process.

bstrEngineName

Name of the extraction engine.

Table

```
Table (index As Long) As ISCBCdrTable (read only)
```

This property is used to retrieve the Table object from an array of Tables of this Field at a specified index.

index

Position of a Table in an array of Tables, zero-based indexing

TableCount

```
TableCount As Long (read only)
```

Returns the number of Tables according to the Field.

Tag

```
Tag As Variant (read/write)
```

The tag property can be used to store an arbitrary variant in the Field. The variant will not be saved when the Workdoc is saved to disk, but it is possible to store information across the different script events.

Text

```
Text As String (read/write)
```

The Text property can be used to read and write the text of the Field. In case of multi-line Fields, the Text property refers only to the first line of the Field. Use the Line property to access other lines than the first one.

Top

```
Top As Long (read/write)
```

Sets / returns the top border of the Field in pixel.

TrainedWithEngine

```
TrainedWithEngine (bstrEngineName As String) As Boolean (read only)
```

This property indicates whether this field is trained with the specified engine.

bstrEngineName

Name of the Engine

Valid

```
Valid As Boolean (read/write)
```

Sets / returns the valid state of the Field.

Width

```
Width As Long (read/write)
```

Sets / returns the width of the Field in pixel.

Worktext

```
Worktext As ISCBCroWorktext (read/write)
```

This property provides access to the Worktext of the Field. In case of multi-line Fields, the Worktext property refers only to the first line of the Field. Use the LineWorktext property to access other lines than the first one.

5.3 SCBCdrFields

5.3.1 SCBCdrFields Interface (Cedar Fields object)

Collection of all Field objects contained in the current WorkDoc object.

5.3.1.1 List of Methods and Properties

Add
Adds new Field with specified name to Field Collection

Clear
Removes all items from Collection

Collection
Returns Collection internally used to store Fields

Count
Returns item count of Field Collection

Item
Returns item from Collection specified by index (ID [1-Count] or name)

ItemByIndex
Returns item from Collection specified by index [1-Count]

ItemByName
Returns item by specified name

ItemExists
Returns TRUE if item with specified name exists

ItemIndex
Returns index of item specified by name [1-Count]

ItemName
Returns name of item specified by index [1-Count]

MoveItem
Moves item specified by OldIndex from OldIndex to NewIndex [1-Count]

Remove

Removes item specified by name from Collection

RemoveByIndex

Removes item specified by index [1-Count] from Collection

Rename

Renames item specified by Oldname from OldName to NewName

Tag

Tag to store arbitrary data during runtime specified by Index [1-Count]

5.3.2 Methods and Properties of SCBCdrFields Interface

Add

```
Add (NewItem As ISCBCdrField, ItemName As String) As Long
```

This method adds a new Field with the specified name to the Field Collection. The internal Collection fires an OnChange Event. Its return value is the index of the inserted item inside the Collection. This index can be used to access the item inside the Collection.

NewItem

[in] Pointer to a SCBCdrField object which should be added to the Collection.

ItemName

[in] Name of the Field item inside the Collection. This name must be used to access the item inside the Collection.

Clear

```
Clear ()
```

Removes all items from the Collection and releases their reference count. If there are no other references to the Collection items, this can free all items from memory. The internal Collection fires an OnChange Event.

Collection

```
Collection As ISCBCroCollection (read only)
```

Returns the Collection which is internally used to store the Fields. The access to the Collection may be necessary if you want to connect to the Collection to receive the change events.

Count

```
Count As Long (read only)
```

Returns the number of items within the Field Collection.

Item

```
Item (Index As Variant) As ISCBCdrField (read only)
```

These read-only properties returns a specified item from the Collection. The Item property is the default property of the ISCBCdrFields Collection.

Index

The index can either be a long value specifying the index within the collection, valid range from 1 to Count, or a string specifying the item by name.

ItemByIndex

```
ItemByIndex (Index As Long) As ISCBCdrField (read only)
```

Returns an item from the Collection specified by index.

Index

Index of the item to retrieve from the Collection, valid range from 1 to Count

ItemByName

```
ItemByName (Name As String) As ISCBCdrField (read only)
```

Returns the Field from the Collection by the specified Field name.

Name

[in] Name of the item to retrieve from the Collection.

ItemExists

```
ItemExists (Name As String) As Boolean
```

Returns TRUE if an item with the specified name exists inside the Collection or FALSE is returned.

Name

Name of item to search for.

ItemIndex

```
ItemIndex (Name As String) As Long (read only)
```

The index of an item specified by name is returned.

Name

Name specifying an item in the Collection.

ItemName

```
ItemName (Index As Long) As String (read only)
```

The name of an item is returned specified by index.

Index

Index specifying an item in the collection, valid range from 1 to Count

MoveItem

```
MoveItem (OldIndex As Long, NewIndex As Long)
```

Moves an item specified by OldIndex from OldIndex to NewIndex. This method causes that all items between OldIndex and NewIndex also change their index after the operation has succeeded. The internal Collection fires an OnChange Event.

OldIndex

[in] Index of item to remove, valid range from 1 to Count

NewIndex

[in] New index of the item after the move has occurred, valid range from 1 to Count

Remove

```
Remove (ItemName As String)
```

Removes the specified item from the Collection and releases the reference count to this item. If there are no other references to the item, this can cause to free this item from memory. The internal Collection fires an OnChange Event.

ItemName

[in] Name of item to remove.

RemoveByIndex

```
RemoveByIndex (Index As Long)
```

Removes the specified item from the Collection and releases the reference count to this item. If there are no other references to the item, this can free this item from memory. The internal Collection fires an OnChange Event.

Index

[in] Index of item to remove, valid range from 1 to Count

Rename

```
Rename (OldName As String, NewName As String)
```

Renames the item specified by Oldname from OldName to NewName. The internal Collection fires an OnChange Event.

OldName

[in] Name of item to rename

NewName

[in] New name of item in Collection.

Tag

```
Tag (Index As Long) As Variant (read/write)
```

This tag can be used to store a variant for each item of the Collection. This value will not be saved with the Workdoc.

Index

Specifies the item index, valid range from 1 to Count.

5.4 SCBCdrCandidate

5.4.1 SCBCdrCandidate Interface (Cedar Candidate Object)

Cedar Candidates are generated during the analysis step and are representing possible results of a Field.

5.4.1.1 List of Methods and Properties

Attractor

Returns attractor of Candidate by zero-based index

AttractorCount

Returns number of attractors for Candidate

CopyToField

Transforms Candidate to Field of Workdoc

FilterID

Returns FilterID value specified by AddCandidate method of Field

FormatConfidence

Sets / returns confidence of string match algorithm performed by format search engine created Candidate

Height

Returns height of Candidate in pixel

KeepSpaces

Sets / returns if text should keep spaces between Words

Left

Returns left border of Candidate in pixel

Line

Returns text of a single line by zero-based index

LineCaption

Caption for each line in case of multi-line Candidates specified by zero-based index

LineCount

Sets / returns number of lines of Candidate

LineWordCount

Returns number of words of the specified line

LineWordID

Returns Word ID within a multi-line Field by zero-based LineIndex

LineWorktext

Returns Worktext object of single line within multi-line Field by zero-based index

PageNr

Returns DocPage number of Candidate location

RemoveAttractor

Removes attractor specified by AttractorIndex [0 - AttractorCount-1]

Text

Returns entire text of Candidate

Top

Returns top coordinate of Candidate in pixel

Weight

Sets / returns evaluation result

Width

Returns width of Candidate in pixel

WordCount

Returns number of Words of Candidate

WordID

Returns Word ID of specified zero-based index within first line

Worktext

Returns Worktext object of first line

The following property and method are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- AddAttractor

5.4.2 Methods and Properties of SCBCdrCandidate Interface

Attractor

```
Attractor (index As Long) As ISCBCdrAttractor (read only)
```

Returns the attractor of the Candidate by a zero-based index.

index

Specifies the index into the attractor array, must be between 0 and AttractorCount - 1.

AttractorCount

```
AttractorCount As Long (read only)
```

Returns the number of attractors for this Candidate.

CopyToField

```
CopyToField (pField As ISCBCdrField)
```

This method should be used to make a Candidate the Field's result. This will copy all required properties from the Candidate to the Field result.

pField

Reference to the Field containing the Candidate. States which field should get the values from the Candidate.

FilterID

```
FilterID As Long (read only)
```

This is the FilterID value as it was specified by the AddCandidate method of the Field.

FormatConfidence

```
FormatConfidence As Double (read/write)
```

Sets / returns the confidence of the string match algorithm performed by the format search engine that has created the Candidate.

Height

```
Height As Long (read only)
```

Returns the height of the Candidate in pixel.

KeepSpaces

```
KeepSpaces As Boolean (read/write)
```

This property specifies if the text created from several Words should keep the spaces between these Words or not. If a Candidate contains only a single Word created from a Worktext, the property has no effect.

Left

```
Left As Long (read only)
```

Returns the left border of the Candidate in pixel.

Line

```
Line (index As Long) As String (read only)
```

Returns the text of a single line. A Candidate can consist of one or more lines. Typically multi-line fields will be used for address search and other methods will extract an item with more than one line. This property allows you to read single lines of the Field.

index

Index of the Line, must be from 0 to LineCount-1.

LineCaption

```
LineCaption (index As Long) As String (read/write)
```

If a Candidate has more than one line, it is possible to assign a caption to each line to provide information about the content of the line.

index

Index of the line, must be from 0 to LineCount – 1

LineCount

```
LineCount As Long (read/write)
```

Returns the number of lines of the Candidate or can be used to set the number of lines of a Field.

LineWordCount

```
LineWordCount (index As Long) As Long (read only)
```

Returns number of words of the specified line.

index

Index of the line.

LineWordID

```
LineWordID (LineIndex As Long, WordIndex As Long) As Long (read only)
```

Returns the Word ID of the specified Line and Word index. The Word ID is the index of a Word within the Workdocs's Word array.

LineIndex

Index of the Line, must be from 0 to LineCount-1.

WordIndex

Index of the Word within the Line.

LineWorktext

```
LineWorktext (index As Long) As ISCBCroWorktext (read/write)
```

Returns the Worktext object of the single line specified by the zero-based index within a multi-line Field

index

Zero-based index of single line

PageNr

```
PageNr As Long (read only)
```

Returns the DocPage number where the Candidate is located.

RemoveAttractor

```
RemoveAttractor (AttractorIndex As Long)
```

Removes the attractor specified by index.

AttractorIndex

Index of attractor to be removed, valid range from 0 to AttractorCount-1.

Text

```
Text As String (read only)
```

Returns the text of the Candidate. In case of a multi-line Candidate only the text of the first line will be returned.

Top

```
Top As Long (read only)
```

Returns the top border of the Candidate in pixel.

Weight

```
Weight As Double (read/write)
```

Sets / returns the result of the evaluation which is between 0 and 1. This property can also be used to implement a script-based evaluation. Changing the weight of a Candidate during the PostEvaluate event will influence the extraction of the Field. To make a Candidate become the result of the Field, just set the weight to 1 and set the weight of all other Candidates to 0.

Width

```
Width As Long (read only)
```

Returns the width of the Candidate in pixel.

WordCount

```
WordCount As Long (read only)
```

Returns the Word count of the Candidate.

WordID

```
WordID (index As Long) As Long (read only)
```

Returns the Word ID of the specified Word index within the first line. The Word ID is the index of a Word within the Workdoc's Word array.

index

Zero-based index of the Word within the line.

Worktext

```
Worktext As ISCBCroWorktext (read only)
```

Returns the Worktext object of the first line.

5.5 SCBCdrTable

5.5.1 SCBCdrTable Interface (Cedar Table Object)

The Cedar Table object represents a logical Table in a Document which is assigned to a Cedar Field of a Workdoc.

5.5.1.1 Type Definitions

CDRTableHighlightMode

Enumeration containing the highlighting mode of a Table.

CDRTableHighlightAllCells	Highlight all cells of Table
CDRTableHighlightAllColumns	Highlight all columns of Table
CDRTableHighlightAllColumnsAdvanced	Advanced highlighting mode for both mapped and unmapped columns
CDRTableHighlightAllRows	Highlight all rows of Table
CDRTableHighlightCell	Highlight particular cell (as set by HighlightColumnIndex and HighlightRowIndex)
CDRTableHighlightColumn	Highlight column (as set by HighlightColumnIndex)
CDRTableHighlightNothing	Highlight nothing
CDRTableHighlightRow	Highlight row (as set by HighlightRowIndex)
CDRTableHighlightTable	Highlight whole Table

CDRLocation

Enumeration containing the location of a row, column or a cell in a Table.

CDRLocationBottom	Bottom corner coordinate
CDRLocationLeft	Left corner coordinate
CDRLocationRight	Right corner coordinate
CDRLocationTop	Top corner coordinate

5.5.1.2 List of Methods and Properties

AddColumn

Adds new column to Table

AddRow

Adds new row to Table

AddUMColumn

Adds new unmapped column to Table

AppendRows

Appends new rows

CellColor

Sets / returns color of Table cell

CellLocation

Sets / returns location of Table cell

CellText

Sets / returns text of Table cell

CellValid

Sets / returns validity flag of Table cell

CellValidationErrorDescription

Sets / returns ErrorDescription for cell validation

CellVisible

Sets / returns Visible flag of Table cell

CellWorktext

Sets / returns Worktext Object of cell

CellWorktextChanged

Sets / returns flag if cell Worktext has changed

Clear

Clears content of Table

ClearColumn

Clears content of existing column

ClearRow

Clears content of existing row

ClearUMColumn

Clears unmapped column

ColumnColor

Sets / returns color of column

ColumnCount
Returns number of columns

ColumnExportEnable
Sets / returns ExportEnable flag of column

ColumnIndex
Returns column index by name

ColumnLabelLocation
Sets / returns location of column label

ColumnLabelText
Sets / returns column label

ColumnLocation
Sets / returns location of column

ColumnMapped
Sets / returns flag if column has been mapped

ColumnName
Returns name of column

ColumnValid
Sets / returns validity flag for column

ColumnVisible
Sets / returns visible flag of column

DeleteColumn
Deletes column specified by name or by zero-based index

DeleteRow
Deletes row specified by zero-based index

DeleteUMColumn
Deletes unmapped column by zero-based index

FieldName
Sets / returns name of CdrField to which CdrTable object belongs to

FillColumn
Fills column with Words of in pixel specified area

FooterLocation

Sets / returns location of Table footer

FooterPageNr

Sets / returns DocPage number of Table footer

FooterText

Sets / returns text of Table footer

HeaderLocation

Sets / returns location of Table header

HeaderPageNr

Sets / returns DocPage number of Table header

HeaderText

Sets / returns text of Table header

HighlightColumnIndex

Sets / returns index of column to be highlighted

HighlightMode

Sets / returns HighlightMode of Table

HighlightRowIndex

Sets / returns index of row to be highlighted

HighlightUMColumnIndex

Sets / returns zero-based index of unmapped column to be highlighted

InsertColumn

Inserts new column after by ColumnIndex (zero-based) specified column

InsertRow

Inserts new row after specified RowIndex (zero-based)

InsertUMColumn

Inserts new unmapped column

LabellinePageNr

Sets / returns DocPage number of label line

LocationExplicit

Sets / returns LocationExplicit flag

MapColumn

Maps unmapped column

MergeRows

Merges two rows specified by zero-based indices

RemoveAllColumns
Removes all mapped table columns

RemoveAllRows
Removes all table rows

RemoveAllUMColumns
Removes all unmapped table columns

RowColor
Sets / returns color of row

RowCount
Returns number of rows

RowLocation
Sets / returns location of row by zero-based RowIndex

RowNumber
Sets / returns actual row number

RowPageNr
Sets / returns DocPage number of row

RowValid
Sets / returns validity flag of row by zero-based RowIndex

RowValidationErrorDescription
Sets / returns ErrorDescription for row validation by zero-based RowIndex

Significance
Sets / returns significance for corresponding evaluation property of Table

SwapColumns
Swaps two columns specified by zero-based indices

TableColor
Sets / returns color of Table

TableFirstPage
Sets / returns DocPage number of begin of Table

TableLastPage
Sets / returns DocPage number of end of Table

TableLocation

Sets / returns location of Table

TableValid

Sets / returns validity flag of Table

TableValidationErrorDescription

Sets / returns ErrorDescription for Table validation

Tag

Sets / returns tag

TotalSignificance

Sets / returns total significance of Table

UMCellColor

Sets / returns color of unmapped Table cell

UMCellLocation

Sets / returns location of unmapped Table cell by zero-based indices

UMCellText

Sets / returns text of unmapped Table cell by zero-based indices

UMCellVisible

Sets / returns Visible flag of unmapped Table cell by zero-based indices

UMCellWorktext

Sets / returns Worktext object of unmapped cell by zero-based indices

UMColumnColor

Sets / returns color of unmapped column

UMColumnCount

Returns number of unmapped columns

UMColumnLabelLocation

Sets / returns location of unmapped column label by zero-based UMColumnIndex

UMColumnLabelText

Sets / returns text of label of unmapped column by zero-based UMColumnIndex

UMColumnLocation

Sets / returns location of unmapped column by zero-based UMColumnIndex

UMColumnVisible

Sets / returns Visible flag of unmapped column by zero-based UMColumnIndex

UnMapColumn

Unmaps column

WeightingFactor

Sets / returns Weighting Factor for corresponding evaluation property

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- RowVisible
- TableVisible

5.5.2 Methods and Poperties of SCBCdrTable Interface

AddColumn

```
AddColumn (ColumnName As String) As Long
```

Adds a new column to a Table. Returns the index of the new column (zero-based).

