Perceptive Intelligent Capture

with Supervised Learning

Scripting Reference Guide

Version 5.5 SP1

November 2012

Prepared by:

Perceptive Engineering

perceptivesoftware

Document Number

Pcv-Scr-5.5SP1

Version 5.5 SP1

November 2012

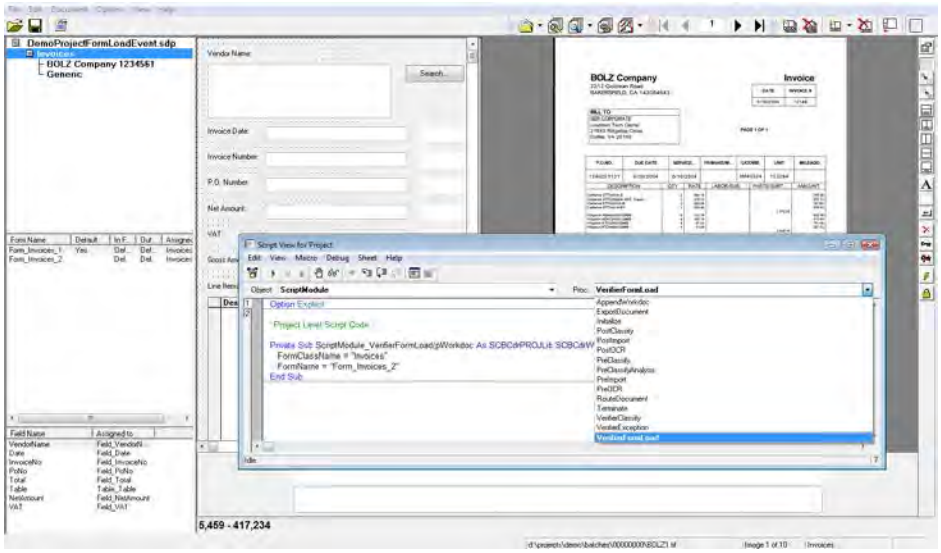
Perceptive Software, Inc.

Contents

Chapter 1 Script Event Reference

1.1 Description - VerifierFormLoadEvent

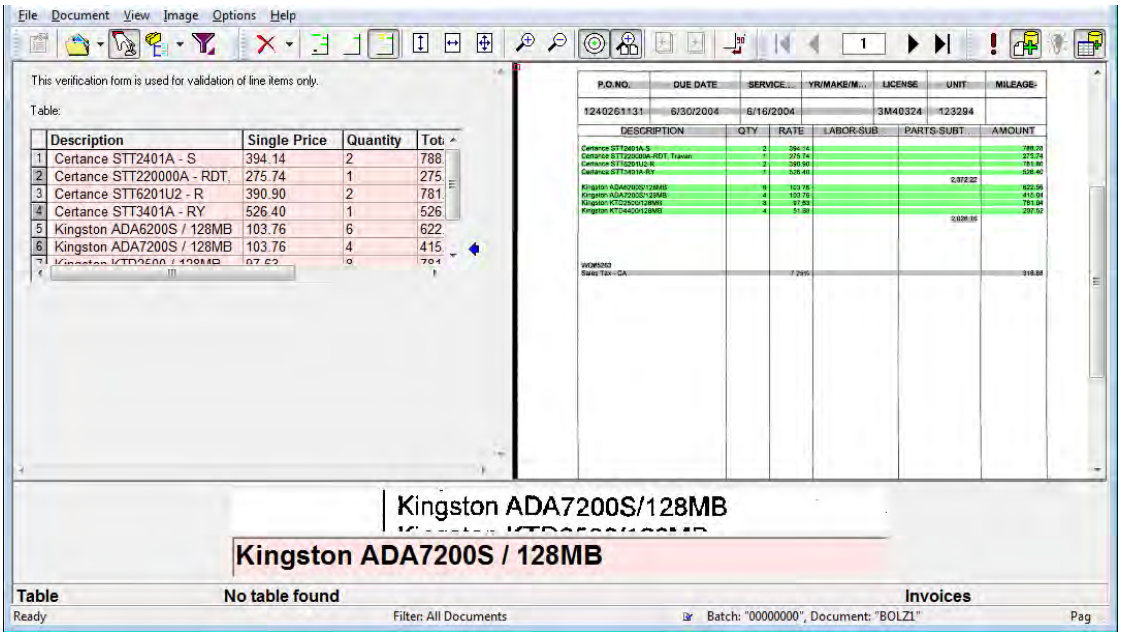
In order to implement the script handler of this event, start the Perceptive Intelligent Capture Designer application, load the desired project file, select the project node in Definition mode, open the Script Editor, select the “Script Module” object and click on the new “VerifierFormLoad” item in “Proc”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

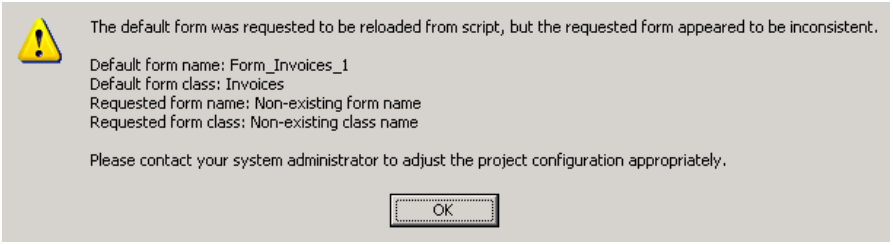


In case the script modifies the form and form's class references incorrectly (for example, referring to a non-existing verification form of a class, or in case the form does not exist in the specified class, and so on), a warning message is displayed to the Verifier user.

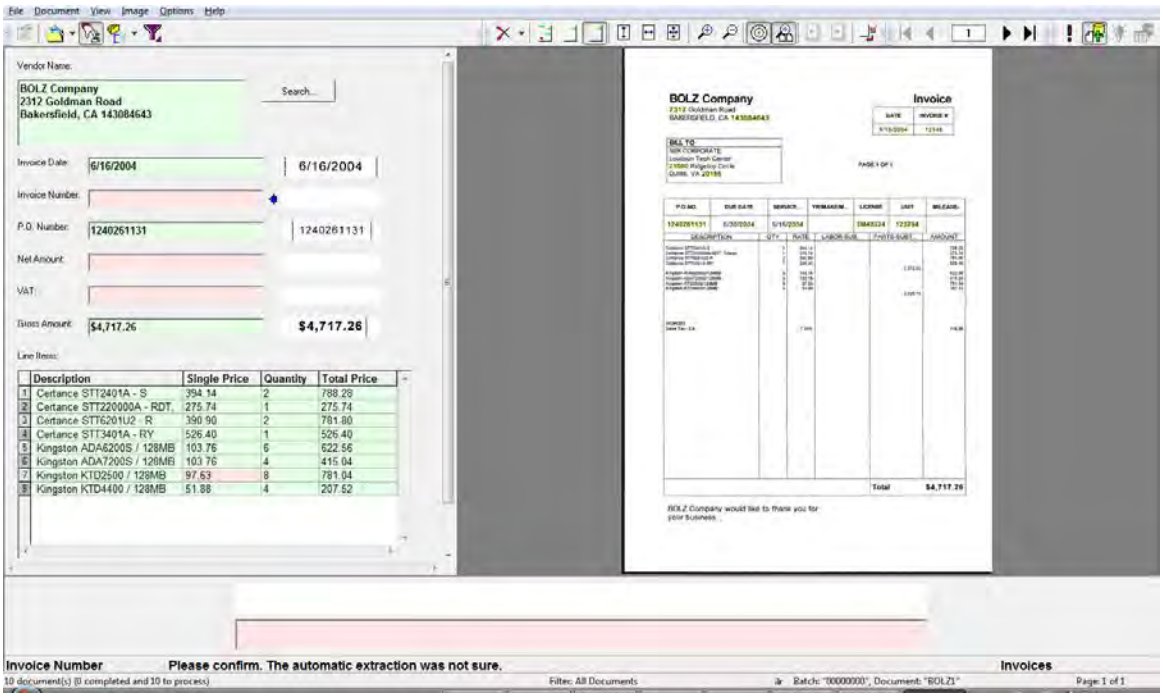
For example, in case of the wrong script like this:

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

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

End Sub
```

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



Then the application loads 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 by the custom script:



Note: the new event is fired from within the Perceptive Intelligent Capture Verifier application only and cannot be tested in Perceptive Intelligent Capture Designer application.

As another relevant extension, the former document class level “FocusChanged” 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 Perceptive Intelligent Capture 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
```

1.1.1. 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.
- To display a specific page of a document instead of the first one.

1.2 ScriptModule

Cedar ScriptModule Event Interface

Project events are specific for one Perceptive Intelligent Capture Project, but within a Perceptive Intelligent Capture 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.

1.2.1. Methods and Properties

1.2.1.1. AppendWorkdoc

AppendWorkdoc

Description Appends a given Workdoc after last Workdoc on the base of CdrMPType.

Syntax

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

Parameters

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.

1.2.1.2. BatchClose

BatchClose

Description This event is launched when the verifier user exits a batch in one of the following methods:

- When verifying a batch and selecting, Return to batch list
- Batch Verification Completion
- Partial Batch verification completion
- The user quits the verifier applications whilst in a batch.

The event is triggered in the Verifier Thick Client and the Web Verifier applications.

Syntax

```
ScriptModule_BatchClose(ByVal UserName As String, ByVal
BatchDatabaseID As Long, ByVal ExternalGroupID As Long,
ByVal ExternalBatchID As String, ByVal TransactionID As
Long, ByVal WorkflowType As
SCBCdrPROJLib.CDRDatabaseWorkflowTypes, BatchState As
Long, BatchReleaseAction As
SCBCdrPROJLib.CDRBatchReleaseAction)
```

Parameters

UserName:	The Username currently logged in who has

closed the batch.

BatchDatabaseID: The unique Batch ID within the database. For the File System, this batch ID is not used.

ExternalGroupID: The Group ID which can be assigned to a batch.

The Group ID can be used with the new Scripting security methods which enable the developer to assign a batch a security group. Only those users belonging to the same Group ID will be able to access batches.

For example, a batch belonging to Group ID 80 will only be accessible by a user who is assigned to group 80.

Read or Write Parameter which can be modified to any long value.

ExternalBatchID: The External Batch ID can be assigned to a batch.

The External Batch ID allows the developer to synchronize a newly created batch of documents with another external system. For example, and archiver, a storage box ID, etc.

Read or Write Parameter which can be modified to any long value.

TransactionID: The Transaction ID can be assigned to a batch.

The Transaction ID allows the developer to synchronize a newly created batch of documents with another external system. For example, and archiver, a storage box ID, etc.

Read or Write Parameter which can be modified to any long value.

WorkflowType: Corresponds to CDRDatabaseWorkflowTypes data type.

BatchState: The current status of the batch being opened, eg status 550 (Extraction Verification).

BatchReleaseAction: Batch Release Action represents the action taken when the last document of the batch has been verified. The parameter can be set, or read from script. By default, it is always set to CDRBatchReleaseActionUserDefined (as user always makes a selection). If registry

value is used to hide batch release dialog box in Verifier thick client, then the last action taken prior to dialog being hidden, will be the one showing in this parameter.

The scripter can set an override value to this parameter, eg, every time batch verification completes, always goes to next invalid batch.

Cancel – means returns to current batch and last document verified

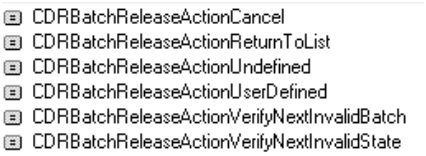
Return To List – return to batch list

Undefined – unknown

Action User Defined – default, user makes a selection on next action to take on batch release

VerifyNextInvalidBatch – open next batch to verify

VerifyNextInvalidState – open current batch to verify in the next invalid state



See Also BatchOpen, Project Event, PostImportBatch, CDRDatabaseWorkflowTypes

Example

```
Example

Private Sub ScriptModule_BatchClose(ByVal UserName As String, ByVal
BatchDatabaseID As Long, ByVal ExternalGroupID As Long, ByVal
ExternalBatchID As String, ByVal TransactionID As Long, ByVal
WorkflowType As SCBCdrPROJLib.CDRDatabaseWorkflowTypes, BatchState As
Long, BatchReleaseAction As SCBCdrPROJLib.CDRBatchReleaseAction)

Call LogMessage(BatchDatabaseID & "," & UserName, "C:\EventTrace_" &
Format(Now,"DDMMYYYY") & ".Log")

End Sub
```

1.2.1.3. BatchOpen

BatchOpen

Description An event that is triggered when the user opens a batch.

Syntax

```
ScriptModule_BatchOpen(ByVal UserName As String, ByVal
BatchDatabaseID As Long, ByVal ExternalGroupID As Long,
ByVal ExternalBatchID As String, ByVal TransactionID As
Long, ByVal WorkflowType As
SCBCdrPROJLib.CDRDatabaseWorkflowTypes, BatchState As
```

Long)

Parameters

UserName:	The Username currently logged in who has opened the batch.
BatchDatabaseID:	The unique Batch ID within the database. For the File System, this batch ID is not used. The Batch ID will be in the form of a numeric value, eg for Batch 00000061, the value 61 will be returned.
ExternalGroupID:	The Group ID which can be assigned to a batch. The Group ID can be used with the new Scripting security methods which enable the developer to assign a batch a security group. Only those users belonging to the same Group ID will be able to access batches. For example, a batch belonging to Group ID 80 will only be accessible by a user who is assigned to group 80. Read or Write Parameter which can be modified to any longvalue.
ExternalBatchID:	The External Batch ID can be assigned to a batch. The External Batch ID allows the developer to synchronize a newly created batch of documents with another external system. For example, an archive ID, a storage box ID, etc. Read or Write Parameter which can be modified to any long value.
TransactionID:	The Transaction ID can be assigned to a batch. The Transaction ID allows the developer to synchronize a newly created batch of documents with another external system. For example, an archive ID, a storage box ID etc. Read or Write Parameter which can be modified to any longvalue.
WorkflowType:	Corresponds to CDRDatabaseWorkflowTypes data type.
BatchState:	The current status of the batch being opened, eg status 550 (Extraction Verification).

See Also BatchClose, Project Event, PostImportBatch, CDRDatabaseWorkflowTypes

Example Example below logs the Batch ID and User name that Opened a batch, with date/time.

LogMessage is a custom function writes a text line into a log file with Date/Time as a prefix.

```
Private Sub ScriptModule_BatchOpen(ByVal UserName As String, ByVal
BatchDatabaseID As Long, ByVal ExternalGroupID As Long, ByVal
ExternalBatchID As String, ByVal TransactionID As Long, ByVal
WorkflowType As SCBCdrPROJLib.CDRDatabaseWorkflowTypes, BatchState As
Long)

   Call LogMessage(BatchDatabaseID & "," & UserName, "C:\EventTrace_"
& Format(Now,"DDMMYYYY") & ".Log")

End Sub
```

1.2.1.4. ExportDocument

ExportDocument

Description Provides the ability to implement a customer specific export of all extracted data.

Syntax

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

Parameters

pWorkdoc:	Workdoc, which should be exported
ExportPath:	Export path, which was configured within the Runtime Server settings (no changes possible)
pCancel:	Set this variable to TRUE to cancel the export

Example

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

1.2.1.5. ForceClassificationReview

ForceClassificationReview

Description In the application, the PostClassify event has been extended so that it can force a manual classification review even if the classification succeeded.

Attribute Read/Write

See also PostClassify

Example The script sample below shows how the manual classification process can be forced from custom script event “PostClassify”.

```
Private Sub ScriptModule_PostClassify(pWorkdoc As
```

```
SCBCdrPROJLib.SCBCdrWorkdoc)

   If pWorkdoc.DocClassName = "VeryImportantClass" Then
pWorkdoc.ForceClassificationReview = True   End If

End Sub
```

1.2.1.6. Initialize

Initialize

Description The Initialize event is called when a batch is opened for processing.

Syntax

```
ScriptModule_Initialize (ModuleName As String)
```

Parameters

ModuleName: Name of the current module, allowed values: "Server", "Designer", "Verifier", Thin Client Verifier

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
```

1.2.1.7. MoveDocument

MoveDocument

Description

This event is launched when the Verifier / Web Verifier User places a document in Exception, and the document is moved out of the batch.

The ScriptModule logs following event information: Old Batch ID, New Batch ID, Reason, Document state. For the event to be triggered, the condition must be set within the application settings that a new exception batch is created when a user places a document to exception.

The event will be triggered for each document that is placed into exception within a single batch.

After placing a document to Exception, the event will be triggered if:

- Batch Verification is completed and all other documents have been verified or placed in exception.
- The user returns to the batch list after placing the document into Exception.

Syntax

```
ScriptModule_MoveDocument(
pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc,
ByVal OldBatchID As String,
ByVal NewBatchID As String,
ByVal Reason As SCBCdrPROJLib.CDRMoveDocumentReason)
```

Parameters	*pWorkdoc*:	The Workdoc Object that is being used. No changes can be made to the workdoc within this event.
	OldBatchID:	The batch ID to which the document belonged prior to placing a document to exception.
	NewBatchID:	The new batch ID to which the document is moving after the document is placed in Exception.
	Reason:	The reason the event is triggered. Thus far, the only reason implemented is for the document moved to exception – this is a value of 1.
	DocState:	The workflow state of the document
See Also	Project Event	

Example

The example below logs a general message for each document placed into exception, showing the old batch ID and the new batch ID.