ColumnName

[in] Name of column

AddRow

```
AddRow () As Long
```

Adds a new row to a Table. Returns the index of the new row (zero-based).

AddUMColumn

```
AddUMColumn (pUMColumnIndex As Long)
```

Adds a new unmapped column to a Table. Returns the index of the new unmapped column.

pUMColumnIndex

The method returns the zero-based index of the new column to this parameter.

AppendRows

```
AppendRows (top As Long, height As Long, PageNumber As Long)
```

Appends new rows over the specified range within the document.

top

Top of region used for creation or new rows

height

Height of region used for creation or new rows

PageNumber

DocpPage number of region

CellColor

```
CellColor (IsValid As Boolean) As OLE_COLOR (read/write)
```

Sets / returns the color of the Table cell.

IsValid

Flag indicating if color refers to valid or invalid Table cells

CellLocation

```
CellLocation (Column As Variant, RowIndex As Long, Location As CDRLocation)
         As Long (read/write)
```

Sets / returns the location of the Table cell.

Column

Zero-based index or name of column

RowIndex

Zero-based index of row

Location

Location parameter

CellText

```
CellText (Column As Variant, RowIndex As Long) As String (read/write)
```

Sets / returns the text of the Table cell

Column

Zero-based index or name of column

RowIndex

Zero-based index of row

CellValid

```
CellValid (Column As Variant, RowIndex As Long) As Boolean (read/write)
```

Sets / returns the validity flag of the Table cell. If the flag is set to false the in-/valid state of the table field will not be changed automatically. The flag can be checked, but the user must set the valid property separately.

Column

Zero-based index of name of column

RowIndex

Zero-based index of row

CellValidationErrorDescription

```
CellValidationErrorDescription (Column As Variant, RowIndex As Long)
         As String (read/write)
```

Sets / returns the ErrorDescription for the cell validation.

Column

Zero-based index or name of column

RowIndex

Zero-based index of row

CellVisible

```
CellVisible (Column As Variant, RowIndex As Long) As Boolean (read/write)
```

Sets / returns Visible flag of the Table cell (currently not used).

Column

Zero-based index of name of column

RowIndex

Zero-based index of row

CellWorktext

```
CellWorktext (Column As Variant, RowIndex As Long) As ISCBCroWorktext (read/write)
```

Sets / returns the Worktext object of the cell.

Column

Zero-based index or name of column

RowIndex

Zero-based index of row

CellWorktextChanged

```
CellWorktextChanged (Column As Variant, RowIndex As Long) As Boolean (read/write)
```

Sets / returns a flag indicating whether the cell Worktext has changed.

Column

Zero-based index or name of column

RowIndex

Zero-based index of row

Clear

```
Clear ()
```

Clears the content of the Table (i.e. removes all columns and all rows and resets all Table attributes).

ClearColumn

```
ClearColumn (Column As Variant)
```

Clears the content of an existing column.

Column

Zero-based index or name of column.

ClearRow

```
ClearRow (RowIndex As Long)
```

Clears the content of an existing row.

RowIndex

Zero-based index of row.

ClearUMColumn

```
ClearUMColumn (UMColumnIndex As Long)
```

Clears the content of an unmapped column.

UMColumnIndex

Zero-based index of unmapped column to be cleared

ColumnColor

```
ColumnColor (IsValid As Boolean) As OLE_COLOR (read/write)
```

Sets / returns the color of a column.

IsValid

Flag indicating if color refers to valid or invalid columns

ColumnCount

```
ColumnCount As Long (read only)
```

Returns the number of the columns.

ColumnExportEnable

```
ColumnExportEnable (Column As Variant) As Boolean (read/write)
```

Sets / returns the ExportEnable flag of a column.

Column

Zero-based index or name of column

ColumnIndex

```
ColumnIndex (ColumnName As String) As Long (read only)
```

This property returns the column index for the name of a column

ColumnName

Name of the column.

ColumnLabelLocation

```
ColumnLabelLocation (Column As Variant, Location As CDRLocation)
         As Long (read/write)
```

Sets / returns the location of a column label (referring to first label line in case of multi-page Tables).

Column

Zero-based index or name of column

Location

Location parameter

ColumnLabelText

```
ColumnLabelText (Column As Variant) As String (read/write)
```

Sets / returns the column label.

Column

Zero-based index or name of column

ColumnLocation

```
ColumnLocation (Column As Variant, PageNr As Long, Location As CDRLocation)
         As Long (read/write)
```

Sets / returns the location of the column.

Column

Zero-based index or name of column

PageNr

DocPage number

Location

Location parameter

ColumnMapped

```
ColumnMapped (Column As Variant) As Boolean (read/write)
```

Sets / returns a flag indicating whether a column has been mapped.

Column

Zero-based index or name of column

ColumnName

```
ColumnName (ColumnIndex As Long) As String (read only)
```

Returns the name of a column.

ColumnIndex

Zero-based Index of column

ColumnValid

```
ColumnValid (Column As Variant) As Boolean (read/write)
```

Sets / returns a validity flag for a column. If the flag is set to false the in-/valid state of the table field will not be changed automatically. The flag can be checked, but the user must set the valid property separately.

Column

Zero-based index or name of column

ColumnVisible

```
ColumnVisible (Column As Variant) As Boolean (read/write)
```

Sets / returns the visible flag of a column. (affects visibility of column in *Verifier*).

Column

Zero-based index or name of column

DeleteColumn

```
DeleteColumn (Column As Variant)
```

Deletes a column specified by its name or by index.

Column

Zero-based index or name of column

DeleteRow

```
DeleteRow (RowIndex As Long)
```

Deletes a row specified by index.

RowIndex

Zero-based index of row.

DeleteUMColumn

```
DeleteUMColumn (UMColumnIndex As Long)
```

Deletes an unmapped column specified by index.

UMColumnIndex

Zero-based index of unmapped column to be deleted

FieldName

```
FieldName As String (read/write)
```

Sets / returns the name of the CdrField to which the CdrTable object belongs to.

FillColumn

```
FillColumn (left As Long, top As Long, width As Long, height As Long,
          PageNumber As Long, Column As Variant)
```

Fills the column with Words of specified area. If the Table is empty, each text line will be assigned to a Table row. Otherwise the existing row segmentation will be used.

left

Left position of area in pixel

top

Top of area in pixel

width

Width of area in pixel

height

Height of area in pixel

PageNumber

DocPage number of area

Column

Zero-based index or name of destination column

FooterLocation

```
FooterLocation (Location As CDRLocation) As Long (read/write)
```

Sets / returns the location of the Table footer.

Location

Location parameter

FooterPageNr

```
FooterPageNr As Long (read/write)
```

Sets / returns the DocPage number of the Table footer.

FooterText

```
FooterText As String (read/write)
```

Sets / returns the text of the Table footer.

HeaderLocation

```
HeaderLocation (Location As CDRLocation) As Long (read/write)
```

Sets / returns the location of the Table header.

Location

Location parameter

HeaderPageNr

```
HeaderPageNr As Long (read/write)
```

Sets / returns the DocPage number of the Table header.

HeaderText

```
HeaderText As String (read/write)
```

Sets / returns the text of the Table header.

HighlightColumnIndex

```
HighlightColumnIndex As Long (read/write)
```

Sets / returns the index of the column to be highlighted.

HighlightMode

```
HighlightMode As CDRTableHighlightMode (read/write)
```

Sets / returns HighlightMode of Table.

CDRTableHighlightTable:	**Highlights whole Table**
CDRTableHighlightAllColumns:	Highlights all columns
CDRTableHighlightAllRows:	Highlights all rows
CDRTableHighlightAllCells:	Highlights all cells
CDRTableHighlightColumn:	Highlights single column (as set by HighlightColumnIndex)
CDRTableHighlightRow:	Highlights single row (as set by HighlightRowIndex)
CDRTableHighlightCell:	Highlights single cell (as set by HighlightColumnIndex and HighlightRowIndex)

HighlightRowIndex

```
HighlightRowIndex As Long (read/write)
```

Sets / returns the index of the row to be highlighted.

HighlightUMColumnIndex

```
HighlightUMColumnIndex As Long (read/write)
```

Sets / returns the zero-based index of an unmapped column to be highlighted.

InsertColumn

```
InsertColumn (ColumnIndex As Long, ColumnName As String)
```

Inserts a new column after by ColumnIndex specified column.

ColumnIndex

Zero-based index of existing column, behind which new column is to be inserted.

ColumnName

Name of new column

InsertRow

```
InsertRow (RowIndex As Long)
```

Inserts a new row after specified RowIndex.

RowIndex

Zero-based index of existing row, below which new row is to be inserted.

InsertUMColumn

```
InsertUMColumn (UMColumnIndex As Long)
```

Inserts new unmapped column.

UMColumnIndex

Zero-based index of new column.

LabellinePageNr

```
LabellinePageNr As Long (read/write)
```

Sets / returns the DocPage number of the label line (first occurrence in case of multi-page Tables).

LocationExplicit

```
LocationExplicit As Boolean (read/write)
```

Sets / returns LocationExplicit flag. If set to VARIANT_TRUE, use location (Table, columns, rows) as set explicitly (column, row, Table) as set explicitly by corresponding properties. If set to VARIANT_FALSE calculate positions of Table, columns and rows from locations of cells.

MapColumn

```
MapColumn (UMColumnIndex As Long, Column As Variant)
```

Maps an unmapped column, i.e. transfers content of unmapped source column to specified target column. If the latter is non-empty, a new unmapped column will be created in order to store the old content of the target column.

UMColumnIndex

Zero-based index of unmapped source column

Column

Zero-based index or name of destination column

MergeRows

```
MergeRows (RowIndex1 As Long, RowIndex2 As Long)
```

Merges two rows specified by two indices.

RowIndex1

Zero-based index of row 1

RowIndex2

Zero-based index of row 2

RemoveAllColumns

```
RemoveAllColumns ()
```

This method removes all mapped table columns.

RemoveAllRows

```
RemoveAllRows ()
```

This method removes all table rows.

RemoveAllUMColumns

```
RemoveAllUMColumns ()
```

This method removes all unmapped table columns.

RowColor

```
RowColor (IsValid As Boolean) As OLE_COLOR (read/write)
```

Sets / returns the color of the row.

IsValid

Flag indicating if color refers to valid or invalid rows

RowCount

```
RowCount As Long (read only)
```

Returns the number of the rows.

RowLocation

```
RowLocation (RowIndex As Long, Location As CDRLocation) As Long (read/write)
```

Sets / returns the location of the row.

RowIndex

Zero-based index of row

Location

Location parameter

RowNumber

```
RowNumber (RowIndex As Long) As Long (read/write)
```

This property sets or returns the actual number of row.

RowIndex

Zero-based index of row.

Example:

```
Private Sub Tabelle_ValidateCell(pTable As SCBCdrPROJLib.SCBCdrTable, pWorkdoc As_
SCBCdrPROJLib.SCBCdrWorkdoc, ByVal Row As Long, ByVal Column As Long, pValid As Boolean)
' Script sample that considers for extraction and validation that the  Table Extraction can
' extract table rows for which a table cell has multiple extracted lines
   Dim nCurrentRow, nRow, nLine As Integer
   Dim strCellText As String

   nCurrentRow = pTable.RowNumber(Row)
   strCellText = pTable.CellText(Column, Row)

   nLine = Row - 1
   nRow = nCurrentRow
   While (nLine >= 0) And (nRow = nCurrentRow)
      nRow = pTable.RowNumber(nLine)
      If (nRow = nCurrentRow) And (pTable.CellText(Column, nLine) <> "") Then
         strCellText = pTable.CellText(Column, nLine) + " " + strCellText
      End If
      nLine = nLine - 1
   Wend

   nLine = Row + 1
   nRow = nCurrentRow
   While (nLine < pTable.RowCount) And (nRow = nCurrentRow)
      nRow = pTable.RowNumber(nLine)
      If (nRow = nCurrentRow) And (pTable.CellText(Column, nLine) <> "") Then
         strCellText = strCellText + " " + pTable.CellText(Column, nLine)
      End If
      nLine = nLine + 1
   Wend

   If strCellText = "" Then
      pValid = False
   Else
      pValid = True
   End If
End Sub
```

RowPageNr

```
RowPageNr (RowIndex As Long) As Long (read/write)
```

Sets / returns the DocPage number of a row.

RowIndex

Zero-based index of row.

RowValid

```
RowValid (RowIndex As Long) As Boolean (read/write)
```

Sets / returns a validity flag of a row. If the flag is set to false the in-/valid state of the table will not be changed automatically. The flag can be checked, but the user must set the valid property separately.

RowIndex

Zero-based index of row

RowValidationErrorDescription

```
RowValidationErrorDescription (RowIndex As Long) As String (read/write)
```

Sets / returns an ErrorDescription for a row validation.

RowIndex

Zero-based index of row

Significance

```
Significance (EvalPropIndex As Long) As Double (read/write)
```

Sets / returns the significance for corresponding evaluation property of the Table.

EvalPropIndex

Index of evaluation property:

1:	**percentage of required columns identified**
2:	percentage of table columns mapped
3:	average percentage of elements found in cell, for which element is required
4:	Average no-overlap to neighboring cells (column view)
5:	Average no-overlap to neighboring cells (row view)

SwapColumns

```
SwapColumns (ColumnIndex1 As Long, ColumnIndex2 As Long)
```

Swaps the two specified columns, i.e. exchanges their whole content (the column names, however, will not be exchanged).

ColumnIndex1

Zero-based index of column 1

ColumnIndex2

Zero-based index of column 2

TableColor

```
TableColor (IsValid As Boolean) As OLE_COLOR (read/write)
```

Sets / returns the color of the Table.

IsValid

Flag indicating if color refers to a valid or an invalid Table.

TableFirstPage

```
TableFirstPage As Long (read/write)
```

Sets / returns the DocPage number of the begin of a Table (must be set after creation of a Table, but must not changed afterwards).

TableLastPage

```
TableLastPage As Long (read/write)
```

Sets / returns the DocPage number of the end of a Table (must be set after creation of a Table, and after assigning the first DocPage, but must not changed afterwards).

TableLocation

```
TableLocation (PageNr As Long, Location As CDRLocation) As Long (read/write)
```

Sets / returns the location of a Table.

PageNr

DocPage number

Location

Location parameter

TableValid

```
TableValid As Boolean (read/write)
```

Sets / returns a validity flag of the Table. If the flag is set to false the in-/valid state of the table field will not be changed automatically. The flag can be checked, but the user must set the valid property separately.

TableValidationErrorDescription

```
TableValidationErrorDescription As String (read/write)
```

Sets / returns an ErrorDescription for the Table validation.

Tag

```
Tag As String (read/write)
```

Sets / returns a tag associated with the Table.

TotalSignificance

```
TotalSignificance As Double (read/write)
```

Sets / returns the total significance of the Table.

UMCellColor

```
UMCellColor As OLE_COLOR (read/write)
```

Sets / returns the color of an unmapped Table cell.

UMCellLocation

```
UMCellLocation (UMColumnIndex As Long, RowIndex As Long, Location As CDRLocation)
          As Long (read/write)
```

Sets / returns the location of an unmapped Table cell.

UMColumnIndex

Zero-based index of unmapped column

RowIndex

Zero-based index of unmapped row

Location

Location parameter

UMCellText

```
UMCellText (UMColumnIndex As Long, RowIndex As Long) As String (read/write)
```

Sets / returns the text of an unmapped Table cell.

UMColumnIndex

Zero-based index of unmapped column

RowIndex

Zero-based index of row

UMCellVisible

```
UMCellVisible (UMColumnIndex As Long, RowIndex As Long) As Boolean (read/write)
```

Sets / returns a Visible flag of an unmapped Table cell.

UMColumnIndex

Zero-based index of unmapped column

RowIndex

Zero-based index of row

UMCellWorktext

```
UMCellWorktext (UMColumnIndex As Long, RowIndex As Long) As ISCBCroWorktext
          (read/write)
```

Sets / returns the Worktext Object of an unmapped cell.

UMColumnIndex

Zero-based index of unmapped column

RowIndex

Zero-based index of row

UMColumnColor

```
UMColumnColor As OLE_COLOR (read/write)
```

Sets / returns the color of an unmapped column.

UMColumnCount

```
UMColumnCount As Long (read only)
```

Returns the number of unmapped columns.

UMColumnLabelLocation

```
UMColumnLabelLocation (UMColumnIndex As Long, Location As CDRLocation)
         As Long (read/write)
```

Sets / returns the location of an unmapped column label.

UMColumnIndex

Zero-based index of unmapped column

Location

Location parameter

UMColumnLabelText

```
UMColumnLabelText (UMColumnIndex As Long) As String (read/write)
```

Sets / returns the text of label of an unmapped column.

UMColumnIndex

Zero-based index of unmapped column

UMColumnLocation

```
UMColumnLocation (UMColumnIndex As Long, PageNr As Long, Location As CDRLocation)
         As Long (read/write)
```

Sets / returns the location of an unmapped column.

UMColumnIndex

Zero-based index of unmapped column

PageNr

DocPage number

Location

Location parameter

UMColumnVisible

```
UMColumnVisible (UMColumnIndex As Long) As Boolean (read/write)
```

Sets / returns a Visible flag of an unmapped column (currently not used).