```
Private Sub ScriptModule_MoveDocument(pWorkdoc As
            SCBCdrPROJLib.SCBCdrWorkdoc, ByVal OldBatchID As
            String, ByVal NewBatchID As String, ByVal Reason As
            SCBCdrPROJLib.CDRMoveDocumentReason)

   If Reason = CDRMoveDocumentToExceptionBatch Then

      Project.LogScriptMessageEx CDRTypeInfo,
CDRSeveritySystemMonitoring, "  Document [" & pWorkdoc.Filename &
"] has been moved from Verifier batch [" & OldBatchID & "] to
exception batch [" & NewBatchID & "]"

      Project.LogScriptMessageEx CDRTypeInfo,
CDRSeveritySystemMonitoring, "  Current document state is [" &
CStr(pWorkdoc.CurrentBatchState) & "]"

   End If

End Sub
```

1.2.1.8. PostClassify

PostClassify

Description	The PostClassify event will be called after all defined classification methods have been executed by the Cedar Project.	
Syntax	`ScriptModule_PostClassify (pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)`	
Parameters	*pWorkdoc:*	Workdoc object which has been classified
See also	ForceClassificationReview	

Example

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

Dim imgDocument As SCBCroImage

Dim lngTagCount As Long

'Imprint number is stored as a Tifftag in the image file - the
following code extracts the Tifftag

'information and sets the field value.

'NOTE: this will only work if there is a single Tifftag - would
require modification for more!

Set imgDocument = pWorkdoc.Image(0)

lngTagCount = imgDocument.TiffTagCount

'Check that there is at least 1 tifftag.

If (lngTagCount > 0) Then

Dim intImageCount As Integer

Dim intImageCounter As Integer

intImageCount=pWorkdoc.PageCount'Get the number of pages in TIF


Dim imgCollection() As SCBCroImage

ReDim imgCollection(intImageCount)   'Set an image collection
variable to store all the pages of the image


'Store all pages of TIF image onto a temporary image collection array

For intImageCounter=0 To intImageCount-1

Set imgCollection(intImageCounter)=pWorkdoc.Image(intImageCounter)

Next

Dim strTag As String

strTag = CStr(Format(Now(), "yyyymmddhhMMss")) & "123456"    'Set the
Info to place into TIF Tag

imgCollection(0).TiffTagClearAll     'Clear All TIF Tags

imgCollection(0).TiffTagAddASCII 33601, strTag   'Add the TIF Tag

imgCollection(0).SaveFile(pWorkdoc.DocFileName(0)) 'Save modified
image collection with TIF Tag and overwrite existing image


'Reset the collection to the new image in workdoc

For intImageCounter=1 To intImageCount-1

imgCollection(intImageCounter).AppendToMultiImageFile(pWorkdoc.DocFil
eName(0))

Next


MsgBox("Tag = " & imgDocument.TiffTagString(lngTagCount))  'Message
box to show TIF Tag

Else

' If there is no Tifftag, can set the field to false - no Tifftag
means that something

'  has gone wrong with scanning. Generate a new Doc ID.

MsgBox("No Tag")

End If
```

```
End Sub
```

Example:

```
Private Sub ScriptModule_PostClassify(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc)
Dim imgDocument As SCBCroImage
Dim lngTagID as long
lngTagID = 12345

Set imgDocument = pWorkdoc.Image(0)

Call fnCreateTiffTag(imgDocument, kngTagID, “Test”)

End Sub
```

1.2.1.9. PostImportBatch

PostImportBatch

Description

An event that is triggered when the Runtime Server is configured to run with security update.

Only one Runtime Server instance should be configured to update system security. The frequency of the security update is determined via the Runtime Server instance properties.

Syntax

```
ScriptModule_PostImportBatch( ByVal BatchDatabaseID As
Long, BatchName As String, Priority As Long, State As
Long, ExternalGroupID As Long, ExternalBatchID As
String, TransactionID As Long, TransactionType As Long)
```

Parameters

BatchDatabaseID: The unique Batch ID from the database. This would be a numeric ID corresponding to the BatchID within the database tables. Read Only Parameter which cannot be modified.

BatchName: The Batch Name which is assigned by the Runtime Server instance. The name is taken from the Import settings of the Runtime Server instance.

Read or Write Parameter which can be modified to any string value.

Priority: The Batch priority which is assigned by the Runtime Server instance. The priority is taken from the Import settings of the Runtime Server instance.

Read or Write Parameter which can be modified to any long value between 1 to 9.

State: The Batch State which is assigned by the Runtime Server instance. The status is taken from the Workflow settings of the Runtime Server instance – Import Success.

Read or Write Parameter which can be

modified to any long value between 100 and 999.

ExternalGroupID: The Group ID which can be assigned to a batch.

The Group ID can be used with the new Scripting security methods which enable the developer to assign a batch a security group. Only those users belonging to the same Group ID will be able to access batches.

For example, a batch belonging to Group ID 80 will only be accessible by a user who is assigned to group 80.

Read or Write Parameter which can be modified to any numeric value.

ExternalBatchID: The External Batch ID can be assigned to a batch.

The External Batch ID allows the developer to synchronize a newly created batch of documents with another external system. For example, and archiver, a storage box ID etc.

Read or Write Parameter which can be modified to any numeric value.

TransactionID: The Transaction ID can be assigned to a batch.

The Transaction ID allows the developer to synchronize a newly created batch of documents with another external system. For example, an archive ID, a storage box ID etc.

Read or Write Parameter which can be modified to any long value.

TransactionType: The Transaction Type can be assigned to a batch.

The Transaction Type allows the developer to synchronize a newly created batch of documents with another external system. For example, an archiveID, a storage box ID etc.

Read or Write Parameter which can be modified to any long value.

See Also ScriptModule Events, SecurityUpdateStart, SecurityUpdateAddUserGroup, SecurityUpdateCommit

Example The example below updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the Perceptive Intelligent Capture user table.

```
Private Sub ScriptModule_PostImportBatch(ByVal BatchDatabaseID As
Long, BatchName As String, Priority As Long, State As Long,
ExternalGroupID As Long, ExternalBatchID As String, TransactionID As
```

```
Long, TransactionType As Long)

   'Set batch priorities after import
   BatchName = "AP Batch_" & CStr(BatchDatabaseID)
   Priority = 2
   State = 102
   ExternalGroupID = 777
   TransactionType = 10
   TransactionID = 2


End Sub
```

1.2.1.10. PreClassify

PreClassify

Description 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.

Syntax `ScriptModule_PreClassify (pWorkdoc As SCBCdrWorkdoc)`

Parameters *pWorkdoc:* Workdoc object, which should be classified

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
```

1.2.1.11. PreClassifyAnalysis

PreClassifyAnalysis

Description The *PreClassifyAnalysis* event is fired between the *PreClassify* and *PostClassify* events which identify the beginning and end of the classification workflow step for a particular document. Using this new event the custom script can clean-up and/or extend classification results before the final decision has been made by the system and before the final classification matrix has been built. The event handler can be implemented for the project level script page.

1.2.1.12. ProcessBatch

ProcessBatch

Description This event is introduced to work with the new custom workflow step within the Runtime Server instance. The ProcessBatch event is launched when the RTS instance begins processing batches

matching the input state criteria.

Syntax

```
ScriptModule_ProcessBatch(pBatch As
SCBCdrPROJLib.ISCBCdrBatch, ByVal InputState As Long,
DesiredOutputStateSucceeded As Long,
DesiredOutputStateFailed As Long)
```

Parameters

pBatch: The Batch Object that is being processed.

InputState: The input state of the batch when Custom Processing was activated on it.

DesiredOutputStateSucceeded: The output state of the batch if the workflow step succeeds.

DesiredOutputStateFailed: The output state of the batch if the workflow step failed.

See also

Project Event

Example

```
Example:
Private Sub ScriptModule_ProcessBatch(pBatch As
SCBCdrPROJLib.ISCBCdrBatch, ByVal InputState As Long,
DesiredOutputStateSucceeded As Long, DesiredOutputStateFailed As
Long)
 Call LogError("   ProcessBatch Event was launched", "C:\EventTrace_"
& Format(Now,"DDMMYYYY") & ".Log")
End Sub
```

The below script should be added to the very beginning of the ProcessBatch event: This script helps to stop a indefinite looping process of state 0 batches.

This script does not set batches to special state 987. The script repairs a batch and stops looping of the custom processing step. Note that it is not possible to set the batch state to something other than zero for a batch with no documents, because batch state is by definition the lowest state of all enclosed documents. If the number of documents is zero, the application just uses the default value - which is zero.

Enhanced recovery script sample:

```
Private Sub ScriptModule_ProcessBatch(pBatch As SCBCdrPROJLib.ISCBCdrBatch,
ByVal InputState As Long, DesiredOutputStateSucceeded As Long,
DesiredOutputStateFailed As Long)


  Dim lFolderIndex As Long
  Dim lDocIndex As Long
  Dim theWorkdoc As SCBCdrWorkdoc
  Dim vLoadingCompletenessStatus As Variant
  Dim lStatus As Long
  Dim bNeedSafetyRestart As Boolean
  Dim strWorkdocName As String
  Dim theImage As SCBCroImage

  On Error GoTo LABEL_ERROR

  pBatch.BatchPriority = 3 '[AE] [2012-03-27] Boost priority for state zero
  documents

  Project.LogScriptMessageEx CDRTypeInfo, CDRSeveritySystemMonitoring,
  "ScriptModule_ProcessBatch starting, batch <" & CStr(pBatch.BatchID) & ">, new
  state <" & CStr(DesiredOutputStateSucceeded) & ">"
```

```
  If ScriptModule.ModuleName <> "Server" Then Exit Sub


  For lFolderIndex = pBatch.FolderCount – 1 To 0 Step -1
  If pBatch.FolderDocCount (lFolderIndex) = 0 Then
       Project.LogScriptMessageEx CDRTypeWarning, CDRSeveritySystemMonitoring,
       "Removed folder with zero documents from batch [" & pBatch.BatchID & "]"
pBatch.DeleteFolder(lFolderIndex, False)
  End If
  Next lFolderIndex

  If pBatch.FolderCount = 0 Then
       Project.LogScriptMessageEx CDRTypeWarning, CDRSeveritySystemMonitoring,
       "Detected batch with zero folders: [" & pBatch.BatchID & "]"
       pBatch.BatchState = 987
  End If

  On Error Resume Next

  For lFolderIndex = 0 To pBatch.FolderCount-1 Step 1
    For lDocIndex = pBatch.FolderDocCount(lFolderIndex) - 1 To 0 Step -1
       If pBatch.FolderDocState(lFolderIndex, lDocIndex) = InputState Then
Err.Clear
bNeedSafetyRestart = False
strWorkdocName = pBatch.FolderWorkdocFileName (lFolderIndex, lDocIndex, False)
Set theWorkdoc = pBatch.LoadWorkdoc(lFolderIndex, lDocIndex)
Project.LogScriptMessageEx CDRTypeInfo, CDRSeveritySystemMonitoring, "Loading of
zero state Workdoc [" & strWorkdocName & "] proceeded with error number [" &
CStr(Err.Number) & "] and error description [" & Err.Description & "]"


lStatus = 1001
If Err.Number = 0 Then
             vLoadingCompletenessStatus =
             theWorkdoc.NamedProperty("LoadingCompletenessStatus")
             lStatus = vLoadingCompletenessStatus
End If

If Err.Number <> 0 Or lStatus > 0 Then
             bNeedSafetyRestart = True
             Project.LogScriptMessageEx CDRTypeWarning,
             CDRSeverityEmailNotification, “True corruption case detected for
             Workdoc [“ & strWorkdocName & “] with stream exit code [“ & CStr
             (lStatus) & “]”
End If

Project.LogScriptMessageEx CDRTypeInfo, CDRSeveritySystemMonitoring,
"PreErrorChecks: Loading return code is {" & CStr(Err.Number) & "} and loading
status is {" & CStr(lStatus) & "}"

If (lStatus > 0 And lStatus <= 700) Then ' if this value is > 700 but <= 790,
then re-OCR is required, if it is greater than 790, then re-importing is needed
- extend the script below to set a different output state, other than the
standard "DesiredOutputStateSucceeded" one


             Project.LogScriptMessageEx CDRTypeInfo,
             CDRSeveritySystemMonitoring, "Loading return code is {" &
             CStr(Err.Number) & "} and loading status is {" & CStr(lStatus) &
             "}"
             Project.LogScriptMessageEx CDRTypeInfo,
             CDRSeveritySystemMonitoring, "Ignoring internal error when loading
             Workdoc [" & theWorkdoc.Filename & "]"
             Err.Clear


theWorkdoc.DocClassName = “”
theWorkdoc.Fields.Clear
theWorkdoc.RebuildBasicObjects
             If Err.Number <> 0 Then
                Project.LogScriptMessageEx CDRTypeWarning,
                CDRSeveritySystemMonitoring, "Recovery script:
                RebuildBasicObjects failed with error code [" &
                CStr(Err.Number) & "] and error description [" &
                Err.Description & "]"
                Err.Clear
                Project.LogScriptMessageEx CDRTypeWarning,
                CDRSeveritySystemMonitoring, "Recovery script: Proceeding with
                attempt to redirecting document to re-OCR state" ' [AE] [2012-
                02-27]
                DesiredOutputStateSucceeded = 100 ' [AE] [2012-02-27]
                theWorkdoc.DocState = CDRDocStateHaveDocs ' [AE] [2012-02-28]
                This call internally triggeres invoking of
                ".InternalClear(false,true)
             End If


             pBatch.FolderDocState(lFolderIndex, lDocIndex) =
             DesiredOutputStateSucceeded

             If Err.Number <> 0 Then
                Project.LogScriptMessageEx CDRTypeError,
                CDRSeveritySystemMonitoring, "Recovery script:
```

```
                    put_FolderDocState failed with error code [" & CStr(Err.Number)
                    & "] and error description [" & Err.Description & "]"
                    Err.Clear
                End If

                pBatch.UpdateDocument(theWorkdoc, lFolderIndex, lDocIndex)
                If Err.Number <> 0 Then
                    Project.LogScriptMessageEx CDRTypeError,
                    CDRSeveritySystemMonitoring, "Recovery script: UpdateDocument
                    failed with error code [" & CStr(Err.Number) & "] and error
                    description [" & Err.Description & "]"
                    Err.Clear
                End If

End If


If Err.Number <> 0 Or (lStatus > 700 And lStatus <= 790) Then ' if this value is
> 700 but <= 790, then re-OCR is required, if it is greater than 790, then re-
importing is needed - extend the script below to set a different output state,
other than the standard "DesiredOutputStateSucceeded" one

                Project.LogScriptMessageEx CDRTypeInfo,
                CDRSeveritySystemMonitoring, "Loading return code is {" &
                CStr(Err.Number) & "} and loading status is {" & CStr(lStatus) &
                "}"
                Project.LogScriptMessageEx CDRTypeInfo,
                CDRSeveritySystemMonitoring, "Ignoring internal error when loading
                Workdoc [" & theWorkdoc.Filename & "]"
                Err.Clear

                DesiredOutputStateSucceeded = 100
                theWorkdoc.DocState = CDRDocStateHaveDocs ' [AE] [2012-02-28] This
                call internally triggeres invoking of ".InternalClear(false,true)
                pBatch.FolderDocState(lFolderIndex, lDocIndex) =
                DesiredOutputStateSucceeded
                If Err.Number <> 0 Then
                    Project.LogScriptMessageEx CDRTypeError,
                    CDRSeveritySystemMonitoring, "Recovery script:
                    put_FolderDocState failed with error code [" & CStr(Err.Number)
                    & "] and error description [" & Err.Description & "]"
                    Err.Clear
                End If

                pBatch.UpdateDocument(theWorkdoc, lFolderIndex, lDocIndex)
                If Err.Number <> 0 Then
                    Project.LogScriptMessageEx CDRTypeError,
                    CDRSeveritySystemMonitoring, "Recovery script: UpdateDocument
                    failed with error code [" & CStr(Err.Number) & "] and error
                    description [" & Err.Description & "]"
                    Err.Clear
                End If


End If

            ' [AE] [2012-03-05] Test that recovery has been succeeded and the
            Workdoc can now be loaded with no issues. This is one extra safety
            solution: "Load document one more time to "test" and recover for
            (from) real document file corruptions".
            If lStatus > 0 And lStatus <= 790 Then
              Set theWorkdoc = Nothing
              Err.Clear
              Set theWorkdoc = pBatch.LoadWorkdoc(lFolderIndex, lDocIndex)
              vLoadingCompletenessStatus =
              theWorkdoc.NamedProperty("LoadingCompletenessStatus")
              lStatus = vLoadingCompletenessStatus
              If Err.Number <> 0 Or lStatus > 0 Then
                  lStatus = 799
              End If
            End If

            ' [AE] [2012-03-27] Additional check for consistency of loaded
            document files

            If lStatus = 0 Then
              Err.Clear
              Set theImage = theWorkdoc.Pages(0).Image(0)
              If Err.Number <> 0 Or theImage Is Nothing Then
                  lStatus = 999
                  bNeedSafetyRestart = True
              End If
            End If