UMColumnIndex

Zero-based index of unmapped column

UnMapColumn

```
UnMapColumn (Column As Variant) As Long
```

Unmaps column, i.e. transfers content of specified source column to new unmapped column. The specified column remains in the Table, however its content is cleared.

Column

Zero-based index or name of source column

WeightingFactor

```
WeightingFactor (EvalPropIndex As Long) As Double (read/write)
```

Sets / returns a Weighting Factor for a corresponding evaluation property.

EvalPropIndex

Index of evaluation property:

1:	**percentage of required columns identified**
2:	percentage of table columns mapped
3:	average percentage of elements found in cell, for which element is required
4:	Average no-overlap to neighboring cells (column view)
5:	Average no-overlap to neighboring cells (row view)

5.5.3 Accessing Fields without Detailed Information

The easiest way to access the position and content of header and table fields is to use the functions of the SCBCdrField (for header fields) and SCBCdrTable objects (for table fields.)

The following script sample is useful for accessing header fields:

```
strFieldText = pField.Text
lFieldPositionLeft = pField.Left
lFieldPositionTop = pField.Top
lFieldPositionWidth = pField.Width
lFieldPositionHeight = pField.Height
```

The following script sample is useful for accessing table fields:

```
strCellText = pTable.CellText(Column, Row)
lCellPositionLeft = pTable.CellLocation(Column, Row, CDRLocationLeft)
lCellPositionTop = pTable.CellLocation(Column, Row, CDRLocationTop)
lCellPositionRight = pTable.CellLocation(Column, Row, CDRLocationRight)
lCellPositionBottom = pTable.CellLocation(Column, Row, CDRLocationBottom)
```

5.5.4 Accessing Fields with Detailed Information

The functions “pField.Text” and “pTable.CellText” retrieve text from all lines in the Worktext of a header field or a table cell. However, detailed information can be accessed only via the SCBCroWorktext object of the header field or table cell.

The following script sample is useful for retrieving a Worktext object for the header field:

```
Dim theFieldWorktext As SCBCroWorktext
Set theFieldWorktext = pField.Worktext
```

The following script sample is useful for retrieving a Worktext object for the table cell:

```
Dim theCellWorktext As SCBCroWorktext
Set theCellWorktext = pTable.CellWorktext(Column, Row)
```

The following extra information can be obtained via the Worktext object:

- The position and confidence of every particular character in a header field or table cell.
- The position and content of every particular line in a field or table cell if they contain multiple lines.

For example, the following script code can be used to explore cell lines on a line-by-line basis:

```
Dim i As Long
Dim theCellWorktext As SCBCroWorktext
Dim strNextLineText As String
Dim lNextLinePositionLeft, lNextLinePositionTop As Long
Dim lNextLinePositionWidth, lNextLinePositionHeight As Long

Set theCellWorktext = pTable.CellWorktext(Column, Row)

For i = 0 To theCellWorktext.LineCount - 1 Step 1
   strNextLineText = theCellWorktext.LineText(i)
   lNextLinePositionLeft = theCellWorktext.LineDim(PX_Left, i)
   lNextLinePositionTop = theCellWorktext.LineDim(PX_Top, i)
   lNextLinePositionWidth = theCellWorktext.LineDim(PX_Width, i)
   lNextLinePositionHeight = theCellWorktext.LineDim(PX_Height, i)
Next i
```

5.5.5 Modifying Content and Position – Single-Line Fields

Sections ***5.5.5*** and ***5.5.6*** describe how to correctly modify table cell content and position. The same techniques can be applied to header fields.

To support interactive learning in Advanced Verifier, the script should correctly modify not only the content but also position of the considered field or table cell. Moreover, positional information is usually even more important for learning, because it gives more key words and search words to the engines, rather than format and text content of the field itself.

5.5.6 Modifying Content – Multi-Line Fields

You can use the “pTable.CellText” property to set the required text to a cell, but doing so will overwrite all existing Worktext lines with one line and their common location. This may negatively affect correctness of the learning information when learning BTE in interactive mode. Also, script code such as pTable.CellWorktext("Unit Price Column", lMyRowIndex) = "New Text" (often used by PS) completely overwrites entire complex Worktext information, including cell lines' text, confidences of every character, and even positional information for every Worktext line.

If your project must fully support multi-line cells, then cell text information must be modified on a line-by-line basis.

For example, overwriting text for a specified cell's Worktext line without modifying its location would require script code similar to the sample shown below. The example modifies the third line of the cell in the 18th row, “Item Description” table column. It sets its text to “Intel S845WD1-E, Entry Workstation Board” without modifying the cell line’s location.

```
Dim strNewLineText As String
Dim theCellWorktext As SCBCroWorktext
Dim lFirstCharInLine, lOldLineLength, lLineIndex As Long

lLineIndex = 2
strNewLineText = "Intel S845WD1-E, Entry Workstation Board"

Set theCellWorktext = pTable.CellWorktext(“Item Description”, 17)

lFirstCharInLine = theCellWorktext.GetCharIndex(lLineIndex, 0)
lOldLineLength = theCellWorktext.LineLength(lLineIndex)
theCellWorktext.InsertBefore(lFirstCharInLine, strNewLineText, 100, 0, 0, 0, 0)
theCellWorktext.Remove(lFirstCharInLine + Len(strNewLineText), lOldLineLength)
```

If it is necessary to extend the cell lines information with all details, it might be important to use “InsertBefore” and “InsertAfter” functions for every new character to specify required position and confidence.

5.5.7 Modifying Position – Multi-Line Fields

The Worktext object stores information about the location of every line separately from the location of the characters that the line contains. In other words, line location is not determined automatically when adding new characters to the line and can be set only at the point when the line is being created. This means that there is no way to dynamically modify line location and that, to achieve this, the script has to create a new instance of the Worktext and then create all the lines from scratch using “AppendLine” method of the Worktext object.

5.5.8 Removal of Lines in Worktext

If a Worktext line is supposed to be removed from a header or table field, this can be easily accomplished via the “RemoveLine” method of Worktext object:

```
theWorktext.RemoveLine(LineIndex)
```

Where “LineIndex” is the index of line in Worktext that has to be removed.

5.5.9 Script Sample: Exchanging Cell Lines

Below is a script sample that demonstrates how to exchange cell lines between two columns.

The sample correctly exchanges the first cell line from the “Description” column and the first line from the “Total Price” column. If the document will be learned (in Verifier or Designer) right after the function below is executed, the applied change will be appropriately inherited in the Table Extraction Learn Set:

```
Private Sub Document_OnAction(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, ByVal ActionName As String)

   Dim i As Long
   Dim theCellWorktext As SCBCroWorktext
   Dim theTable As SCBCdrTable
   Dim lColumnOnePos, lColumnTwoPos As Long
   Dim strColumnOneText, strColumnTwoText As String

   Set theTable = pWorkdoc.Fields("Table").Table(pWorkdoc.Fields("Table").ActiveTableIndex)

   For i = 0 To theTable.RowCount - 1 Step 1

      lColumnOnePos = theTable.CellWorktext("Description", i).LineDim(PX_Left, 0)
      lColumnTwoPos = theTable.CellWorktext("Total Price", i).LineDim(PX_Left, 0)
      theTable.CellWorktext("Total Price", i).ReplaceLineLocation(PX_Left, 0, lColumnOnePos)
      theTable.CellWorktext("Description", i).ReplaceLineLocation(PX_Left, 0, lColumnTwoPos)
      lColumnOnePos = theTable.CellWorktext("Description", i).LineDim(PX_Top, 0)
      lColumnTwoPos = theTable.CellWorktext("Total Price", i).LineDim(PX_Top, 0)
      theTable.CellWorktext("Total Price", i).ReplaceLineLocation(PX_Top, 0, lColumnOnePos)
      theTable.CellWorktext("Description", i).ReplaceLineLocation(PX_Top, 0, lColumnTwoPos)
      lColumnOnePos = theTable.CellWorktext("Description", i).LineDim(PX_Width, 0)
      lColumnTwoPos = theTable.CellWorktext("Total Price", i).LineDim(PX_Width, 0)
      theTable.CellWorktext("Total Price", i).ReplaceLineLocation(PX_Width, 0, lColumnOnePos)
      theTable.CellWorktext("Description", i).ReplaceLineLocation(PX_Width, 0, lColumnTwoPos)
      lColumnOnePos = theTable.CellWorktext("Description", i).LineDim(PX_Height, 0)
      lColumnTwoPos = theTable.CellWorktext("Total Price", i).LineDim(PX_Height, 0)
      theTable.CellWorktext("Total Price", i).ReplaceLineLocation(PX_Height, 0, lColumnOnePos)
      theTable.CellWorktext("Description", i).ReplaceLineLocation(PX_Height, 0, lColumnTwoPos)
      strColumnOneText = theTable.CellWorktext("Description", i).LineText(0)
      strColumnTwoText = theTable.CellWorktext("Total Price", i).LineText(0)
      theTable.CellWorktext("Total Price", i).ReplaceLineText(0, strColumnOneText)
      theTable.CellWorktext("Description", i).ReplaceLineText(0, strColumnTwoText)
   Next i

End Sub
```

5.6 SCBCdrTextblock

5.6.1 SCBCdrTextblock Interface (Cedar TextBlock Object)

This object represents a TextBlock on a Document. A TextBlock may contain one or more than one line.

5.6.1.1 List of Methods and Properties

Color
Sets / returns color used for TextBlock highlighting

Height
Returns height of TextBlock in pixel

Left
Returns left border of TextBlock in pixel

PageNr
Returns number of DocPage where TextBlock is located

Text
Returns text of TextBlock

Top
Returns top border of TextBlock in pixel

Visible
Sets / returns if highlighted rectangle of TextBlock is visible

Weight
Returns block weight

Width
Returns width of the TextBlock in pixel

WordCount
Returns number of Words belonging to TextBlock

WordID
Returns WordID of Word specified by zero-based index inside TextBlock

5.6.2 Methods and properties of SCBCdrTextblock Interface

Color

```
Color As OLE_COLOR (read/write)
```

Sets / returns the color that will be used for TextBlock highlighting.

Height

```
Height As Long (read only)
```

Returns the height of the TextBlock in pixel.

Left

```
Left As Long (read only)
```

Returns the left border of the TextBlock in pixel.

PageNr

```
PageNr As Long (read only)
```

Returns the number of the DocPage where the TextBlock is located.

Text

```
Text As String (read only)
```

The whole text of the TextBlock is returned.

Top

```
Top As Long (read only)
```

Returns the top border of the TextBlock in pixel.

Visible

```
Visible As Boolean (read/write)
```

This property controls if the highlighted rectangle of the TextBlock should be visible if the TextBlock highlighting is enabled.

Weight

```
Weight As Double (read only)
```

This property returns the block weight.

Width

```
Width As Long (read only)
```

The width of the TextBlock is returned in pixel.

WordCount

```
WordCount As Long (read only)
```

The number of Words belonging to the TextBlock is returned.

WordID

```
WordID (index As Long) As Long (read only)
```

This property contains the WordIDs of all Words belonging to the TextBlock. The WordID can be used as index for the Word array of the Workdoc. The sequence of the Words inside the TextBlock is sorted

by the geometrical position of the Words while the sequence of the Words inside the Workdoc is determined by the used OCR.

index

Index of Word inside the TextBlock. Must be between 0 and WordCount-1.

5.7 SCBCdrWord

5.7.1 SCBCdrWord Interface (Cedar Word object)

This object represents a textual Word of a Document.

5.7.1.1 List of Methods and properties

Color

Sets / returns color used for highlighting of checked Words

Height

Returns height of Word in pixel

Left

Returns left border of Word in pixel

PageNr

Returns number of DocPage where Word is located

StartPos

Returns index of first character of Word inside Worktext attached to Workdoc, valid range from 0 to WordCount - 1

Text

Returns text of Word

TextLen

Returns number of characters of Word

Tooltip

Sets / returns tool tip string displayed in checked Words highlight mode.

Top

Returns top border of Word in pixel

Visible

Sets / returns if highlighted rectangle of Word is visible

Width

Returns width of Word in pixel

Worktext

Returns Worktext object of Word

5.7.2 Methods and Properties of SCBCdrWord Interface

Color

```
Color As OLE_COLOR (read/write)
```

The color that will be used for highlighting checked Words is set / returned. This requires to set the visible property of the Word to TRUE and select the “Highlight Checked Words” of the view menu.

Height

```
Height As Long (read only)
```

Returns the height of the Word in pixel.

Left

```
Left As Long (read only)
```

Returns the left border of the Word in pixel.

PageNr

```
PageNr As Long (read only)
```

Returns the number of the DocPage where the Word is located.

StartPos

```
StartPos As Long (read only)
```

Returns the index of the first character of the Word inside the Worktext attached to the Workdoc. The valid range of the returned value is between 1 and WordCount - 1.

Text

```
Text As String (read only)
```

The text of the Word is returned.

TextLen

```
TextLen As Long (read only)
```

The number of characters of the Word is returned.

Tooltip

```
Tooltip As String (read/write)
```

This property sets / returns a tooltip string which will be displayed in the checked Words highlight mode. This requires to set the visible property of the Word to TRUE and select the “Highlight Checked Words” of the view menu. If the tool tip string is empty, the Word’s text will be used as tool tip text.

Top

```
Top As Long (read only)
```

Returns the top border of the Word in pixel.

Visible

```
Visible As Boolean (read/write)
```

This property sets / returns if the highlighted rectangle of the Word should be visible if the Word highlighting for checked Words is enabled. In case of normal Word highlighting all Words are highlighted regardless of the state of the visible property.

Width

```
Width As Long (read only)
```

Returns the width of the Word in pixel.

Worktext

```
Worktext As ISCBCroWorktext (read only)
```

This property returns the Worktext object of the Word.

5.8 SCBCdrDocPage

5.8.1 SCBCdrDocPage Interface (Cedar DocPage object)

An object representing a single DocPage within a Workdoc.

5.8.1.1 Type Definitions

CDRPageSource

Enumeration containing the Page source.

CDRPageSourceFrontPage	Front Page assigned to Workdoc
CDRPageSourceRearPage	Rear Page assigned to Workdoc
CDRPageSourceUnknown	Assigned Page to Workdoc is not known

CroLinesDir

Enumeration specifying the direction of a line.

CroLinesDir_Horizontal	Horizontal line
CroLinesDir_Vertical	Vertical line

CroLinesKooType

Further information about a line.

CroLinesKoorType_Angle	Angle of line
CroLinesKoorType_FirstPX	Starting abscissa of line
CroLinesKoorType_FirstPY	Starting ordinate of line
CroLinesKoorType_Length	Length of line
CroLinesKoorType_SecondPX	Ending abscissa of line
CroLinesKoorType_SecondPY	Ending ordinate of line
CroLinesKoorType_Thick	Thickness of line

5.8.1.2 List of Methods and Properties

DisplayImage

Sets / returns displayed DocPage in Viewer, valid range from 0 to ImageCount-1

DocIndex

Returns document index of DocPage within Workdoc, zero-based ImageIndex

DocPageIndex

Returns DocPage offset inside document, zero-based ImageIndex

GetResolution

Returns resolution of specified Image in pixel, zero-based ImageIndex

Height

Returns height of DocPage in millimeter

Image

Returns Image object by zero-based index of DocPage

ImageCount

Returns number of Images available for DocPage

Line

Returns some specific information regarding a Line

LinesCount

Returns number of straight lines detected in document

PageSource

Sets / returns source of DocPage

Rotate

Rotates underlying Images by specified angle

Rotation

Returns by Rotate applied rotation angle

Text

Returns text of DocPage if OCR was executed

Width

Returns width of DocPage in millimeter

5.8.2 Methods and Properties of SCBCdrDocPage Interface

DisplayImage

```
DisplayImage As Long (read/write)
```

A DocPage of a Workdoc can have up to two Images, e.g., a black/white image for OCR and a color image for manual correction or archiving. The DisplayImage property specifies the index of the Image, which should be displayed if the DocPage is visible inside a Viewer. The value of this property can take values of 0 to ImageCount-1.

DocIndex

```
DocIndex (ImageIndex As Long) As Long (read only)
```

This property specifies the index of the document inside the Workdoc where this DocPage belongs. See also the DocFileName and DocFileType property of the SCBCdrWorkdoc object.

ImageIndex

ImageIndex of the DocPage. Valid indices are 0 to ImageCount-1.

DocPageIndex

```
DocPageIndex (ImageIndex As Long) As Long (read only)
```

This property specifies the DocPage offset inside the document where this DocPage belongs.

ImageIndex

Index of the Image of the DocPage. Valid indexes are 0 to ImageCount-1.

GetResolution

```
GetResolution (ImageIndex As Long, pXRes As Long, pYRes As Long)
```

This method returns the resolution of the specified Image in pixel.

ImageIndex

[in] Index of the Image of the DocPage. Valid indices are 0 to ImageCount-1.

pXRes

[out] Will contain the x resolution after execution of the method.

pYRes

[out] Will contain the y resolution after execution of the method.

Height

```
Height As Double (read only)
```

This property returns the height of the DocPage in millimeter.

Image

```
Image (index As Long) As ISCBCroImage (read only)
```

Returns an Image object for the specified index of the DocPage. If the DocPage was based on a text based document, the text will be rendered to a new created SCBCroImage or a reference to Image will be provided.

index

Index of the Image of the DocPage. Valid indices are 0 to ImageCount-1.