If Err.Number <> 0 Or (lStatus > 790) Then ' if this value is > 700 but <= 790,
then re-OCR is required, if it is greater than 790, then re-importing is needed
- extend the script below to set a different output state, other than the
standard "DesiredOutputStateSucceeded" one
```

```
                    Project.LogScriptMessageEx CDRTypeInfo,
                    CDRSeveritySystemMonitoring, "Loading return code is {" &
                    CStr(Err.Number) & "} and loading status is {" & CStr(lStatus) &
                    "}"
                    Project.LogScriptMessageEx CDRTypeInfo,
                    CDRSeveritySystemMonitoring, "Ignoring internal error when loading
                    Workdoc [" & theWorkdoc.Filename & "]"
                    Project.LogScriptMessageEx CDRTypeWarning,
                    CDRSeverityEmailNotification, “Document [“ & strWorkdocName & “]
                    with stream exit code [“ & CStr (lStatus) & “] will be redirected
                    to manual processing state”
                    Err.Clear

                    DesiredOutputStateSucceeded = 850
                    pBatch.FolderDocState(lFolderIndex, lDocIndex) =
                    DesiredOutputStateSucceeded
                    If Err.Number <> 0 Then
                        Project.LogScriptMessageEx CDRTypeError,
                        CDRSeveritySystemMonitoring, "Recovery script:
                        put_FolderDocState failed with error code [" & CStr(Err.Number)
                        & "] and error description [" & Err.Description & "]"
                        Err.Clear
                    End If

                    ' [AE] [2012-03-27] Do not call update document in case of 850
                    type recovery - just update the document state via the call above

                    ‘ pBatch.UpdateDocument(theWorkdoc, lFolderIndex, lDocIndex)
                    ‘ If Err.Number <> 0 Then
                        ‘ Project.LogScriptMessageEx CDRTypeError,
                        CDRSeveritySystemMonitoring, "Recovery script: UpdateDocument
                        failed with error code [" & CStr(Err.Number) & "] and error
                        description [" & Err.Description & "]"
                        ‘ Err.Clear
                    ‘ End If

End If

Set theWorkdoc = Nothing

' [AE] [2012-03-05] Auto-apply the RTS instance restart after recovering every
single case of true document loading failure. This is to ensure that
corruption's side effects are not cumulated across multiple auto-recovered
documents and clean documents are not negatively affected by attempts to load a
corrupted one.
If bNeedSafetyRestart = True Then
                    Project.PerformScriptCommandRTS(1, 0, 0, "Applying safety
                    recovery restart")
                    GoTo LABEL_SUCCESS
End If

       End If
   Next lDocIndex
 Next lFolderIndex

 LABEL_SUCCESS:
   Project.LogScriptMessageEx CDRTypeInfo, CDRSeveritySystemMonitoring,
   "ScriptModule_ProcessBatch finished sucessfully, batch <" &
   CStr(pBatch.BatchID) & ">, new state <" & CStr(DesiredOutputStateSucceeded)
   & ">, old state <" & CStr(InputState) & ">"
   Exit Sub

LABEL_ERROR:
 Project.LogScriptMessageEx CDRTypeError, CDRSeveritySystemMonitoring,
 "ScriptModule_ProcessBatch, finished with Error: " & Err.Description


End Sub
```

Use the corresponding Terminate Event (see section 1.2.1.17) script instead to delete these empty batches. Do not use both scripts within one project, because the Terminate Event script will make it impossible to load the ProcessBatch script.

1.2.1.13. RouteDocument

RouteDocument

Description Routes a document to a special state, depending on the data of the current WorkDoc.

Syntax

```
ScriptModule_RouteDocument (pWorkdoc As ISCBCdrWorkdoc,
State As Single)
```

Parameters

pWorkdoc: Workdoc object, which was classified and extracted

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

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
```

For example, in an environment where the Batch folder is shared between multiple organisations (either country groups, or departments), it is possible to allocate verifiers their own workflow configurations.

The following script automatically sets the Batch status after extraction to a status which is country based (eg, GB is status 550, Germany is status 551…).

```
Private Sub ScriptModule_RouteDocument(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, State As Integer)
'Event triggered after an event execution completes.
If State = 550 And Not fnIsVerifier() Then   'If the batch state is
550 and document is not in verifier
Select Case CountryCode    'Check country code and set batch status
accordingly.
Case "GB"
State = 550
Case "DE"
State = 551
Case "BENL"
State = 552
Case "IE"
State = 553
Case "RU"
State = 554
Case "US"
State = 555
Case Else
State = 550
End Select

pWorkdoc.Save(pWorkdoc.Filename,"")   'Save the work doc after
changing document status
End If
End Sub
```

1.2.1.14. SecurityUpdateAddUserGroup

SecurityUpdateAddUserGroup and SecurityUpdateAddUserGroupPwd

Description

This method is used to update, or add, the database security credentials. This script call is used in creating or updating the Perceptive Intelligent Capture users, roles, and groups.

When updating the security policy of Perceptive Intelligent Capture via custom script, only the database tables will be updated. The project security will not be modified after a script update.

Use the *SecurityUpdateAddUserGroupPwd* method to import user accounts with predefined passwords.

Syntax

```
SecurityUpdateAddUserGroup(UserName As String,
ExternalGroupID As Long, UserRole As
String,UserDomain String)
```

```
SecurityUpdateAddUserGroupPwd (UserName As String,
UserPassword as String, ExternalGroupID As Long,
UserRole As String,UserDomain String)
```

Parameters

UserName: The Username to create or update within the database. This will be the user credentials to type to log into the system. If Domain is populated, the user must enter MyDomain\UserName for logging into the verification application.

UserPassword: This password will be applied only when creating a new user. For those auto-imported users that were previously imported into Perceptive Intelligent Capture, the password will remain unchanged.

Use case rules:

1. Auto-imported users with empty password are required to change their password upon first login.
2. Auto-imported users with NON-empty password are NOT required to change their password upon first login.
3. Auto-imported users who already changed their password upon first login will not be required to change their password anymore.

ExternalGroupID: The external group ID security number. A batch and a user can be assigned a group ID which would enable the user to verify

only batches which fall under the same group ID that is assigned to that user.

UserRole: The user role assigned to the verifier user. The role can be one of the following text strings:

- VER - Verifier user.
- SLV - Verifier supervisor (learnset nomination)
- SLM - Learnset Manager (global learnset manager)
- ADM - Administrator.
- FLT - Filtering

UserDomain: The user domain is left blank if no Windows Authentication is used. Or when using Windows Authentication, populated with the Domain name the Windows user belongs to.

The following combinations of roles are possible:

```
   Project.SecurityUpdateAddUserGroup "User2", 999, "VER|FLT",
"BDomain"
```

--> This creates a user with Verifier and Filter roles, but with no SET role

```
   Project.SecurityUpdateAddUserGroup "User2", 999, "VER|SET|FLT",
"BDomain"
```

--> This creates a user with Verifier, Settings and Filter roles

```
   Project.SecurityUpdateAddUserGroup "User2", 999, "VER",
"BDomain"
```

--> This creates a user with Verifier role only, with no SET and FLT role

There is no need to combine SET/FLT roles with ADM, SLV, or SLM as these are already containing FLT and SET roles by default.

See Also SecurityUpdateStart, SecurityUpdateCommit, UpdateSystemSecurity, PostImportBatch

Example The example below updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the Perceptive Intelligent Capture user table.

```
Private Sub ScriptModule_UpdateSystemSecurity(ByVal InstanceName
As String)
   Project.SecurityUpdateStart
   Project.SecurityUpdateAddUserGroup "User1", 777, "VER",
"BDomain"
   Project.SecurityUpdateAddUserGroup "User2", 999, "SLV",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User3", 111, "VER",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User4", 888, "SLM",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User5", 222, "SET",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User6", 777, "VER|FLT",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User10", 777, "ADM",
```

```
"BDomain "
   Project.SecurityUpdateCommit
End Sub
```

1.2.1.15. SecurityUpdateCommit

SecurityUpdateCommit

Description

This method completes the security update process. This script call is required in order to complete updating the Perceptive Intelligent Capture users, roles, and groups.

When updating the security policy of Perceptive Intelligent Capture via custom script, only the database tables will be updated. The project security will not be modified after a script update.

Syntax

`Project.SecurityUpdateCommit`

Parameters

There are no parameters for this method.