ImageCount

```
ImageCount As Long (read only)
```

Returns the number of Images available for the DocPage. A DocPage of a Workdoc can have up to two Images, e.g. a black/white Image for OCR and a color image for manual correction or archiving. Typically there will be only one Image available.

Line

```
Line (LineIndex As Long, LineDir As CroLinesDir, KooType As CroLinesKooType)
         As Long (read only)
```

This property returns some specific property of line, viz. starting X ect., of some specific index and direction.

LineIndex

Zero-based index of the Line.

LineDir

Direction of Line (Horizontal or Vertical).

KooType

Information of a Line (starting X, starting Y, End X, End Y etc.)

LinesCount

```
LinesCount (LinesDir As CroLinesDir) As Long (read only)
```

This property returns the number of horizontal or vertical Lines present in a document.

LinesDir

Direction of Line (Horizontal or Vertical).

PageSource

```
PageSource As CDRPageSource (read/write)
```

Sets / returns a source of a DocPage. At the time of scanning, a DocPage can be directly assigned to Workdoc.

Rotate

```
Rotate (angle As Double)
```

This method rotates the underlying Images by the specified angle. Internally the documents (e.g., Tiff Files) will not be changed, but the DocPage stores the rotation angle and rotates the Image before displaying it inside a viewer.

angle

Specifies the rotation angle in a range of -180 to +180.

Rotation

```
Rotation As Double (read only)
```

This property returns the rotation angle as it was applied by Rotate method.

Text

```
Text As String (read only)
```

Returns the text of the DocPage if OCR was already executed.

Width

```
Width As Double (read only)
```

Returns the width of the DocPage in millimeter.

5.9 SCBCdrFolder

5.9.1 SCBCdrFolder Interface (Cedar Folder object)

A Folder may represent an array of Workdocs within a Batch. A Folder may contain one ore more Workdocs. During classification and extraction it is possible to access all Workdocs of the same Folder from script.

5.9.1.1 List of Methods and Properties

AddDocument
Adds Workdoc into Folder at last returned position

Clear
Frees all in between allocated memories by Folder

Document
Returns Workdoc from specified zero-based index of document array of Folder

DocumentCount
Returns number of Workdocs within Folder

FolderData
Stores / retrieves variable number of strings

InsertDocument
Inserts Workdoc into Folder at zero-based index

MoveDocument
Moves Workdoc from zero-based FromIndex in Folder to zero-based ToIndex position

RemoveDocument
Removes Workdoc from Folder specified by zero-based index

5.9.2 Methods and Properties of Interface SCBCdrFolder

AddDocument

```
AddDocument (pWorkdoc As ISCBCdrWorkdoc, pNewIndex As Long)
```

This method is used to add a Workdoc into a Folder at the last position and also returns the position where the Workdoc is appended.

pWorkdoc
[in] Added Workdoc Object

pNewIndex
[out] Index position in a Folder where Workdoc is inserted

Clear

```
Clear ()
```

Frees all the in between allocated memories by Folder.

Document

```
Document (Index As Long) As ISCBCdrWorkdoc (read only)
```

This property returns a Workdoc from the specified index of the document array of the Folder. A Workdoc can get its own index within the Workdoc from the SCBCdrWorkdoc.FolderIndex property.

Index

The index of the Workdoc within the Folder. Must be from 0 to DocumentCount-1.

DocumentCount

```
DocumentCount As Long (read only)
```

The number of Workdocs within the Folder is returned.

FolderData

```
FolderData (Index As String) As String (read/write)
```

The FolderData property provides the possibility to store and load a variable number of strings using any string as index key. The FolderData string list will not be saved to disk when the workdoc is saved, but you can use it during extraction to exchange information about the extraction states between different workdocs of a folder.

Reading the FolderData with an index where no data was saved before returns an empty string. This can be used to vote extraction results over the documents of a folder.

Index

Any non-empty string which is used as index key

Example:

```
' writing FolderData
pWorkdoc.Folder.FolderData("NumberFound") = "1"
pWorkdoc.Folder.FolderData("Number") = pWorkdoc.Field("Number")
…
' reading FolderData
if pWorkdoc.Folder.FolderData("NumberFound") = "1" then
 if len(pWorkdoc.Field("Number")) > 0 then
   ' takeover the result from the other workdoc
   pWorkdoc.Field("Number") = pWorkdoc.Folder.FolderData("Number")
 else
   ' compare results
   if pWorkdoc.Field("Number") = pWorkdoc.Folder.FolderData("Number") then
      ' found the same number again
   else
      ' found a different number on this document
   end if
 end if
end if
```

InsertDocument

```
InsertDocument (Index As Long, pWorkdoc As ISCBCdrWorkdoc)
```

This method is used to insert a Workdoc into a Folder at some given position.

Index

Index at which Workdoc is to be inserted, zero-based indexing

pWorkdoc

Workdoc object

MoveDocument

```
MoveDocument (FromIndex As Long, ToIndex As Long)
```

This method is used to move a Workdoc from one position to another position in a Folder.

FromIndex

Zero-based Index from where Workdoc is moved

ToIndex

Zero-based index where Workdoc is to be placed

RemoveDocument

```
RemoveDocument (index As Long)
```

This method is used to remove a Workdoc from a given index from a Folder.

index

Zero-based index in a Folder from where Workdoc is to be removed.

Chapter 6 Oracle Forms Recognition Project Object Reference (SCBCdrPROJLib)

6.1 SCBCdrProject

6.1.1 SCBCdrProject Interface (Cedar Project object)

The Cedar Project object represents a complete Project definition including all DocClasses, all FieldDefs, all used classification and extraction methods and all implemented script methods.

6.1.1.1 Type Definitions

CDRClassifyMode

This type defines the algorithms for how the results of several classification engines can be combined.

CDRClassifyAverage	Average will be computed
CDRClassifyMax	Maximum will be computed
CDRClassifyWeightedDistance	For each cell of classification matrix difference between maximum of column and classification weight is calculated

CdrSLWDifferentResultsAction

When the Template and Associative Search engines determine different results during classification, there are different options how the program should continue the processing.

CdrDoNothing	Let *Verifier* user decide to skip special processing altogether.
CdrDoSmartDecision	Make a smart decision, e.g. the machine makes the decision for the classification.
CdrUseDocumentClassName	Automatically assign current document class name to the supplier field content.
CdrUseSupplierField	Automatically assign supplier field content to the document class name.

CdrForceValidationMode

This table defines the options for Force Validation.

CdrForceValDefault	CdrForceValidationModeDefault: ForceValidationMode inherited
CdrForceValForbidden	CdrForceValidationModeForbidden: ForceValidation (3*return) not allowed
CdrForceValPermitted	CdrForceValidationModePermitted: ForceValidation (3*return) allowed

CdrType

This type defines the different message types.

CDRTypeInfo	An informational message.
CDRTypeWarning	A warning message.
CDRTypeError	An error message.

CdrSeverity

This type defines the different message severities.

CDRSeverityLogFileOnly	• Store the message to the log file only.
CDRSeveritySystemMonitoring	Store the message in the log file and forward it to the host instance's MMC console and to the System Monitoring

	service of the Runtime Server.
CDRSeverityEmailNotification	Store the message in the log file and forward it to the MMC console / System Monitoring view and send as an e-mail to the system administrators via System Monitoring service of Runtime Server.

6.1.1.2 List of Methods and Properties

AllClasses

Returns Collection of all defined DocClasses of current Project

BaseClasses

Returns Collection containing all defined BaseDocClasses

ClassificationMode

Returns used classification mode

DefaultClassifyResult

Returns default DocClass name to which a document is redirected if no other DocClass fits

DefaultLanguage

Returns language used as default

Filename

Returns filename of Project including directory path

ForceValidation

Sets / returns Force Validation

GetVerifierProject

Gets *Verifier* Project

LastAddressPoolUpdate

Returns the time when the address pool was updated last time

Lock

Lock the project for updating

LogScriptMessageEx

Writes a message to a log file.

MinClassificationDistance

Sets / returns minimal distance of classification results

MinClassificationWeight

Sets / returns the minimal classification weight

MinParentClsDistance

Sets / returns minimal distance between classification weight of parent and derived DocClasses

MinParentClsWeight

Sets / returns minimal parent classification weight

MoveDocClass

Moves DocClass specified by Name to new ParentDocClass specified by NewParentName

NoUI

Sets / returns NoUI, to handle if the login dialog is displayed

Page

Returns Cairo Page object of current Project

ParentWindow

Sets the parent window of the login dialog

PerformScriptCommandRTS

Stops and restarts an Runtime Server instance.

PrepareClassification

Prepares the specified document for further extraction process

ShowValidationTemplates

Displays validation templates and their settings in given container

SLWDifferentResultsAction

Sets / returns Action to be done if template classification and supplier extraction have different results

SLWSupplierInvalidIfDifferentClsResults

Sets / returns if Supplier Field is made invalid if template classification and supplier extraction have different results

Unlock

Unlock the project after updating

UpdateAddressPool

Update address analysis pool

ValidationSettingsColl

Returns Collection of all activated validation engines

ValidationTemplates

Returns Collection of all available validation templates

VersionCount

Returns number of versions

WordSegmentationChars

Sets / returns string containing all characters used for Word segmentation

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- AddDocClass
- AnalysisTemplates
- Classify
- ClassifySettings
- Clear
- DeinitScriptEngine
- DeinitScriptModule
- EnableClassificationDebugging
- EnableScriptDebugging
- EvalTemplates
- ExportDocument
- GetSettings
- GetVersionInfo
- HideScript
- HistoryDepth
- ImportDocClass
- InitScriptEngine
- InitScriptModule
- Load
- LoadNetworkData
- LoadVersion
- ProcessDocument
- ProcessFolder
- RemoveDocClass
- RemoveVersion
- RenameDocClass
- RouteDocument
- Save
- SaveNetworkData
- SetLoadError
- SetSaveError
- ShowAnalysisTemplates
- ShowDocClassifyDlg
- ShowEvalTemplates
- ShowLanguageDlg
- ShowMultiPageDlg
- ShowProjectClassifyDlg
- ShowProjectOptDlg
- ShowScript

6.1.2 Methods and Properties of SCBCdrProject Interface

AllClasses

```
AllClasses As ISCBCdrDocClasses (read only)
```

Returns a Collection of all defined DocClasses of this Project. See also ISCBCdrDocClasses and ISCBCdrDocClass for further information.

BaseClasses

```
BaseClasses As ISCBCdrDocClasses (read only)
```

Returns a Collection containing all defined BaseDocClasses. See also ISCBCdrDocClasses and ISCBCdrDocClass for further information.

ClassificationMode

```
ClassificationMode As CDRClassifyMode (read/write)
```

Returns the used classification mode.

DefaultClassifyResult

```
DefaultClassifyResult As String (read/write)
```

Returns the default DocClass name to which a document is redirected if no other DocClass fits.

DefaultLanguage

```
DefaultLanguage As String (read only)
```

Returns the language used as default.

Filename

```
Filename As String (read only)
```

Returns the filename of the Project including the directory path.

ForceValidation

```
ForceValidation As CdrForceValidationMode (read/write)
```

If ForceValidation is set to 'permitted' then the user can overrule the validation by pressing three times on the Return key. If it is set to 'forbidden' then the user cannot change the content of the field disregarding the validation rules. Defaultvalue is 'forbidden'. If is is set to default, then the Force Validation settings of the parent document class are taken.

GetVerifierProject

```
GetVerifierProject (ppVal As Object)
```

This method returns the *Verifier* Project.

ppVal

[out] Verifier Project Object

LastAddressPoolUpdate

```
LastAddressPoolUpdate As Date (read only)
```

This property returns the time when the address pool was updated for the last time.

Lock

```
Lock ()
```

This property locks the Project for updating.

LogScriptMessageEx

```
LogScriptMessageEx( Type As CdrType, Severity As CdrSeverity, Message As String )
```

This method writes a message to a log file.

Type

The message type: information, warning or error.

Severity

Indicates where the message is to be written.

Message

The message to log.

User-defined errors, warnings and informational messages can be written to the application log files reserved for calling applications with custom scripting. Supplementary information is written for every logged entry including the date and time the message was generated, the available virtual memory, the process identifier, and Windows resource usage. The user-defined messages can be added to the custom script via a new “LogScriptMessageEx” method of the Project interface.

All logs are written into **Oracle Forms Recognition**’s log folder where all Runtime Server component-level logs, crash logs, etc. are collected. The log files generated by the Verifier application have file names prefixed with “V_”; by Designer with “D_”; by Runtime Server with “H_” and with “U_” when logging from any other **Oracle Forms Recognition** tool.

The message is always written to the log file but depending on the severity of the message, it can also be sent to the system monitoring service of the Runtime Server or via e-mail to the registered system administrator recipients see section.

Note that notification service (cumulative System Monitoring and e-mail notification) is applicable for Runtime Server instances only and unavailable in other Oracle Forms Recognition applications and tools, where attempt to log with “CDRSeveritySystemMonitoring” or “CDRSeverityEmailNotification” severity will simply function as equivalent of severity “CDRSeverityLogFileOnly”.

For the script-generated log entries of severity “CDRSeveritySystemMonitoring” the corresponding message is going to be forwarded to the cumulative System Monitoring pane of the Runtime Server. For particular Runtime Server instance this is though going to happen only if at least one instance of

System Monitoring service is running on any machine in local area network being correctly configured to accept system notification from the machine (physical server), where the host instance is running and logging messages.

For the script-generated log entries of severity “CDRSeverityEmailNotification” the message is going to be additionally sent via e-mail to the pre-configured list of Oracle Forms Recognition system administrators / professional services members. The mandatory requirement (in addition to the requirements listed above for the system monitoring notification) for such a message to be sent is correct configuration of e-mail sub-system of the System Monitoring service.

Example:

```
Private Sub ScriptModule_PostClassify(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)
   Project.LogScriptMessageEx(CDRTypeError, CDRSeverityEmailNotification, _
      "Critical Error in PostClassify Event when processing document " & pWorkdoc.Filename)
End Sub

Private Sub ScriptModule_PreClassify(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)
   Project.LogScriptMessageEx(CDRTypeInfo, CDRSeverityLogFileOnly, _
      "The classification process has been started for document " & pWorkdoc.Filename)
End Sub
```

MinClassificationDistance

```
MinClassificationDistance As Double (read/write)
```

Sets / returns the minimal distance of classification results.

MinClassificationWeight

```
MinClassificationWeight As Double (read/write)
```

Sets / returns the minimal classification weight.

MinParentClsDistance

```
MinParentClsDistance As Double (read/write)
```

Sets / returns the minimal distance between the classification weight of the parent and the derived DocClasses.

MinParentClsWeight

```
MinParentClsWeight As Double (read/write)
```

Sets / returns minimal parent classification weight. This value is used as threshold during parent classification.

MoveDocClass

```
MoveDocClass (Name As String, NewParentName As String)
```

Moves a DocClass specified by its Name to a new ParentDocClass specified by NewParentName.

Name

Name of moved DocClass

NewParentName

Name of new ParentDocClass

NoUI

```
NoUI As Boolean (read/write)
```

This property sets or returns NoUI. If NoUI set to true, then no login dialog is displayed.

Page

```
Page As ISCBCroPage (read only)
```

Returns Cairo Page object of current Project. This Page is used to define OCR settings which are used for executing OCR on all DocPages.

ParentWindow

```
ParentWindow As Long (write only)
```

This property sets the parent window of the login dialog.

lhWnd

[in] Window handle of windows operating system.

PerformScriptCommandRTS

```
PerformScriptCommandRTS ( CommandID As Long, Param1 As Long,
                          Param2 As Long, Description As String )
```

This method allows commands to be invoked to control Runtime Server instances.

CommandID

Identifier of the command to execute. 0 = stop instance, 1 = start instance.

Param1

Type of message to log when the command executes:

0 = informational message,

1 = warnings

2 = error messages. Note that error messages are additionally forwarded to MMC administration console of the Runtime Server.

Param2

User error code of the message.

Description

Description of the message to log in the common Runtime Server log file and in the case of error messages on the MMC administration console.

Example:

```
‘ This script code stops document processing for the current Runtime Server
‘ instance and logs specified message as error with error code “777”
Project.PerformScriptCommandRTS 0, 2, 777, "RTS is going to be stopped from Custom Script"

‘ This script code restarts the current Runtime Server instance and logs
‘ specified message as warning with error code “999”
Project.PerformScriptCommandRTS 1, 1, 999, "RTS is going to be restarted from Custom Script"
```

PrepareClassification

```
PrepareClassification (pWorkdoc As ISCBCdrWorkdoc)
```

This method prepares the specified document for further extraction process, for example, creates document fields according to the document class. This method should be used after the classification is finished. Note that usually this method is executed automatically by the system as the first phase of extraction process, but, in principle, it could be used in case further extraction is not required.

pWorkdoc

Reference to the currently processed Workdoc.

ShowValidationTemplates

```
ShowValidationTemplates (pContainer As ISCBCdrPPGContainer)
```

This method is used to display the validation templates and their settings in a given container.

pContainer

Container used to save the validation templates and their settings.

SLWDifferentResultsAction

```
SLWDifferentResultsAction As CdrSLWDifferentResultsAction (read/write)
```

This property sets or returns the action to be done if a template classification and supplier extraction have different results.

SLWSupplierInvalidIfDifferentClsResults

```
SLWSupplierInvalidIfDifferentClsResults As Boolean (read/write)
```

This property sets or returns if a Supplier Field is made invalid when the template classification and supplier extraction have different results. If the results are different, the field is set to invalid and the property value is true.

Unlock

```
Unlock ()
```

This method unlocks the Project after updating.

UpdateAddressPool

```
UpdateAddressPool ()
```

This methods is used to update the address analysis pool.

ValidationSettingsColl

```
ValidationSettingsColl As ISCBCroCollection (read only)
```

This property returns a collection of all activated validation engines.

ValidationTemplates

```
ValidationTemplates As ISCBCroCollection (read only)
```

This property returns a collection of all available validation templates.

VersionCount

```
VersionCount (Filename As String) As Long (read only)
```

Returns the number of versions available for specified filename.

Filename

Name of the file.

WordSegmentationChars

```
WordSegmentationChars As String (read/write)
```

Sets / returns a string containing all characters used for Word segmentation. If one of these characters is detected in a Word it is splitted.

6.2 SCBCdrDocClass

6.2.1 SCBCdrDocClass Interface (Cedar DocClass object)

A Cedar DocClass object represents a single document class within a Cedar project class hierarchy.

6.2.1.1 Type Definitions

CdrFocusChangeReason

This enumeration defines the reason for the focus change of a *Verifier* field edit.

CdrEnterPressed	Focus changed by pressing Enter
CdrFcrCandidateCopied	Focus changed (refreshed) because a candidate and its location was copied to the field
CdrFcrRefreshed	Focus changed (refreshed) because the selection area and its location was copied to the field
CdrFcrSelectionCopied	Focus changed (refreshed) because the selection area and its location was copied to the field
CdrFcrWordCopied	Focus changed (refreshed) because a word and its location was appended to the field
CdrFormLoaded	Focus changed because of loading form
CdrMouseClicked	Focus changed because of mouse click
CdrSelectedOutside	Focus changed because of some selection outside
CdrTableCellSelected	Focus changed because of the selection of a Table cell
CdrTabPressed	Focus changed because of pressing Tab key
CdrUnknownReason	Focus changed because of an unknown reason

CDRsiModule

This type defines the module in which the smart index definition should be used.

CDRsiModuleDistiller	Use smart indexing in automatic Field extraction
CDRsiModuleDistVer	Use smart indexing in automatic Field extraction and manual Field validation
CDRsiModuleVerifier	Use smart indexing in manual Field validation

CdrForceValidationMode

This enumeration defines the different options for the ForceValidation.

CdrForceValDefault	CdrForceValidationModeDefault: ForceValidationMode inherited
CdrForceValForbidden	CdrForceValidationModeForbidden: ForceValidation (3*return) not allowed
CdrForceValPermitted	CdrForceValidationModePermitted: ForceValidation (3*return) allowed

6.2.1.2 List of Methods and Properties

ClassificationField

Sets / returns name of field that is used for classification

ClassificationRedirection

Returns DocClass name to which classification result is redirected

ClassifySettings

Collection of chosen classification engines and their settings

DerivedDocClasses

Returns Collection of directly derived DocClasses

DisplayName

Sets / returns display name of DocClass, currently not used

Fields

Returns FieldDefs of DocClass as Collection

ForceSubtreeClassification

Sets / returns if classification to sub tree of this DocClass is forced

ForceValidation

Sets / returns ForceValidation

GetFieldAnalysisSettings

Returns field analysis settings

Hidden

Sets / returns if DocClass is hidden

InitField

Reinitializes a required field in workdoc

ManualTableTrainingMode

Sets / returns option for manual BTE training mode

Name

Sets / returns name of Document class

Page

Returns Page object

Parent

Returns parent DocClass of actual DocClass

ShowClassValidationDlg

Displays property page of validation settings for this class

ShowFieldValidationDlg

Displays property page of validation settings for specified field name

ShowGeneralFieldPPG

Starts field settings property page specifying the active tab

SubtreeClsMinDist

Sets / returns minimal distance to classification weight of derived DocClasses

SubtreeClsMinWeight

Sets / returns minimal classification weight of derived DocClasses

UseDerivedValidation

Sets / returns if derived validation rules are used

ValidationSettingsColl

Returns collection containing all activated validation engines

ValidationTemplateName

Sets / returns name of validation template

ValidClassificationResult

Sets / returns if DocClass is valid classification result

VisibleInCorrection

Sets / returns if DocClass is visible in *Verifier*

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- AddFieldDef
- AnalyzeDocument
- AnalyzeField
- EvaluateDocument
- EvaluateField
- Export
- Extract
- FocusChanged
- FormatField
- OnAction
- PrepareExtraction
- RemoveFieldDef
- RenameFieldDef
- ShowAnalysisDlg
- ShowEvalDlg
- SmartIndex
- SmartIndexField
- StaticPageCount
- Validate
- ValidateField

6.2.2 Methods and properties of SCBCdrDocClass Interface

ClassificationField

```
ClassificationField As String (read/write)
```

This property is used either to read or to write the name of the field that is used for the classification.

ClassificationRedirection

```
ClassificationRedirection As String (read/write)
```

In case of a classification to this class the document can be redirected to another DocClass. This property returns the name of the target DocClass. The redirection occurs at the end of classification.

ClassifySettings

```
ClassifySettings As ISCBCroCollection (read only)
```

Collection of chosen classification engines and their settings for this DocClass.

DerivedDocClasses

```
DerivedDocClasses As ISCBCdrDocClasses (read only)
```

Returns a collection of all DocClasses derived directly from this DocClass. If no further DocClasses are derived Nothing is returned.

DisplayName

```
DisplayName As String (read/write)
```

Sets / returns the display name of the DocClass currently not used, if nothing inserted here the DocClass name is used.

Fields

```
Fields As ISCBCdrFieldDefs (read only)
```

The FieldDef Collection provides access to FieldDefs of a DocClass. See the definition of SCBCdrFieldDefs for details of the FieldDefs collection and the definition of SCBCdrFieldDef for details for accessing the properties of a FieldDef.

ForceSubtreeClassification

```
ForceSubtreeClassification As Boolean (read/write)
```

Sets / returns if the classification to the sub tree of this DocClass is forced.

ForceValidation

```
ForceValidation As CdrForceValidationMode (read/write)
```

If ForceValidation is set to 'permitted' then the user can overrule the validation by pressing three times on the Return key. If it is set to 'forbidden' then the user cannot change the content of the field disregarding the validation rules. Defaultvalue is 'forbidden'. If is is set to default, then the Force Validation settings of the parent document class are taken.

GetFieldAnalysisSettings

```
GetFieldAnalysisSettings (FieldName As String, Language As String,
         ppAnalysisSettings As ISCBCdrAnalysisSettings)
```

This property returns the analysis settings for the document class.

FieldName

The name of the field for which the analysis settings are retrieved.

ppAnalysisSettings

The name of the analysis settings object that is used in the code to assign the settings to, see script sample.

Example:

```
' This script samples shows how to retrieve the analysis settings
' to assign them for example to be used for the associative
' search engine

Dim theDocClass As SCBCdrDocClass
Dim theAnalysisSettings As ISCBCdrAnalysisSettings
Dim theSupplierSettings As Object

Set theDocClass=Project.AllClasses.ItemByName (pWorkdoc.DocClassName)
' Get the settings for the field VendorName
theDocClass.GetFieldAnalysisSettings "VendorName","German", theAnalysisSettings
Set theSupplierSettings = theAnalysisSettings
```

Hidden

```
Hidden As Boolean (read/write)
```

Specifies if the DocClass should be visible in the Project designer.

InitField

```
InitField (pWorkdoc As ISCBCdrWorkdoc, pField As ISCBCdrField)
```

This method reinitializes a required field in workdoc. The Field object is cleared.

pWorkdoc

Current workdoc.

pField

Field to be cleared.

ManualTableTrainingMode

```
ManualTableTrainingMode As Boolean (read/write)
```

This property sets or returns the option for manual Table Extraction training mode. The default value is FALSE, so that tables are trained automatically, not considering user training actions applied during interactive learning.

Name

```
Name As String (read/write)
```

This property reads or write the name of the Document Class.

Page

```
Page As ISCBCroPage (read only)
```

Returns the Page object of this DocClass with all defined zones and their OCR settings.

Parent

```
Parent As ISCBCdrDocClass (read only)
```

Returns the parent DocClass of the actual DocClass. If the chosen DocClass is a BaseClass the property returns Nothing.

ShowClassValidationDlg

```
ShowClassValidationDlg (pContainer As ISCBCdrPPGContainer)
```

This method displays the property page of validation settings for this document class.

pContainer

Container in which the property page should be displayed.

ShowFieldValidationDlg

```
ShowFieldValidationDlg (FieldName As String, pContainer As ISCBCdrPPGContainer)
```

This method displays the property page of the validation settings for the specified field name.

FieldName

Field for which the dialog is shown.

pContainer

Container in which the property page should be displayed.

ShowGeneralFieldPPG

```
ShowGeneralFieldPPG (FieldName As String, TabIndexActive As Long,
         pContainer As ISCBCdrPPGContainer)
```

Starts field settings property page specifying the active tab

FieldName

Field for which the dialog is shown.

TabIndexActive

Zerobased Index for the tab that should be displayed.

pContainer

Container in which the property page should be displayed.

SubtreeClsMinDist

```
SubtreeClsMinDist As Double (read/write)
```

Returns the minimal distance to the classification weight of the derived DocClasses.

SubtreeClsMinWeight

```
SubtreeClsMinWeight As Double (read/write)
```

Sets / returns the minimal classification weight of the derived DocClasses.

UseDerivedValidation

```
UseDerivedValidation As Boolean (read/write)
```

This property sets or returns a Boolean value, when derived validation rules are used. Defaultvalue for a new inserted field is False.

ValidationSettingsColl

```
ValidationSettingsColl As ISCBCroCollection (read only)
```

This property returns a collection of all activated validation engines.

ValidationTemplateName

```
ValidationTemplateName As String (read/write)
```

This property sets or returns the name of the validation template.

ValidClassificationResult

```
ValidClassificationResult As Boolean (read/write)
```

Sets / returns if this DocClass is a valid classification result or if it is omitted for classification.

VisibleInCorrection

```
VisibleInCorrection As Boolean (read/write)
```

Sets / returns if this DocClass is visible in *Verifier*.

6.3 SCBCdrDocClasses

6.3.1 SCBCdrDocClasses Interface (Cedar DocClass Collection)

This Collection contains all defined DocClass objects of the Cedar Project.

6.3.1.1 List of methods and properties

Collection
Returns Collection internally used to store DocClasses

Count
Returns item count of Collection

Item
Returns item from Collection specified by index [1-Count] or name

ItemByIndex
Returns item from Collection specified by index [1-Count]

ItemByName
Returns item from Collection specified by name

ItemExists
Returns TRUE if item with specified name exists

ItemIndex
Returns index of item specified by name [1-Count]

ItemName
Returns name of item specified by index [1-Count]

Tag
Tag to store arbitrary data during runtime specified by index [1-Count]

6.3.2 Methods and properties of SCBCdrDocClasses Interface

Collection

```
Collection As ISCBCroCollection (read only)
```

Returns the Collection which is internally used to store the DocClasses. The access to the Collection my be necessary if you want to connect to receive the change events.

Count

```
Count As Long (read only)
```

Returns the number of items within the Collection.

Item

```
Item (Index As Variant) As ISCBCdrDocClass (read only)
```

Returns a specified item from the Collection. The Item property is the default property of the ISCBCroCollection interface. Using this Method pay attention to release the returned interface when it is no longer needed.

Index

[in] The index can either be a Long value specifying the index within the collection, valid range from 1 to Count, or a String specifying the item by name.

ItemByIndex

```
ItemByIndex (Index As Long) As ISCBCdrDocClass (read only)
```

Returns an item from the Collection specified by index.

When this Method is called, pay attention to release the returned interface when it is no longer needed.

Index

[in] Index of the item to retrieve from the Collection, valid range from 1 to Count

ItemByName

```
ItemByName (Name As String) As ISCBCdrDocClass (read only)
```

Returns an item from the Collection specified by name.

When this Method is called, pay attention to release the returned interface when it is no longer needed.

Name

[in] Name of the item to retrieve from the Collection.

ItemExists

```
ItemExists (Name As String) As Boolean
```

Returns TRUE if an item with specified name exists inside the Collection or FALSE is returned.

Name

[in] Name of item to search for.

ItemIndex

```
ItemIndex (Name As String) As Long (read only)
```

The index of an item specified by name is returned.

Name

[in] Name specifying an item in the Collection.

ItemName

```
ItemName (Index As Long) As String (read only)
```

The name of an item specified by index is returned.

Index

[in] Index specifying an item in the Collection, valid range from 1 to Count

Tag

```
Tag (Index As Long) As Variant (read/write)
```

This tag can be used to store a variant for each item of the Collection. This value will not be saved with the Project.

Index

Specifies the item index, valid from 1 to Count

6.4 SCBCdrFieldDef

6.4.1 SCBCdrFieldDef Interface (Cedar FieldDef object)

A Cedar FieldDef object represents the definition of a single FieldDef inside a Cedar DocClass.

6.4.1.1 Type Definitions

CdrFieldFormat

This type defines the default format of a certain field. (Not yet implemented)

CdrFieldFormatCurrency	CdrFieldFormatCurrency
CdrFieldFormatDate	CdrFieldFormatDate
CdrFieldFormatExtNumber	CdrFieldFormatExtNumber
CdrFieldFormatNone	CdrFieldFormatNone
CdrFieldFormatNumber	CdrFieldFormatNumber

CDRFieldType

This type defines the type of a FieldDef.

CDRFieldTypeTable	The Field type is Table.
CDRFieldTypeText	The Field type is text, which may be single or multi-line text.

CdrForceValidationMode

This enumeration defines the different options for the ForceValidation.

CdrForceValDefault	CdrForceValidationModeDefault: ForceValidationMode inherited
CdrForceValForbidden	CdrForceValidationModeForbidden: ForceValidation (3*return) not allowed
CdrForceValPermitted	CdrForceValidationModePermitted: ForceValidation (3*return) allowed

CdrValFieldType

This enumeration contains different validation types for fields.

CdrAmountValidation	Used for amount values or general numeric values.
CdrChkboxValidation	Field as used check box.
CdrCustomValidation	TBD
CdrDateValidation	Used for date values.
CdrListValidation	Used for lists.
CdrTableValidation	Used for tables.
CdrTextValidation	Used for text values, strings.

6.4.1.2 List of Methods and Properties

AlwaysValid

Sets / returns if content of FieldDef is always valid

AnalysisTemplate

Returns name of analysis template

AppendListItem

Adds a new list item and returns a new item index for it

ColumnCount

Returns number of Table columns

ColumnName

Returns name of Table column specified by zero-based index

DefaultValidationSettings

Returns validation settings with default language

Derived

Returns TRUE if FieldDef properties are derived from upper DocClass

DisplayName

Returns displayed name of Field

EvalSetting

Sets / returns activated evaluation engine and its settings

EvalTemplate

Returns name of evaluation template

FieldID

Returns internally used FieldID

FieldType

Sets / returns type of FieldDef (Text or Table)

ForceValidation

Sets / returns ForceValidation

ListItem
Sets / returns list item string for index

ListItemCount
Returns number of strings in the list

MaxLength
Returns maximal number of characters permitted

MinLength
Sets / returns minimal number of characters

Name
Sets / returns name of FieldDef

NoRejects
Sets / returns if rejects permitted

OCRConfidence
Sets / returns confidence level for OCR valid from 0 to 100

RemoveListItem
Removes a list item by its index

SmartIndex
Sets / returns SmartIndex definitions

UseDerivedOCRSettings
Sets / returns if OCR settings of parent DocClass are used

UseDerivedValidation
Sets / returns if derived validation rules are used

UseMaxLen
Sets / returns if maximal number of characters is limited to value given by MaxLength

UseMinLen
Sets / returns if usage of minimal number of characters given by property MinLength is activated

ValidationSettings
Sets / returns chosen validation engine and its settings

ValidationTemplate
Returns name of validation template

ValidationType
Returns ValidationType

VerifierColumnWidth

Sets / returns column width of Table specified by zero-based index

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- AnalysisSetting
- OCRSettings
- ShowStdPPG

6.4.2 Methods and properties of SCBCdrFieldDef Interface

AlwaysValid

```
AlwaysValid As Boolean (read/write)
```

Sets / returns if the content of this FieldDef is always valid. No further examinations of the result are made.

AnalysisTemplate

```
AnalysisTemplate (Language As String) As String (read only)
```

Returns the name of the analysis template if used.

Language

Language parameter

AppendListItem

```
AppendListItem (bstrItem As String) As Long
```

This method adds a new list item and returns a new item index for it.

bstrItem

String inserted into the list.

ColumnCount

```
ColumnCount As Long (read only)
```

Returns the number of Table columns if FieldType is Table.

ColumnName

```
ColumnName (ColumnIndex As Long) As String (read only)
```

Returns the name of a Table column if FieldType is Table.

ColumnIndex

Zero-based index of the Table column

DefaultValidationSettings

```
DefaultValidationSettings As ISCBCdrValidationSettings (read only)
```

This property returns the validation settings with default language.

ppValSettings

ValidationSettings object for the default language.

Derived

```
Derived As Boolean (read only)
```

Returns TRUE if the FieldDef properties are derived from an upper DocClass.