See Also

SecurityUpdateStart, SecurityUpdateAddUserGroup, UpdateSystemSecurity, PostImportBatch

Example

The example below updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the Perceptive Intelligent Capture user table.

```
Private Sub ScriptModule_UpdateSystemSecurity(ByVal InstanceName
As String)
   Project.SecurityUpdateStart
   Project.SecurityUpdateAddUserGroup "User1", 777, "VER",
"BDomain"
   Project.SecurityUpdateAddUserGroup "User2", 999, "SLV",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User3", 111, "VER",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User4", 888, "SLM",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User5", 222, "SET",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User6", 777, "VER|FLT",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User10", 777, "ADM",
"BDomain "
   Project.SecurityUpdateCommit
End Sub
```

1.2.1.16. SecurityUpdateStart

SecurityUpdateStart

Description

This method instantiates the security update process. This script call is required in order to begin updating the Perceptive Intelligent Capture users, roles, and groups.

When updating the security policy of Perceptive Intelligent Capture via custom script, only the database tables will be updated. The project security will not be modified after a script update.

Syntax

Project.SecurityUpdateStart

Parameters There are no parameters for this method.

See Also SecurityUpdateAddUserGroup, SecurityUpdateCommit, UpdateSystemSecurity, PostImportBatch

Example The example below updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the Perceptive Intelligent Capture user table.

```
Private Sub ScriptModule_UpdateSystemSecurity(ByVal InstanceName
As String)
   Project.SecurityUpdateStart
   Project.SecurityUpdateAddUserGroup "User1", 777, "VER",
"BDomain"
   Project.SecurityUpdateAddUserGroup "User2", 999, "SLV",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User3", 111, "VER",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User4", 888, "SLM",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User5", 222, "SET",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User6", 777, "VER|FLT",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User10", 777, "ADM",
"BDomain "
   Project.SecurityUpdateCommit
End Sub
```

1.2.1.17. Terminate

Terminate

Description The Terminate event is called before a batch is closed after processing.

Syntax

```
ScriptModule_Terminate (ModuleName as String)
```

Parameters *ModuleName:* Name of the current module, values: ‘‘Designer,’ "Verifier" or "Server"

Example

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

This script can be added to one of the real projects, triggering the Terminate event in RTS. This script will erase all state 0 batches that contain zero folders. Do not use this script piece together with the corresponding ProcessBatch Event Script (section 1.2.1.11) within one project, because this Terminate Event script will make it impossible to load the ProcessBatch script:

```
Private Sub ScriptModule_Terminate(ByVal ModuleName As String)


On Error GoTo LABEL_ERROR

Project.LogScriptMessageEx CDRTypeInfo, CDRSeveritySystemMonitoring,
"Processing ScriptModule_Terminate event"
```

```
Dim i As Long

Dim pBatchRoot As New SCBCdrBATCHLib.SCBCdrBatchRoot


pBatchRoot.ActivateSupport = True

pBatchRoot.SetConnectionProperties("Version 5.4 SP1 Job", "Zero
Folder Batch Terminator", False)

pBatchRoot.Connect("Version 5.4 SP1 Job", "", "LOGIN_AS_CURRENT", "",
"Zero Folder Batch Terminator")

pBatchRoot.SetFilter(0)


For i = 0 To pBatchRoot.BatchCount - 1 Step 1

If pBatchRoot.FolderCount(i) = 0 Then

Project.LogScriptMessageEx CDRTypeWarning,
CDRSeveritySystemMonitoring, "Zero Folder Batch Terminator detected
batch with zero folders: [" & pBatchRoot.BatchID(i) & "]"

pBatchRoot.DeleteBatch(pBatchRoot.BatchID(i), False, 0, 0)

End If

Next i


Exit Sub


LABEL_ERROR:

Project.LogScriptMessageEx CDRTypeWarning,
CDRSeveritySystemMonitoring, "Zero Folder Batch Terminator failed to
search for zero folder batches. Error description: " &
Err.Description


End Sub
```

1.2.1.18. UpdateSystemSecurity

UpdateSystemSecurity

Description

An event that is triggered when the Runtime Server is configured to run with security update.

Only one Runtime Server instance should be configured to update system security. The frequency of the security update is determined via the Runtime Server instance properties.

Syntax

```
ScriptModule_UpdateSystemSecurity(ByVal InstanceName As
String)
```

Parameters

InstanceName: The Runtime Server instance name that is calling the UpdateSystemSecurity event.

See Also

ScriptModule Events, SecurityUpdateStart, SecurityUpdateAddUserGroup, SecurityUpdateCommit, PostImportBatch

Example

The example below updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the Perceptive Intelligent Capture user table.

```
Private Sub ScriptModule_UpdateSystemSecurity(ByVal InstanceName As
String)
   Project.SecurityUpdateStart
   Project.SecurityUpdateAddUserGroup "User1", 777, "VER", "BDomain"
   Project.SecurityUpdateAddUserGroup "User2", 999, "SLV", "BDomain "
   Project.SecurityUpdateAddUserGroup "User3", 111, "VER", "BDomain "
   Project.SecurityUpdateAddUserGroup "User4", 888, "SLM", "BDomain "
   Project.SecurityUpdateAddUserGroup "User5", 222, "SET", "BDomain "
   Project.SecurityUpdateAddUserGroup "User6", 777, "VER|FLT",
"BDomain "
   Project.SecurityUpdateAddUserGroup "User10", 777, "ADM", "BDomain
"
   Project.SecurityUpdateCommit
End Sub
```

1.2.1.19. VerifierClassify

VerifierClassify

Description This event occurs only in Verifier when a document is manually classified.

Syntax

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

Parameters

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.

1.2.1.20. VerifierFormLoad

VerifierFormLoad

Description There is a project event that can be called within Perceptive Intelligent Capture which enables the user to switch verification forms between different types of classes or to default the Verifier application to display a certain page instead of the first one (see DisplayPage for more details).

Syntax

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

Parameters

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 Perceptive Intelligent Capture custom script to point to a different document class, in case the desired verification form is located in this different class.

Example

```
Private Sub ScriptModule_VerifierFormLoad(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, FormClassName As String, FormName As
String)
Select Case UCase(FormClassName)
Case "ALLRAUER"
FormClassName = "Invoices"
FormName = "Form_Invoices_2"
Case "BASH"
FormClassName = "Invoices"
FormName = "Form_Invoices_2"
Case "COMPUTER 2001"
FormClassName = "Invoices"
FormName = "Form_Invoices_2"
Case "CONTAC"
FormClassName = "Invoices"
FormName = "Form_Invoices_1"
Case "DRV"
FormClassName = "Invoices"
FormName = "Form_Invoices_1"
Case "RAB"
FormClassName = "Invoices"
FormName = "Form_Invoices_1"
Case "RUBIN"
FormClassName = "Invoices"
FormName = "Form_Invoices_2"
Case "XODEX"
FormClassName = "Invoices"
FormName = "Form_Invoices_2"
Case Else
FormClassName = "Invoices"
FormName = "Form_Invoices_1"
End Select
End Sub
```

1.3 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.

1.3.1. FocusChanged

FocusChanged

Description 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.

Syntax

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

Parameters

pWorkdoc: Reference to the currently displayed workdoc.

Reason: Reason of the current focus change, which can be 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.

Example

```
Example:
Privat 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").Act
iveTableIndex)
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
```

1.3.2. OnAction

OnAction

Description 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.

Syntax

```
Document_OnAction (pWorkdoc As ISCBCdrWorkdoc,
ActionName As String)
```

Parameters

pWorkdoc: Reference to the currently displayed workdoc.

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

Example

```
Sub Document_OnAction(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, ByVal
 ActionName As String)


    If ActionName = "ShowBestSuppliers" Then

       Call
 fnShowBestSuppliers(pWorkdoc,pWorkdoc.Fields(FIELDNAME),"", "", "")


    End If

End Sub
```

1.3.3. PostExtract

PostExtract

Description 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 classifications. 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.

Syntax

```
Document_PostExtract (pWorkdoc As ISCBCdrWorkdoc)
```

Parameters

pWorkdoc: Current Workdoc object

Example

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

'get fields name and number
Number = pWorkdoc.Fields(“Number”)
```

```
Name = pWorkdoc.Fields(“Name”)

End Sub
```

1.3.4. PreExtract

PreExtract

Description The PreExtract event will be called before any defined analysis or evaluation method will be executed by the Cedar DocClass.

Syntax

```
Document_PreExtract (pWorkdoc As ISCBCdrWorkdoc)
```

Parameters *pWorkdoc:* Current Workdoc object

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
```

1.3.5. PreVerifierTrain

PreVerifierTrain

Description A new PreVerifierTrain event has been added to control SLW training in Verifier, Learnset Manager, and Designer.

This event is called at the point when an applications starts learning for a document in the supervised learning workflow (SLW).

Syntax

```
Document_PreVerifierTrain(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, pMode As Long)
```

Parameters *pMode:* It is reserved for for further useand should not be used in the present software version.