DisplayName

```
DisplayName As String (read/write)
```

The DisplayName can be different from the FieldDef name and does not have any restrictions about the used character set while the FieldDef name must be a valid basic name. An application may use the DisplayName instead of the FieldDef name to show a more readable name of the FieldDef.

EvalSetting

```
EvalSetting (Language As String) As Object (read/write)
```

Sets / returns activated evaluation engine and its settings.

Language

Language parameter

EvalTemplate

```
EvalTemplate (Language As String) As String (read only)
```

Returns the name of the evaluation template if used.

Language

Language of Project

FieldID

```
FieldID As Long (read only)
```

This read-only property returns the internally used FieldID.

FieldType

```
FieldType As CDRFieldType (read/write)
```

This property sets or returns the type of the FieldDef. It is either a text or a table.

ForceValidation

```
ForceValidation As CdrForceValidationMode (read/write)
```

This property sets or returns the mode for the ForceValidation.

ListItem

```
ListItem (lIndex As Long) As String (read/write)
```

This property sets or returns a list item string for a given index.

lIndex

Zero-based index.

ListItemCount

```
ListItemCount As Long (read only)
```

This property returns the number of strings in the ListItem list.

MaxLength

```
MaxLength As Long (read/write)
```

Returns the maximal number of characters permitted for this FieldDef.

MinLength

```
MinLength As Long (read/write)
```

Sets / returns the minimal number of characters for this FieldDef.

Name

```
Name As String (read/write)
```

Sets / returns the name of the FieldDef.

NoRejects

```
NoRejects As Boolean (read/write)
```

Sets / returns if rejects are permitted.

OCRConfidence

```
OCRConfidence As Long (read/write)
```

Sets / returns the confidence level for OCR. The value must be between 0 and 100.

RemoveListItem

```
RemoveListItem (lIndex As Long)
```

This property removes a list item by its index.

lIndex

Index of entry to be removed from the list.

SmartIndex

```
SmartIndex As ISCBCdrSmartIndex (read/write)
```

The SmartIndex property contains all definitions about smart indexing.

UseDerivedOCRSettings

```
UseDerivedOCRSettings As Boolean (read/write)
```

Sets / returns if OCR settings of the parent DocClass are used.

UseDerivedValidation

```
UseDerivedValidation As Boolean (read/write)
```

Sets / returns if the derived validation rules are used for validation of this FieldDef.

UseMaxLen

```
UseMaxLen As Boolean (read/write)
```

Sets / returns if the maximal number of characters is limited to the value given by MaxLength.

UseMinLen

```
UseMinLen As Boolean (read/write)
```

Sets / returns if the usage of the minimal number of characters given by the property MinLength is activated.

ValidationSettings

```
ValidationSettings (Language As String) As ISCBCdrValidationSettings (read/write)
```

This property sets or returns the chosen validation engine and its settings.

Language

Defines the language for classification, extraction and validation.

ValidationTemplate

```
ValidationTemplate (Language As String) As String (read only)
```

This property returns the name of validation template.

Language

Defines the language for classification, extraction and validation.

ValidationType

```
ValidationType As CdrValFieldType (read only)
```

This property returns the type of validation.

VerifierColumnWidth

```
VerifierColumnWidth (ColumnIndex As Long) As Long (read/write)
```

Sets /returns the width of the specified column of the Table. This value will be used from *Verifier* to display a table assigned to this FieldDef.

ColumnIndex

Zero-based Index of the Table column

6.5 SCBCdrFieldDefs

6.5.1 Interface of SCBCdrFieldDefs Interface (Cedar FieldDef Collection)

This Collection contains all defined FieldDef objects of a single DocClass.

6.5.1.1 List of Methods and Properties

Collection

Returns Collection internally used to store FieldDefs

Count

Returns number of items within Collection

Item

Returns item from Collection specified by index (ID [1-Count] or name)

ItemByIndex

Returns item from Collection specified by index

ItemByName

Returns item from Collection specified by name

ItemExists

Returns TRUE if item with specified name exists

ItemIndex

Returns index of item specified by name [1-Count]

ItemName

Returns name of item specified by index [1-Count]

Tag

Tag to store arbitrary data during runtime specified by Index [1 to Count]

6.5.2 Methods and Properties of SCBCdrFieldDefs Interface

Collection

```
Collection As ISCBCroCollection (read only)
```

Returns the Collection which is internally used to store the FieldDefs. The access to the Collection may be necessary if you want to connect to the Collection to receive the change events.

Count

```
Count As Long (read only)
```

Returns the number of items within the FieldDef Collection.

Item

```
Item (Index As Variant) As ISCBCdrFieldDef (read only)
```

Returns a specified item from the Collection. The Item property is the default property of the ISCBCroCollection interface.

Index

[in] The index can either be a Long value specifying the index (1 to Count) within the Collection or a String specifying the item by name.

ItemByIndex

```
ItemByIndex (Index As Long) As ISCBCdrFieldDef (read only)
```

Returns an item from the Collection specified by index.

Index

[in] Index of the item to retrieve from the Collection.

ItemByName

```
ItemByName (Name As String) As ISCBCdrFieldDef (read only)
```

Returns an item from the Collection specified by name.

Name

[in] Name of the item to retrieve from the Collection.

ItemExists

```
ItemExists (Name As String) As Boolean
```

Returns TRUE if an item with specified name exists inside the Collection or FALSE is returned.

Name

[in] Name of item to search for.

ItemIndex

```
ItemIndex (Name As String) As Long (read only)
```

The index of an item specified by name is returned.

Name

[in] Name specifying an item in the Collection.

ItemName

```
ItemName (Index As Long) As String (read only)
```

The name of an item specified by index is returned.

Index

[in] Index specifying an item in the Collection, valid range form 1 to Count

Tag

```
Tag (Index As Long) As Variant (read/write)
```

This tag can be used to store a variant for each item of the Collection. This value will not be saved with the Project.

Index

Specifies the item index, valid range from 1 to Count

6.6 SCBCdrSettings

6.6.1 Interface of SCBCdrSettings Interface (Cedar Settings object)

The Cedar Settings object stores arbitrary strings for usage in script.

6.6.1.1 List of Methods and Properties

ActiveClient

Sets / returns name of active client

AddClient

Adds new client with specified name to Settings object

AddKey

Adds key specified by its name and its Parent (empty string in case of BaseKey)

Clear

Clears all clients and keys from Settings object

Client

Returns name of specified client specified by zero-based index

ClientCount
Returns client count

GlobalLearnsetPath
Sets / returns global learn set path

Key
Returns key specified by zero-based index

KeyCount
Returns count of keys

KeyIcon
Sets / returns value for the key specified by its name

KeyParent
Returns parent name of specified zero-based key index

MoveKey
Moves key specified by its name to NewParent

ProjectFileName
Sets / returns file name of Project

RemoveClient
Removes client specified by name

RemoveKey
Removes key specified by its name

SupervisedLearningDisabled
Sets / returns value of supervised learning in *Designer* and local *Verifier* workstations

TopDownEventSequence
Sets / returns value of top-down event sequence

Value
Sets/ returns value of ***specified*** key

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- Load
- Save

6.6.2 Methods and properties of SCBCdrSettings Interface

ActiveClient

```
ActiveClient As String (read/write)
```

Sets / returns name of the currently active client

AddClient

```
AddClient (newVal As String)
```

Adds a new client with the specified name to the current Settings object. The clients and their key values can be configured in *Designer*, project settings, project settings tree tab.

newVal

Name of new client

AddKey

```
AddKey (newVal As String, Parent As String)
```

Adds a new key specified by its name and its Parent. The keys can be configured in *Designer*, project settings, project settings tree tab.

newVal

New key name

Parent

Name of the parent key, in case of a new base key use an empty string for the Parent.

Clear

```
Clear ()
```

Clears all clients and keys from the Settings object. The keys and the clients can be configured in *Designer*, project settings, project settings tree tab.

Client

```
Client (Index As Long) As String (read only)
```

Returns the name of the specified client.

Index

Zero-based client index.

ClientCount

```
ClientCount As Long (read only)
```

Returns the number of clients.

GlobalLearnsetPath

```
GlobalLearnsetPath As String (read/write)
```

This property sets or returns the global learn set path.

Key

```
Key (Index As Long) As String (read only)
```

Returns the key name specified by index. The keys can be configured in *Designer*, project settings, project settings tree tab.

Index

Zero-based index of the key.

KeyCount

KeyCount As Long (read only)

Returns the number of keys. The keys can be configured in *Designer*, project settings, project settings tree tab.

KeyIcon

```
KeyIcon (Key As String) As String (read/write)
```

This property sets new value for the specified key or returns the key's value. The keys can be configured in *Designer*, project settings, project settings tree tab.

Key

Name of the key.

KeyParent

```
KeyParent (Index As Long) As String (read only)
```

Returns the parent name of specified key index. The keys can be configured in *Designer*, project settings, project settings tree tab.

Index

Zero-based key index.

MoveKey

```
MoveKey (Key As String, NewParent As String)
```

Moves a key specified by its name to the NewParent specified by its name. The keys can be configured in *Designer*, project settings, project settings tree tab.

Key

Name of key that should be moved

NewParent

Name of new parent, empty string in case of moving it as a base key

ProjectFileName

```
ProjectFileName As String (read/write)
```

This property sets or returns the file name of the Project.

RemoveClient

```
RemoveClient (ClientName As String)
```

Removes a client specified by its name. The clients and their key values can be configured in *Designer*, project settings, project settings tree tab.

ClientName

Name of client that should be removed

RemoveKey

```
RemoveKey (KeyName As String)
```

Removes a key specified by its name. The keys can be configured in *Designer*, project settings, project settings tree tab.

KeyName

Name of key that is removed.

SupervisedLearningDisabled

```
SupervisedLearningDisabled As Boolean (read/write)
```

This property sets or returns the value of supervised learning in *Designer* and local *Verifier* workstations. For the default value 'True', Supervised Learning is disabled.

TopDownEventSequence

```
TopDownEventSequence As Boolean (read/write)
```

This property sets or returns the value of top-down event sequence. For 'True' the proccessing chronology is set to top-down else bottom-up.

Value

```
Value (Key As String, Parent As String, Client As String) As String (read/write)
```

Returns the value of the specified key.

Key

Key name, which is assigned to the value.

Parent

Parent name of the key.

Client

Name of the client. Can be an empty string. In that case the active client will be used.

Example:

```
MyDBPath = Settings.Value(“DatabaseName”, “”, “”)
```

```
' now we can open the database
DB.Open(MyDBPath, …)
```

6.7 SCBCdrScriptModule

6.7.1 SCBCdrScriptModule Interface (Cedar ScriptModule object)

This is a global object at the project level. All the script module events occurred at project level belongs to this object.

6.7.1.1 List of Methods and Properties

ModuleName

Returns name of module that initialized ScriptModule

ReadZone

Reads on SCBCroImage object a zone specified by name and returns result as string

ReadZoneEx

Reads on SCBCroImage object zone specified by name and returns SCBCroWorktext object as result

6.7.2 Methods and Oroperties of SCBCdrScriptModule Interface

ModuleName

```
ModuleName As String (read only)
```

Returns the name of the module that initialized ScriptModule, 'Designer', 'Server' or 'Verifier' in case of Oracle Forms Recognition Designer, 'Server' in case of Oracle Forms Recognition Runtime Server, 'Verifier' in case of Oracle Forms Recognition Verifier is returned.

ReadZone

```
ReadZone (Image As ISCBCroImage, ZoneName As String) As String
```

This method can be used to read a zone on a CroImage object, which settings are saved before in the ScanJobs' definition. The settings of the CroZone can be selected by its name. The result of the method will be returned as a string.

Image

[in] SCBCroImage object

ZoneName

[in] Name of Zone which is read

ReadZoneEx

```
ReadZoneEx (Image As ISCBCroImage, ZoneName As String, Result As ISCBCroWorktext)
```

This method can be used to read a zone on a CroImage object, which settings are saved before in the ScanJobs' definition. The settings of the zone can be selected by the name of the zone. The result of the method will be a SCBCroWorktext object.

Image

[in] SCBCroImage object

ZoneName

[in] Name of read zone

Result

[in] Result of reading returned as SCBCroWorktext object

6.8 SCBCdrScriptAccess

6.8.1 SCBCdrScriptAccess Interface (Cedar ScriptAccess object)

Oracle Forms Recognition provides a public interface “SCBCdrScriptAccess” for external access to the project and class level custom script pages. The new interface can be queried from the main “SCBCdrProject” interface available in Oracle Forms Recognition custom script. Using this interface it is possible to retrieve, modify and dump project and class level scripts.

6.8.1.1 List of Methods and Properties

DumpAllPages

Dumps all script pages available in the project as a Unicode text file. Output is either to the specified file name or, when empty, to the project file location using the project’s file name with “.sax” extension.

ExportAllPages

CURRENTLY NOT SUPPORTED. Exports all available script pages in a reimportable format to the specified folder.

ExportClassPage

CURRENTLY NOT SUPPORTED. Exports the specified script page to a script dump file.

GetPageCode

Retrieves the project or specified class level script code. When the ClassName parameter is empty, returns the project level script.

ImportAllPages

CURRENTLY NOT SUPPORTED. Imports all available script pages using script dumps from the specified folder.

ImportClassPage

CURRENTLY NOT SUPPORTED. Imports the specified script page from a script dump file.

SetPageCode

Assigns the project or specified class level script code. When the ClassName parameter is empty, sets the project level script.

6.8.2 Methods and Properties of SCBCdrFieldDefs Interface

DumpAllPages

```
DumpAllPages(FileName As String)
```

Dumps all script pages available in the project as a Unicode text file. Output is either to the specified file name or, when empty, to the project file location using the project's file name with ".sax" extension.

FileName

[in] name of the dump file.

ExportAllPages

```
ExportAllPages(FolderName As String)
```

CURRENTLY NOT SUPPORTED. Exports all available script pages in a reimportable format to the specified folder.

FolderName

[in] name of the folder to save the script pages to.

ExportClassPage

```
ExportClassPage(FolderName As String, ClassName As String)
```

CURRENTLY NOT SUPPORTED. Exports the specified script page to a script dump file.

FolderName

[in] name of the folder to save the script page to.

ClassName

[in] name of the class to export the script for.

GetPageCode

```
GetPageCode(ClassName As String, ScriptCode As String)
```

Retrieves the project or specified class level script code. When the ClassName parameter is empty, returns the project level script.

ClassName

[in] name of the class.

ScriptCode

[out] class script code.

ImportAllPages

```
ImportAllPages(FolderName As String)
```

CURRENTLY NOT SUPPORTED. Imports all available script pages using script dumps from the specified folder.

FolderName

[in] name of the folder to load the script pages from.

ImportClassPage

```
ImportClassPage(FolderName As String, ClassName As String)
```

CURRENTLY NOT SUPPORTED. Imports the specified script page from a script dump file.

FolderName

[in] name of the folder to load the script page from.

ClassName

[in] name of the class to import the script for.

SetPageCode

```
SetPageCode(ClassName As String, ScriptCode As String)
```

Assigns the project or specified class level script code. When the ClassName parameter is empty, sets the project level script.

ClassName

[in] name of the class.

ScriptCode

[out] class script code.

6.8.2.1 Usage Example

The script example below shows how the current project's script code pages can be modified, retrieved or dumped to a Unicode text file:

```
Dim strClassName As String
Dim strProjectCode As String
Dim strClassCode As String
Dim theScriptAccess As SCBCdrPROJLib.SCBCdrScriptAccess

' Queries "Script Access" interface
Dim theObject As Object
Set theObject = Project
Set theScriptAccess = theObject

' Retrieves entire script code (including custom references) of the "My Class Name"
' class level script page
strClassName = "My Class Name"
theScriptAccess.GetPageCode strClassName, strClassCode

' Composes extension to the existing script
strClassCode = strClassCode + vbCrLf + vbCrLf + "Private Sub ShowMessage(strMessage As String)"
     + vbCrLf + vbTab + "MsgBox strMessage, vbOkOnly" + vbCrLf + "End Sub" + vbCrLf

' Assignes the extended script to the same class level script page in order to modify it
' with adding desired new function
theScriptAccess.SetPageCode strClassName, strClassCode
```

```
' Dumps all available script pages (including custom references) in Unicode format
theScriptAccess.DumpAllPages ""

' Retrieves project level script code
theScriptAccess.GetPageCode "", strProjectCode
```

Chapter 7 Oracle Forms Recognition Analysis Engines Object Reference

7.1 SCBCdrAssociativeDbExtractionSettings

7.1.1 Interface SCBCdrAssociativeDbExtractionSettings Interface (Associative Database Extraction Settings Interface)

This interface covers all methods and properties that are required for controlling and accessing the new universal format of the ASSA engine's pool.

7.1.1.1 Type Definitions

CdrAutoUpdateType

This enumeration is used to specify the automatic import property.

CdrAUTFile	Automatic import from file for associative search field.
CdrAUTNone	No automatic import for associative search field.
CdrAUTODBC	Automatic import from ODBC source for associative search field.

7.1.1.2 List of Methods and Properties

AddColumn

Adds a new column field to the Pool

AddPhrase

Appends a new phrase to list of phrases to be used for address analysis

ChangeEntry

Updates or inserts the specified column with the entry data

ClassNameFormat

Sets or reads format definition for a document class name

ColumnCount

Returns number of columns of currently opened pool

ColumnName

Sets / returns the name of column

CommitAddEntry

After execution of StartAddEntry and ChangeEntry changes take effect

CommitUpdate

Closes and saves the currently opened pool

EnableCandidateEvaluation
Sets / returns if candidate evaluation permitted

EntryCount
Returns the number of entries of the pool

EvalFirstPageOnly
Sets / returns if candidate evaluation is processed only for the first page

FieldContentsFormat
Sets / returns format definition for the representation of the engine results

FindLocation
Sets / returns if address analysis is enabled

GeneratePool
Imports the pool from specified source

GeneratePoolFromCsvFile
Removes previous pool and generates a new one using CSV file

GeneratePoolFromODBC
Removes previous pool and generates a new one using ODBC source

GetClassNameByID
Returns formatted document class name for the pool entry specified by its unique ID

GetEntry
Returns the content of a field that is specified by its index and column name

GetFormattedValueByID
Returns formatted entry representation for the pool entry specified by its unique ID

GetIDByIndex
Returns unique ID of an entry by index

GetIndexByID
Returns index of an entry by its unique ID

GetSearchArea
Returns area on the document to search in

IdentityColumn
Sets /returns the name of column of unique ID

ImportFieldNames
Sets / returns if from column names are taken from first line of CSV file

ImportFileName

Sets / returns the name of CSV file that should be imported

ImportFileNameRelative

Sets / returns if the name of CSV file should be stored relative to the path of project file

IsPhraseIncluded

Sets / returns if a phrase to find address is sufficient

IsSearchField

Sets / returns if a field is used for associative search

LastImportTimeStamp

Returns the timestamp for the last import

MaxCandidates

Sets / returns the maximum number of results of the associative search engine

MinDistance

Sets / returns the required minimum distance to next best candidate for a valid result

MinRelevance

Sets or returns the minimum relevance for search results

MinThreshold

Sets / returns the required minimum value for a valid engine result

ODBCName

Sets / returns name of ODBC source

Password

Sets / returns the password of the ODBC source

Phrase

Sets / returns phrase by its index

PhrasesCount

Returns the number of phrases used for address analysis

PoolName

Sets / returns the name of the associative search pool

PoolPath

Sets / returns the name of path of the associative search pool

PoolPathRelative

Sets / returns if the pool should be saved relative to the path of the project

ProjectPath

Returns the path of the project file

RemovePhrase

Removes a phrase from list of phrases for address analysis specified by its index

SavePoolInternal

Sets / returns if pool should be saved within the project file or as separate files

Separator

Sets / returns separator, either semicolon or comma, used for csv file

SetSearchArea

Sets area on the document to search in

SQLQuery

Sets /returns an SQL statement used to import ODBC source

StartAddEntry

Prepares the insertion of a new entry to the associative search pool

StartUpdate

Generates and opens a new empty pool or opens respectively overwrites an existing pool

Username

Sets / returns username required for login into the ODBC source

VendorTypeColumn

Sets / returns the column that defines the vendor type

The following properties and methods are visible using the type library but must not be used from script. These methods are for integration purpose only and may cause undefined behavior if called from script.

- AutomaticImportMethod
- ClearColumns
- GetEntryByID
- SearchFieldModified

7.1.2 Methods and Properties of SCBCdrAssociativeDbExtractionSettings Interface

AddColumn

```
AddColumn (ColumnName As String, IsSearchField As Boolean, NewColumnIndex As Long)
```

Adds a new column field to the pool. This method returns an error if a pool does not exist or already contains already an entry.

ColumnName

[in] Name of the column field.

IsSearchField

[in] Boolean value that ha to be set to true when the inserted column field is a search field.

NewColumnIndex

[out] Index of the newly created entry in the pool.

AddPhrase

```
AddPhrase (Phrase As String, IsIncludePhrase As Boolean)
```

This method appends a new phrase to the list of phrases to be used for the address analysis.

Phrase

[in] This string variables contains the phrase that is added to the list.

IsIncludePhrase

[in] If the value of the Boolean variable is true and the phrase is found, then the resulting address will be accepted. If the value of the Boolean variable is false and the phrase is found, then the address will not be accepted.

ChangeEntry

```
ChangeEntry (ColumnName As String, EntryData As String)
```

This method updates or inserts the content of the entry data to the specified column.

Use this method only in context with the StartUpdate, StartAddEntry, ChangeEntry, CommitAddEntry and CommitUpdate, to ensure that the pool is consistent after the changes are applied.

ColumnName

[in] Name of the column that is changed.

EntryData

[in] The content of the specified column is updated with this data.

ClassNameFormat

```
ClassNameFormat As String (read/write)
```

This property sets or reads the format definition for a document class name.

For example the expression '[VendorName]' will be replaced by the content of the field VendorName. At least one general field name in the format string must be specified.

ColumnCount

```
ColumnCount As Long (read only)
```

This method returns the number of columns of currently opened pool.

Be reminded that the method "StartUpdate" has to be executed first, to open the pool.

ColumnName

```
ColumnName (ColumnIndex As Long) As String (read/write)
```

This method returns or sets the name of the column by its index.

ColumnIndex

[in] Index of the column to retrieve [zero-based].

CommitAddEntry

```
CommitAddEntry (NewIndex As Long)
```

After execution of StartAddEntry and ChangeEntry changes take effect.

Use this method only in context with the StartUpdate, StartAddEntry, ChangeEntry, CommitAddEntry and CommitUpdate.

NewIndex

[out] Index of new entry.

CommitUpdate

```
CommitUpdate ()
```

This method closes and saves the currently opened pool.

Be reminded to use this method every time when "ChangeEntry" has been applied, to make the pool consistent again.

EnableCandidateEvaluation

```
EnableCandidateEvaluation As Boolean (read/write)
```

Sets / returns if candidate evaluation permitted. If the evaluation is not performed then the confidence values of the search results are taken.

EntryCount

```
EntryCount As Long (read only)
```

Returns the number of entries of the pool.

EvalFirstPageOnly

```
EvalFirstPageOnly As Boolean (read/write)
```

Sets / returns if candidate evaluation is processed only for the first page.

FieldContentsFormat

```
FieldContentsFormat As String (read/write)
```

This method sets / returns the format definition for the representation of the engine results.

FindLocation

```
FindLocation As Boolean (read/write)
```

Sets / returns if address analysis is enabled. True position the address is found.

GeneratePool

```
GeneratePool ()
```

Imports the pool from specified source by the property AutomaticImportMethod.

GeneratePoolFromCsvFile

```
GeneratePoolFromCsvFile ()
```

This method removes the previous pool and generates a new one using CSV file designed in the new format.

The CSV file is a semicolon-separated text list of vendor information (fields). One text line represents one vendor (supplier). The first 3 fields (system fields) are designed for vendor's showing and unifying purposes only that means they are not used for extraction.

GeneratePoolFromODBC

```
GeneratePoolFromODBC ()
```

Removes previous pool and generates a new one using ODBC source using the paramters set on the property page.

GetClassNameByID

```
GetClassNameByID (IDHigh As Long, IDLow As Long, ClassName As String)
```

Returns the formatted document class name for the pool entry specified by its unique ID.

IDHigh

[in] Upper part of 64 bit unique IDs.

IDLow

[in] Lower part of 64 bit unique IDs.

ClassName

[out] Formatted document class name for the specified entry.

GetEntry

```
GetEntry (Index As Long, FieldName As String) As String
```

This method returns the content of a field that is specified by its index and the column name.

Index

[in] Index of the entry to be retrieved.

FieldName

[in] Name of the column to be retrieved.

GetFormattedValueByID

```
GetFormattedValueByID (IDHigh As Long, IDLow As Long, FormattedValue As String)
```

This methods returns the formatted entry representation for the pool entry specified by its unique ID.

IDHigh

[in] Upper part of 64-bit unique ID.

IDLow

[in] Lower part of 64-bit unique ID.

FormattedValue

[out] Formatted entry representation for the specified entry.

GetIDByIndex

```
GetIDByIndex (Index As Long, IDHigh As Long, IDLow As Long)
```

Returns unique ID of an entry by index.

Index

[in] Zero-based index.

IDHigh

[out] Upper part of 64-bit unique ID.

IDLow

[out] Lower part of 64-bit unique ID.

GetIndexByID

```
GetIndexByID (IDHigh As Long, IDLow As Long, Index As Long)
```

Returns index of an entry by its unique ID.

IDHigh

[in] Upper part of 64-bit unique ID.

IDLow

[in] Lower part of 64-bit unique ID.

Index

[out] Zero-based index.

GetSearchArea

```
GetSearchArea (SearchAreaIndex As Long, Left As Long, Top As Long,
         Width As Long, Height As Long)
```

Returns area on the document to search in

SearchAreaIndex

Zero-based index of search area; at the moment two areas are supported.

Left

Distance in % from left border of document.

Top

Distance in % from top of document.

Width

Width in % of search area.

Height

Height in % of search area.

IdentityColumn

```
IdentityColumn As String (read/write)
```

Sets /returns the name of column of unique ID.

ImportFieldNames

```
ImportFieldNames As Boolean (read/write)
```

Sets / returns if from column names are taken from first line of CSV file.

ImportFileName

```
ImportFileName As String (read/write)
```

Sets / returns the name of CSV file that should be imported.

ImportFileNameRelative

```
ImportFileNameRelative As Boolean (read/write)
```

Sets / returns if the name of CSV file should be stored relative to the path of project file.

IsPhraseIncluded

```
IsPhraseIncluded (PhraseIndex As Long) As Boolean (read/write)
```

Sets / returns if a phrase to find address is sufficient.

If the value of the Boolean variable is true and the phrase is found, then the resulting address will be accepted. If the value of the Boolean variable is false and the phrase is found, then the resulting address will not be accepted.

PhraseIndex

[in] Index of phrase [zero-based].

IsSearchField

```
IsSearchField (ColumnIndex As Long) As Boolean (read/write)
```

Sets / returns if a field is used for associative search. If the value is changed, the changes take effect not before a new import of the search pool.

ColumnIndex

[in] Index of column [zero-based].

LastImportTimeStamp

```
LastImportTimeStamp As Date (read only)
```

Returns the timestamp for the last import.

MaxCandidates

```
MaxCandidates As Long (read/write)
```

Sets / returns the maximum number of results of the associative search engine.

MinDistance

```
MinDistance As Double (read/write)
```

Sets / returns the required minimum distance to next best candidate for a valid result.

MinRelevance

```
MinRelevance As Double (read/write)
```

This property sets or returns the minimum relevance for search results, default value is 0.0.

MinThreshold

```
MinThreshold As Double (read/write)
```

Sets / returns the required minimum value for a valid engine result.

ODBCName

```
ODBCName As String (read/write)
```

This property sets / returns the name of the ODBC source.

Password

```
Password As String (read/write)
```

Sets / returns the password of the ODBC source.

Phrase

```
Phrase (PhraseIndex As Long) As String (read/write)
```

Sets / returns phrase by its index.

PhraseIndex

[in] Index of phrase [zero-based].

PhrasesCount

```
PhrasesCount As Long (read only)
```

Returns the number of phrases used for address analysis.

PoolName

```
PoolName As String (read/write)
```

Sets / returns the name of the associative search pool.

PoolPath

```
PoolPath As String (read/write)
```

Sets / returns the name of path of the associative search pool.

PoolPathRelative

```
PoolPathRelative As Boolean (read/write)
```

Sets / returns if the pool should be saved relative to the path of the project.

ProjectPath

```
ProjectPath As String (read only)
```

Returns the path of the project file.

RemovePhrase

```
RemovePhrase (PhraseIndex As Long)
```

Removes a phrase from list of phrases for address analysis specified by its index.

PhraseIndex

[in] Index of the phrase that should be deleted [zero-based].

SavePoolInternal

```
SavePoolInternal As Boolean (read/write)
```

Sets / returns if pool should be saved within the project file or as separate files.

Separator

```
Separator As String (read/write)
```

Sets / returns separator, either semicolon or comma, used for csv file.

SetSearchArea

```
SetSearchArea (SearchAreaIndex As Long, Left As Long, Top As Long,
         Width As Long, Height As Long)
```

Sets area on the document to search in.

SearchAreaIndex

Zero-based index of search area; at the moment two areas are supported.

Left

Distance in % from left border of document.

Top

Distance in % from top of document.

Width

Width in % of search area.

Height

Height in % of search area.

SQLQuery

```
SQLQuery As String (read/write)
```

Sets /returns an SQL statement used to import ODBC source. Within this select statement the name of the queried fields must correspond to columns of the pool.

StartAddEntry

```
StartAddEntry ()
```

Prepares the insertion of a new entry to the associative search pool.

Be reminded to use this method only in context with the StartUpdate, StartAddEntry, ChangeEntry, CommitAddEntry and CommitUpdate.

StartUpdate

```
StartUpdate (RemoveExistingPool As Boolean)
```

This method generates and opens a new empty pool, or opens an existing pool for the update. An existing pool can also be overwritten when the variable RemoveExistingPool is set to true.

RemoveExistingPool

[in] When this Boolean variable is set to true, than the old pool is removed,

otherwise the existing pool is supposed to be updated by further “AddPhrase” calls. Note that in this case, it should not be required to

call “AddColumn” function, because the former column information has to be taken.

Moreover, in case this parameter is true, and then “AddColumn” method is

invoked, the “AddColumn” method will report an error, because it must be prohibited to modify the existing column.

Username

```
Username As String (read/write)
```

Sets / returns username required for the login into the ODBC source.

VendorTypeColumn

```
VendorTypeColumn As String (read/write)
```

Sets / returns the column that defines the vendor type. In self-learning workflow this column will be checked and influences the creation of new document classes. If vendor type is "High Volume Vendor" then a new class is always created automatically, when it is a "Low Volume Vendor" then the class will not be created and in case of a "Normal Vendor", the decision is left to the user.

The content of the returned column, can have the following values:

0: Normal Vendor

1: High Volume Vendor

2: Low Volume Vendor

Chapter 8 Oracle Forms Recognition StringComp Object Reference (SCBCdrSTRCOMPLib)

8.1 SCBCdrStringComp

8.1.1 SCBCdrStringComp Interface (Cedar StringComp Object)

This component provides several implementations of string compare algorithms.

8.1.1.1 Type Definitions

CdrCompareType

String Compare Algorithm

CdrTypeLevenShtein	Levenshtein algorithm
CdrTypeRegularExpression	Regular expression
CdrTypeSimpleExpression	Simple expression
CdrTypeStringComp	Exact string compare
CdrTypeTrigram	Trigram algorithm

8.1.1.2 List of Methods and Properties

CaseSensitive

Sets / returns if compare algorithm should work case sensitive

CompType

Sets / returns the selected compare algorithm

Distance

Perform the selected string compare algorithm

LevDeletions

Returns the count of deletions calculated by the last Distance function

LevInsertions

Returns the count of insertions calculated by the last Distance function

LevRejects

Returns the count of rejects calculated by the last Distance function

LevReplacements

Returns the count of replacements calculated by the last Distance function

LevSame

Returns the count of equal characters calculated by the last Distance function

LevTraceMatrix

Returns the Levenshtein trace matrix calculated by the last Distance function

LevTraceResult

Returns the Levenshtein trace result calculated by the last Distance function

MatchEndPosition

Returns the matching end position calculated by the last Distance function

MatchStartPosition

Returns the matching start position calculated by the last Distance function

SearchExpression

Sets / returns the current search expression

ValidateSearchExpression

Check for syntax errors in the passed search expression

8.1.2 Methods and Properties SCBCdrStringComp Interface

CaseSensitive

```
CaseSensitive As Boolean (read/write)
```

This option controls if the compare algorithm should work case sensitive.

CompType

```
CompType As CdrCompareType (read/write)
```

This option selects the compare algorithm used for the next call of Distance.

Distance

```
Distance (String As String, Distance As Double)
```

Perform the selected string compare algorithm. The search expression and the compare method must be initialized before. The return value is the distance between the search expression and the string parameter, which is between 0.0 and 1.0. A distance of 0.0 means that the search expression matches the string parameter exactly and a distance of 1.0 means that there is no match at all. Most algorithms can also return a value between 0.0 and 1.0 which provides the possibility to compare strings in a fault tolerant way.

String

[in] Specifies the string which should be compared with the search expression.

Distance

[out] Contains the distance of the compare operation, which will be between 0.0 and 1.0.

LevDeletions

```
LevDeletions As Single (read only)
```

Returns the count of deletions calculated by the last Distance function. This is only valid if the Levenshtein algorithm was selected during the last Distance call.

LevInsertions

```
LevInsertions As Single (read only)
```

Returns the count of insertions calculated by the last Distance function. This is only valid if the Levenshtein algorithm was selected during the last Distance call.

LevRejects

```
LevRejects As Single (read only)
```

Returns the count of rejects calculated by the last Distance function. This is only valid if the Levenshtein algorithm was selected during the last Distance call.

LevReplacements

```
LevReplacements As Single (read only)
```

Returns the count of replacements calculated by the last Distance function. This is only valid if the Levenshtein algorithm was selected during the last Distance call.

LevSame

```
LevSame As Single (read only)
```

Returns the count of equal characters calculated by the last Distance function. This is only valid if the Levenshtein algorithm was selected during the last Distance call.

LevTraceMatrix

```
LevTraceMatrix As String (read only)
```

Returns the Levenshtein trace matrix calculated by the last Distance function. This is only valid if the Levenshtein algorithm was selected during the last Distance call.