Example The following script example demonstrates how the new script event can be used in order to apply a substitution of the primary Associative Seach Engine field with another result referring to a different pool.

```
Private Sub Document_PreVerifierTrain(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc, pMode As Long)
   If pWorkdoc.DocClassName = "NotGoodForPrimaryASEField" Then
Project.AllClasses.ItemByName("Invoices").ClassificationField =
"SecondaryAseField"
```

```
    End If
End Sub
```

1.3.6. Validate

Validate

Description

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.

Syntax

```
Document_Validate (pWorkdoc As ISCBCdrWorkdoc, pValid
As Boolean)
```

Parameters

pWorkdoc: Current Workdoc object

pValid: Parameter containing the current valid state of the Workdoc

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
```

1.3.7. VerifierTrain

VerifierTrain

Description 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

Syntax

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

Parameters

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.

1.4 <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.

1.4.1. CellChecked

CellChecked

Description Occurs when a check-box cell of the table has been checked or unchecked by the user.

Syntax

```
<Fieldn>_CellChecked (pTable As ISCBCdrTable, pWorkdoc
As ISCBCdrWorkdoc, Row As Long, Column As Long, Checked
As Boolean)
```

Parameters

pTable: Current Table object.

pWorkdoc: Current Workdoc object.

Row: This parameter contains the index of the current row on which the user clicked.

Column: This parameter contains the index 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.

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
```

1.4.2. CellFocusChanged

CellFocusChanged

Description This event occurs each time the focus inside the verification table is going to be changed or can be changed potentially.

Syntax

```
<Fieldn>_CellFocusChanged (pTable As ISCBCdrTable,
pWorkdoc As ISCBCdrWorkdoc, Reason As
CdrTableFocusChangeReason, OldRow As Long, OldColumn As
Long, pNewRow As Long, pNewColumn As Long)
```

Parameters

pTable: Current Table object.

pWorkdoc: Current Workdoc object.

Reason: Parameter that contains the kind of focus change that has occurred

OldRow: This parameter contains the index of the derivation row.

OldColumn: This parameter contains the index of the derivation column.

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

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

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
```

1.4.3. Format

Format

Description

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 Scripting Guide engines. If the user wants to change the output format for the fields' content rather use the script event FormatForExport.

Syntax `<Fieldn>_Format (pField As ISCBCdrField)`

Parameters *pField:* Field object

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
```

1.4.4. FormatForExport

FormatForExport

Description 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 Scripting Guide engines. This formatted text can also be used for Export.

Syntax `<Fieldn>_FormatForExport (pField As ISCBCdrField)`

Parameters *pField:* Current field.

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
```

1.4.5. PostAnalysis

PostAnalysis

Description 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.

Syntax `<Fieldn>_PostAnalysis (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)`

Parameters *pField:* Object containing the Field

pWorkdoc: Current Workdoc object

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
```

1.4.6. PostEvaluate

PostEvaluate

Description 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.

Syntax `<Fieldn>_PostEvaluate (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)`

Parameters

pField: Object containing the Field

pWorkdoc: Current Workdoc object

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
```

1.4.7. PreExtract

PreExtract

Description The PreExtract event will be called before any defined analysis or evaluation method for this Field is executed by the Cedar DocClass.

Syntax `<Fieldn>_PreExtract (pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)`

Parameters

pField: Object containing the Field

pWorkdoc: Current Workdoc object

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
```

1.4.8. SmartIndex

SmartIndex

Description The 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.

Syntax

```
<Fieldn>_SmartIndex (pField As ISCBCdrField, pWorkdoc
As ISCBCdrWorkdoc)
```

Parameters

pField: Object containing the current Field

pWorkdoc: Current Workdoc object

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
```

1.4.9. TableHeaderClicked

TableHeaderClicked

Description 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.

Syntax

```
<Fieldn>_TableHeaderClicked (pTable As ISCBCdrTable,
pWorkdoc As ISCBCdrWorkdoc, ClickType As
CdrTableHeaderClickType, Row As Long, Column As Long,
pSkipDefaultHandler As Boolean)
```

Parameters

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.

Row: This parameter contains the index of the current row on which the user clicked.

Column: This parameter contains the index 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.

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
```

1.4.10. Validate

Validate

Description The field Validate event can be used to perform project specific validation rules. Use the pValid parameter to return the validation decision. So if the parameter remains unchanged or if the event is not implemented, the document state gets valid if all fields are valid.

Syntax

```
<Fieldn>_Validate (pField As ISCBCdrField, pWorkdoc As
ISCBCdrWorkdoc, pValid As Boolean)
```

Parameters

pField: Object containing the current Field

pWorkdoc: Current Workdoc object

pValid: Parameter containing the current valid state of

the Field

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
```

1.4.11. ValidateCell

ValidateCell

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

Syntax

```
<Fieldn>_ValidateCell (pTable As ISCBCdrTable,
pWorkdoc As ISCBCdrWorkdoc, Row As Long, Column As
Long, pValid As Boolean)
```

Parameters

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.

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
```

1.4.12. ValidateRow

ValidateRow

Description Implement validation rules, which combine two or more cells of a row.

Syntax

```
<Fieldn>_ValidateRow (pTable As ISCBCdrTable, pWorkdoc
As ISCBCdrWorkdoc, Row As Long, pValid As Boolean)
```

Parameters

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

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
```

1.4.13. ValidateTable

ValidateTable

Description Implements a validation rule for the entire Table.

Syntax

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

Parameters

pTable: Table object

pWorkdoc: Current Workdoc object

pValid: Parameter containing the current valid state of the 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 2 Workdoc Object Reference (SCBCdrWorkdocLib)

2.1 SCBCdrWorkdoc

2.1.1. Description

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

2.1.2. Type Definitions

CDRDatabaseWorkflowTypes

The Workflow Type of the batch. These are standard Perceptive Intelligent Capture workflow settings for batches.
This type interface is a member of the Cedar project library.

Available Types	Description
CDRAutoTrainingFailed	Value 20
CDRAutoTrainingSucceeded	Value 19
CDRClassificationFailed	Value 8
CDRClassificationSucceeded	Value 7
CDRCleanupFailed	Value 26
CDRCleanupSucceeded	Value 25
CDRDocumentSeparationFailed	Value 6
CDRDocumentSeparationSucceeded	Value 5
CDREmailImportFailed	Value 32
CDREmailImportSucceeded	Value 31
CDRExportFailed	Value 24
CDRExportSucceeded	Value 23
CDRExtractionFailed	Value 10
CDRExtractionSucceeded	Value 9
CDRFileSystemExportFailed	Value 28
CDRFileSystemExportSucceeded	Value 27
CDRImportFailed	Value 2
CDRImportSucceeded	Value 1
CDRManualClassificationIncomplete	Value 14

CDRManualClassificationSucceeded	Value 13
CDRManualDocumentSeparationIncomplete	Value 12
CDRManualDocumentSeparationSucceeded	Value 11
CDRManualFinalValidationFullyIncomplete	Value 18
CDRManualFinalValidationSucceeded	Value 17
CDRManualTrainingFailed	Value 22
CDRManualTrainingSucceeded	Value 21
CDRModifiedByDesignerApplication	Value 33
CDRModifiedByVerifierApplication	Value 34
CDROCRFailed	Value 4
CDROCRSucceeded	Value 3
CDRPartialManualValidationIncomplete	Value 16
CDRPartialManualValidationSucceeded	Value 15
CDRReserved	Value 100
CDRReset	Value 0
CDRScanningFailed	Value 30
CDRScanningSucceeded	Value 29

CdrEdgeSide

The definition which determines the type of alignment/edges.

Available Types	Description
CDREdgeLeft	Chooses left alignment (left edges) in analysis.
CDREdgeRight	Chooses right alignment (right edges) in analysis.

CDRHighlightMode

The highlighting mode for the workdoc displaying for the user (e.g. highlight candidates, highlight fields only etc).

Available Types	Description
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 highlighting
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

This data type is responsible for specifying the result of classification for a specific document class and specific classification engine. This is the same as the cell inside the

classification matrix within Designer.

Available Types	Description
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	Classification weight property has valid content
CDRClassifyYes	For sure document belongs to this DocClass

CDRDocState

The definition which determines the current state of the document within the workflow.

Available Types	Description
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	Initial state of document
CDRDocStateValid	Validity state of document
CDRDocStateWorktext	Worktext is assigned to document

CDRPageAssignment

This data type is responsible for specifying how the Document Pages are assigned to the Workdoc.

Available Types	Description
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

This data type is responsible for specifying the export type of PDF image out of Perceptive Intelligent Capture.

Available Types	**Description**
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.

Available Types	**Description**
CDRDocFileTypeCroCIDoc	Cairo CIDocument
CDRDocFileTypeCroImage	Cairo image object
CDRDocFileTypeRawText	Created from plain text without document
CDRDocFileTypeUnknown	Unknown file type, maybe attachment

2.1.3. Methods and Properties

AddDocFile

Description Adds a file (CIDoc, image, raw text) into the workdoc.

Syntax

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

Parameters

FilePath:	Path to the file to be added
FileType:	Filetype of the specified file. CIDoc, Image etc.
Assignment:	It specifies how DocPages are assigned to the Workdoc

Example This code shows how to add a CI-PDF file to the workdoc.