LevTraceResult

```
LevTraceResult As String (read only)
```

Returns the Levenshtein trace result calculated by the last Distance function. This is only valid if the Levenshtein algorithm was selected during the last Distance call.

MatchEndPosition

```
MatchEndPosition As Single (read only)
```

Returns the matching end position calculated by the last Distance function. This is only valid if the simple or regular expression algorithm was selected during the last Distance call.

MatchStartPosition

```
MatchStartPosition As Single (read only)
```

Returns the matching start position calculated by the last Distance function. This is only valid if the simple or regular expression algorithm was selected during the last Distance call.

SearchExpression

```
SearchExpression As String (read/write)
```

This option contains the search expression which should be used for the next compare operation. In case of simple or regular expression, the string containing that expression must be set to this property not to the Distance function.

ValidateSearchExpression

```
ValidateSearchExpression (Type As CdrCompareType, SearchExpression As String)
         As Boolean
```

Perform a syntax check for the specified compare method and search expression. In case of simple or regular expression the search expression can contain a syntax error. In this case the return value will be FALSE. In all other cases the return value will be TRUE.

Type

Compare method which should be used for validation.

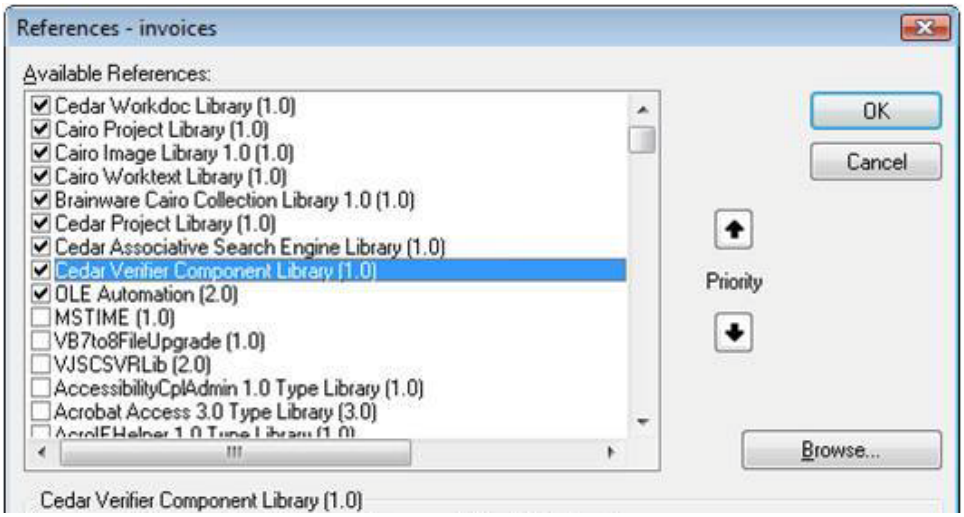
SearchExpression

Search expression which should be validated.

Chapter 9 Oracle Forms Recognition Verifier Object Reference (DISTILLERVERIFIERCOMPLib)

9.1 Adding Type Library References

To implement the custom script code for changing default colors and fonts for desired verification forms, first open the project file in Oracle Forms Recognition Designer application, select required class to modify the verification form (or choose the project level script sheet – see below for more details on this topic) and switch to Script Editor view to insert the type library references.



Select “OLE Automation (2.0)” and “Cedar Verifier Component Library (1.0)” as references from the list of available references in “File → References…” dialog box of the WinWrap / SAX Basic script editor:

Note: You can also do it without this dialog, i.e., without a need to search in this dialog for the required reference. Just open your project in Designer application and insert the following two entries at the very top of the required script page:

```
'#Reference {00020430-0000-0000-C000-000000000046}#2.0#0#C:\Windows\system32\stdole2.tlb#OLE Automation
'#Reference {4C58EA47-848D-11D4-805C-00902734E7E3}#1.0#0# C:\Program Files\Oracle\Forms
Recognition\Components\Cedar\CdrVer.dll#Cedar Verifier Component Library
```

Then save the project file ignoring possible script errors, close the Designer application and re-open the project again. The desired references should be available for the corresponding script sheet now.

9.2 SCBCdrVerificationFormComp

9.2.1 SCBCdrVerificationForm Interface (Cedar VerificationForm Object)

This interface is used to set properties specific for verification form object, as well as to set default properties for embedded elements, like verification fields, labels, tables, buttons, and so on.

9.2.1.1 List of Methods and Properties

DefaultLabelFont

Sets / returns default font for all label elements available on this verification form.

DefaultLabelFontColor

Sets / returns default color for all label elements available on this verification form.

DefaultLabelBackgroundColor

Sets / returns default background color for all label elements available on this verification form.

DefaultFieldFont

Sets / returns default font for all verification field elements available on this verification form.

DefaultFieldFontColor

Sets / returns default color for all verification field elements available on this verification form.

DefaultElementBackgroundColorValid

Sets / returns default color for all valid (valid in terms of validation status) field elements available on this verification form.

DefaultElementBackgroundColorInvalid

Sets / returns default color for all invalid (invalid in terms of validation status) field elements available on this verification form.

FormBackgroundColor

Sets / returns background color for the form.

FormBackgroundColorDI

Sets / returns background color for the Direct Input control on the form.

9.2.2 Methods and Properties SCBCdrVerificationForm Interface

DefaultLabelFont

```
DefaultLabelFont As SdtFont
```

Sets / returns default font for all label elements available on this verification form. The system will always use this font setting for all label elements where individual font is not assigned. If the individual font was assigned but then has to be reset to the form-default one, it can always be done via assigning of “Nothing” value.

For example:

```
theForm.VerificationTables.ItemByName(“MyTable”) Font = Nothing
```

This comment is applicable for all methods below related to assignment of font property.

DefaultLabelFontColor

```
DefaultLabelFontColor As OLE_COLOR
```

Sets / returns default color for all label elements available on this verification form. The system will always use this color setting for all label elements where individual color is not assigned. If the individual color was assigned but then has to be reset to the form-default one, it can always be done via assigning of special value, like in the example below:

```
Dim clrDefaultColor As OLE_COLOR
clrDefaultColor = -1
theForm.VerificationLabels.ItemByIndex(lNextLabelIndex).FontColor = clrDefaultColor
```

This comment is applicable for all methods below related to assignment of font property.

DefaultLabelBackgroundColor

```
DefaultLabelBackgroundColor As OLE_COLOR
```

Sets / returns default background color for all label elements available on this verification form.

DefaultFieldFont

```
DefaultFieldFont  As StdFont
```

Sets / returns default font for all verification field elements available on this verification form.

This setting is applicable for the following field types:
- Normal verification field (so-called header field).
- Verification table
- Drop down list field
- Check box field.

DefaultFieldFontColor

```
DefaultFieldFontColor  As OLE_COLOR
```

Sets / returns default color for all verification field elements available on this verification form.

This setting is applicable for the following field types:
- Normal verification field (so-called header field).
- Verification table
- Drop down list field
- Check box field

DefaultElementBackgroundColorValid

```
DefaultElementBackgroundColorValid As OLE_COLOR
```

Sets / returns default color for all valid (valid in terms of validation status) field elements available on this verification form.

This setting is applicable for the following field types:
- Normal verification field (so-called header field).
- Verification table
- Drop down list field
- Check box field

DefaultElementBackgroundColorInvalid

```
DefaultElementBackgroundColorInvalid As OLE_COLOR
```

Sets / returns default color for all invalid (invalid in terms of validation status) field elements available on this verification form.

This settings is applicable for the following field types:
- Normal verification field (so-called header field).
- Verification table
- Drop down list field
- Check box field

FormBackgroundColor

```
FormBackgroundColor As OLE_COLOR
```

Sets / returns background color for the form.

FormBackgroundColorDI

```
FormBackgroundColorDI As OLE_COLOR
```

Sets / returns background color for the Direct Input control on the form, i.e. for the area around the Direct Input field (Direct Input control is an element that zooms currently validated field).

9.3 SCBCdrVerificationField Interface

9.3.1 SCBCdrVerificationField Interface (Cedar VerificationField Object)

This interface is used to identify verification properties specific for header fields' validation elements, like drop down lists, check-boxes, and normal edit fields.

9.3.1.1 List of Methods and Properties

Font

Sets / returns font settings for the individual verification field element.

FontColor

Sets / returns font color for the individual verification field element.

BackgroundColorValid

Sets / returns background color for the individual verification field element, when the element is valid in terms of current validation status.

BackgroundColorInvalid

Sets / returns background color for the individual verification field element, when the element is invalid in terms of current validation status.

9.3.2 Methods and Properties SCBCdrVerificationField Interface

Font

```
Font As StdFont
```

Sets / returns font settings for the individual verification field element.

FontColor

```
FontColor As OLE_COLOR
```

Sets / returns font color for the individual verification field element.

BackgroundColorValid

```
BackgroundColorValid As OLE_COLOR
```

Sets / returns background color for the individual verification field element, when the element is valid in terms of current validation status.

BackgroundColorInvalid

```
BackgroundColorInvalid As OLE_COLOR
```

Sets / returns background color for the individual verification field element, when the element is invalid in terms of current validation status.

9.4 SCBCdrVerificationTable Interface

9.4.1 SCBCdrVerificationTable Interface (Cedar VerificationTable Object)

This interface is used to identify verification properties specific for table validation elements.

9.4.1.1 List of Methods and Properties

Font

Sets / returns font settings for the individual table field element.

FontColor

Sets / returns font color for the individual table field element.

BackgroundColorValid

Sets / returns background color for the individual verification table element, when the table cell is valid in terms of current validation status.

BackgroundColorInvalid

Sets / returns background color for the individual verification table element, when the table cell is invalid in terms of current validation status.

HeaderFont

Sets / returns font settings for all header buttons of the table field element.

HeaderFontColor

Sets / returns font color for the header buttons of the table field element.

HeaderBackgroundColor

Sets / returns background color for all header buttons of the table field element.

9.4.2 Methods and Properties SCBCdrVerificationTable Interface

FontFont

```
FontFont As StdFont
```

Sets / returns font settings for the individual table field element. The settings affect all cells of the table.

Note that for the currently selected element the font type is automatically set to Bold. Therefore if the present font setting for all elements is set to Bold, then the currently selected table cell's font type is going to be set automatically to the opposite style, i.e., to Normal font's weight.

FontColor

```
FontColor As OLE_COLOR
```

Sets / returns font color for the individual table field element. The settings affect all cells of the table.

BackgroundColorValid

```
BackgroundColorValid As OLE_COLOR
```

Sets / returns background color for the individual verification table element, when the table cell is valid in terms of current validation status.

BackgroundColorInvalid

```
BackgroundColorInvalid As OLE_COLOR
```

Sets / returns background color for the individual verification table element, when the table cell is invalid in terms of current validation status.

HeaderFont

```
HeaderFont As StdFont
```

Sets / returns font settings for all header buttons of the table field element, including row header buttons, column header buttons and the table header button (small control in the left-top corner of the table).

HeaderFontColor

```
HeaderFontColor As OLE_COLOR
```

Sets / returns font color for the header buttons of the table field element, including row header buttons and column header buttons.

HeaderBackgroundColor

```
HeaderBackgroundColor As OLE_COLOR
```

Sets / returns background color for all header buttons of the table field element, including row header buttons, column header buttons, and the table header button.

For the row header buttons, the background color is going to be adjusted automatically to lighter / darker one for each next pair of rows in order to provide with a better visual cue to distinguish between neighboring rows.

9.5 SCBCdrVerificationButton Interface

This interface is used to set verification properties specific for all custom buttons defined on a verification form.

9.5.1 SCBCdrVerificationButton Interface (Cedar VerificationButton Object)

Font

Sets / returns font settings (name, type, and styles) for the individual custom button control.

FontColor

Sets / returns font color for the individual custom button control.

BackgroundColor

Sets / returns background color for the individual custom button control.

9.5.2 Methods and Properties SCBCdrVerificationButton Interface

Font

```
Font As StdFont
```

Sets / returns font settings (name, type, and styles) for the individual custom button control.

FontColor

```
FontColor As OLE_COLOR
```

Sets / returns font color for the individual custom button control.

BackgroundColor

```
BackgroundColor As OLE_COLOR
```

Sets / returns background color for the individual custom button control.

9.6 SCBCdrVerificationLabel Interface

9.6.1 SCBCdrVerificationLabel Interface (Cedar VerificationLabel Object)

This interface is used to set verification properties specific for all custom label elements defined on a verification form.

9.6.1.1 List of Methods and Properties

Font

Sets / returns font settings (name, type, and styles) for the individual custom label control.

FontColor

Sets / returns font color for the individual custom label control.

BackgroundColor

Sets / returns background color for the individual custom label control.

9.6.2 Methods and Properties SCBCdrVerificationLabel Interface

Font

```
Font As StdFont
```

Sets / returns font settings (name, type, and styles) for the individual custom label control.

FontColor

```
FontColor As OLE_COLOR
```

Sets / returns font color for the individual custom label control.

BackgroundColor

```
BackgroundColor As OLE_COLOR
```

Sets / returns background color for the individual custom label control.

9.7 Dynamic & Static Modification of Verification Forms' Properties

As soon as the object properties have been correctly adjusted in script, this will affect all Oracle Forms Recognition applications that may use the modified verification form(s) for representation of documents. This includes: Verifier Test and Verifier Train modes of Oracle Forms Recognition Designer application, document validation mode of Oracle Forms Recognition Verifier application, and Verifier Correction mode of IOracle Forms Recognition Learnset Manager tool.

The entire set of the new script methods can be used to either adjust the properties of verification elements dynamically or statically.

The dynamic modification can be implemented either in a "Script_Initialize" event (at project level script page) or for each individual document in "FocusChanged" event using "CdrBeforeFormLoaded" "Reason" conditional statement (at particular class script sheet).

The static modification can be applied via launching the corresponding script code once in Designer application (placed in any invoked script event) and then clicking on the "Save" project button. The saved project stores the modified properties of the verification forms and their elements and, when opened in Oracle Forms Recognition Verifier or another application, provides with the new look of the verification forms.

The dynamic way of forms properties' modification can be useful for, e.g., dynamic translation to different languages according to current system locale settings.

Note: Use integrated "GetLocale" method to query current workstation's regional settings:

```
Dim lOriginalLocale As Long
lOriginalLocale = GetLocale
```

Note that "GetLocale" method is unavailable in SAX Basic scripting engine and can only be used with the new WinWrap Basic engine.

Note that dynamic way of modification is currently inapplicable for Oracle Forms Recognition Learnset Manager tool.

9.7.1 Notes & Tricks

Note A

All properties listed above are supposed to have immediate effect on the look of the updated elements. In other words, there is actually no need to place all modifications of these properties in the events called prior to loading the verification form. Compared with the other previously available properties of the verification objects all the new ones are truly dynamic, it can be used for immediate adjustment of the GUI in any event fired during validation of one particular document, i.e., in a handler of "OnAction" event in order to change some UI properties when a user clicks on a custom button.

Note B

When there are more than one different font used, the script has to create a separate instance for each individual font:

```
Dim theFont1 As New StdFont
Dim theFont2 As New StdFont
```

Otherwise, if the same font object's instance is reused to just assign different properties when defining the font for the second group of controls, this will lead to automatic changing of font properties of the first controls' group.

Note C

The standard visualization of check-box fields, there is no color differentiation between valid and invalid check-box field's status. Now this can be changed using SCBCdrVerificationField::BackgroundColorValid and SCBCdrVerificationField::BackgroundColorInvalid correspondingly.

9.7.2 Usage

This set of new methods can be used for:

- Personalization of verification forms, providing Verifier users with custom, better looking GUI.
- Adjustment of the way valid and invalid field status is presented if standard color-differentiators are, for any, reason insufficient.
- Identification of critical elements (e.g., in terms of importance of the validation to apply for a specific item) on the form, highlighting it with a different color or font style.
- Setting greater font sizes / bold styles for people with partial loss of sight, when required.

Glossary

Batch A logical organizational structure to control a set of documents during the process. A batch is normally created during the scan process from a batch of paper. The status of a batch is used to manage the input flow.

Document On the one hand, a paper document consisting of one or more pages that are logically connected. On the other hand, document is the general name for the central entity the system processes. A document is classified, a document has fields that are extracted and corrected, and a document can have one or more images attached.

Engine A software component as DLL for one specific step in document processing. Engines provide methods for OCR, classification and extraction. Engines can be individually added and are only available when installed. Each engine carries it's own property page for configuration.

Folder A logical structure inside a batch for coherent documents. A folder may, for example, consist of all pages of the correspondence with many folders inside one batch.

Image A digital raster image normally created during the scan process. The image is compressed and stored in a specific format. Internally, *Oracle Forms Recognition* uses a structure called Image that represents the raster image. The Image possesses methods to load, store, and manipulate it. The image can be displayed in the viewer.

Status A state variable in the input flow that is kept for each document, folder, and batch. With the status the next step in the input flow can be determined. The status of folder is the calculated from the status of the documents, the status of the batch is calculated from the status of the folders.

Valid A state variable on the level of fields and documents. Depending on validation rules a field is correct and is set valid. If all fields are valid the document is valid.

Workdoc An internal structure representing the logical structure of a document. The Workdoc represents the runtime data of one single document and is stored in a file with the extension *.wdc. Since the Workdoc includes all OCR and analysis results, it may exceed the image files by size.

Worktext An internal structure representing the OCR result of the recognition process of a document. The Worktext contains the characters with their positions, bounding rectangles, font properties, and confidences.

Index