```
ddDocFile("C:\coversheet.pdf",CDRDocFileTypeCroCIDoc,CDRPageAssignNewPa
ge)
```

AddField

Description Adds a Field to the Workdoc

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

Parameters

Name :	Contains the name for the new field

Example This example adds the field “AdditionalField” to the workdoc

```
ddField("AdditionalField")
```

AddHighlightRectangle

Description Adds a Highlight rectangle on the page described by the parameters below. Set HighlightMode SCBCDRHighlightRectangles to highlight all rectangles.

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

Parameters

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

Example
```
pWorkdoc.AddHighlightRectangle(10,10,100,100,1,vbCyan)
```

AnalyzeAlignedBlocks

Description 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.

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

Parameters

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

Description 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.

Syntax

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

Parameters

XDist:	Minimum X distance between two TextBlocks
YDist:	Minimum Y distance between two TextBlocks

AnalyzeEdges

Syntax

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

Description 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.

Parameters

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 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

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

Syntax

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

Parameters

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 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

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

Syntax
```
AnalyzeParagraphs ()
```

AppendWorkdoc

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

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

Parameters

	pWorkdoc:	Workdoc that is to be appended

AssignDocToPage

Description 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.

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

Parameters

	DocIndex:	Zero-based CIDoc or Image Index
	DocPage:	Zero-based DocPage inside the Image or CIDoc
	WorkdocPage:	Zero-based DocPage inside the Workdoc

AttractorColor

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

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

Example This example sets the AttractorColor to green

BatchID

Description A new property of the workdoc has been introduced to allow the developer to retrieve the Batch ID that the current workdoc resides in.

When a document is placed in a new exception batch, the attribute updates to the new Batch ID.

Attribute strBatchID As String (Read Only)

Example The script sample below shows how to retrieve the Batch ID.

```
chID As String
  strBatchID = pWorkdoc.NamedProperty("BatchID")
```

BlockColor

Syntax `BlockColor As OLE_COLOR (read/write)`

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

Example This example sets the color for block highlighting to cyan

```
lockColor = vbCyan
```

BlockCount

Description 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.

Syntax `BlockCount As Long (read only)`

Example This example writes the text of each block to the string array 'strBlockText'.

```
Dim intBlockCount As Integer
Dim i as Long
intBlockCount = pWorkdoc.BlockCount -1
ReDim strBlockText(intBlockCount)
For i=0 To intBlockCount
   strBlockText(i) = pWorkdoc.TextBlock(i).Text
Next i
```

CandidateColor

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

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

Example This example sets the candidate color to magenta

```
pWorkdoc.CandidateColor = vbMagenta
```

Clear

Description 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.

Syntax
```
Clear ()
```

ClearHighlightRectangles

Description Removes all highlight rectangles.

Syntax
```
ClearHighlightRectangles ()
```

ClsEngineConfidence

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

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

Parameters

lMethodIndex:	Zero-based engine index in collection of classification engines.

Example This example shows a message box with the confidence value for each classification engine.

```
ividualResult As Double
  Dim lEngineIndex As Long
  For lEngineIndex = 0 To Project.ClassifySettings.Count
  dblIndividualResult = (pWorkdoc.ClsEngineConfidence(lEngineIndex))
  MsgBox "The classification confidence is " & dblIndividualResult
```

ClsEngineDistance

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

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

Parameters

lMethodIndex: Zero-based engine index in collection of classification engines.

Example This example shows a message box for each class, showing the classification engine distance.

```
ividualResult As Double
  Dim lEngineIndex As Long
  For lEngineIndex = 0 To Project.ClassifySettings.Count
  dblIndividualResult = (pWorkdoc.ClsEngineDistance(lEngineIndex))
  MsgBox "The engine distance is " & dblIndividualResult
  Next lEngineIndex
```

ClsEngineResult

Description 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.

The method returns the classification matrix as CDRClassifyResult. See the type definition for further details.

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

Parameters

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 Perceptive Intelligent Capture 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.

Example The following example sets the classification result of the Brainware Classify Engine to YES for a document in docclass “VOID”. If Brainware Classify is the only engine or all other classes would be CDRClassifyNo, the document would get classified as VOID.

```
lsEngineResult(Project.ClassifySettings.ItemIndex("Brainware Classify
  Engine"), Project.AllClasses.ItemIndex("VOID"))= CDRClassifyYes
```

ClsEngineWeight

Description Provides access to the classification weights within the Classification Result Matrix.

Syntax

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

Parameters

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 Perceptive Intelligent Capture 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 is 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.

CreationDate

Description A new property of the workdoc has been introduced to allow the developer to retrieve the Creation Date of the current workdoc.

When a document is placed in a new exception batch, the attribute updates to a new date/time stamp.

Attribute Read Only

Example The script sample below shows how to retrieve the Creation Date.

```
tionDate As Date
 dtCreationDate = pWorkdoc.NamedProperty("CreationDate")
```

CreationDateAsFileTimeUTC

Description A new property of the workdoc has been introduced to allow the developer to retrieve the Creation Date in UTC of the current workdoc.

When a document is placed in a new exception batch, the attribute updates to a new date/time stamp.

Attribute Read Only

Example The script sample below shows how to retrieve the Creation Date.

```
Dim dtCreationDateUTC As Long
dtCreationDateUTC =
pWorkdoc.NamedProperty("CreationDateAsFileTimeUtc")
```

CreateFromWorktext

Description Creates Workdoc from the OCRed text of an Image.

Syntax `CreateFromWorktext (pWorktext As ISCBCroWorktext)`

Parameters

pWorktext:	Object pointer of Worktext.

CutPage

Description Cuts the current Workdoc and generates a new Workdoc from DocPages present after the given PageIndex.

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

Parameters

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

CurrentBatchState

Description This is a property which returns the temporary document batch state (a numeric value between 0 and 999. This value is set by the

methods LoadWorkdoc and UpdateDocument of the Cedar Batch component.

Syntax `pWorkdoc.CurrentBatchState (Read only)`

DeleteFile

Description Deletes all wdcs and corresponding TIFs of the Workdoc.

Syntax `DeleteFile (DeleteDocFiles As Boolean)`

Parameters *DeleteDocFiles:* Flag to inform whether to delete files or not

DisplayPage

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

Syntax `DisplayPage As Long (read/write)`

Example If a customer requires to default Verifier to display a specific page of each document instead of the first one, use the DisplayPage property in the script.

In the example below, the script looks at all pages greater than, or equal to, 4 and displays Page 3.

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

If pWorkdoc.PageCount >=3 Then pWorkdoc.DisplayPage = 2

End Sub
```

presents Page 1, 1-Page 2, 2-Page 3, etc.

DocClassName

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

Syntax `DocClassName As String (read/write)`

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
```

DocFileCount

Description Returns the number of documents from which the Workdoc was built from.

Syntax

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

DocFileDatabaseID – Unique ID

Description The read only property pWorkdoc.DocFileDatabaseID returns the database ID of document files attached to a Perceptive Intelligent Capture Workdoc. It corresponds to the [File].[Id] value in the database. The document file index has to be passed as a parameter when using DocFileDatabaseID property.

Use this property in custom script as a unique identifier of document files that were processed by Perceptive Intelligent Capture.

Attribute Read only

Syntax

```
DocFileDatabaseID (ByVal Index As long) As Long
```

Parameters

Index	The index parameter has a valid range from 0 to PageCount-1

Example

```
Dim lUniqueID As Long
lUniqueID = pWorkdoc.DocFileDatabaseID(pWorkdoc.DocFileCount - 1)
```

The script example above demonstrates how to retrieve the unique ID of the last document file attached to a Workdoc.

DocFileName

Description Returns the full pathname of a document (image or text file) the Workdoc was built from.

Syntax

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

Parameters

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

Example 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)
```

The script function below returns the TIF file creation date and can

be used.

```
Public Function fnGetFileDate(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc) As String
Dim FSO As New Scripting.FileSystemObject
Dim oFile As Scripting.File
Dim strFileName As String
Dim dtCreated As Date
strFileName = Replace(pWorkdoc.DocFileName(0),".wdc",".tif")
If FSO.FileExists(strFileName) Then
Set oFile = FSO.GetFile(strFileName)
dtCreated = oFile.DateCreated
fnGetFileDate = Month(dtCreated) & "/" & Day(dtCreated) & "/" &
Year(dtCreated)
End If
Set FSO = Nothing
Set oFile = Nothing
End Function
```

DocFileType

Description Returns the file type of the document by the specified index.

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

Parameters

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

DocState

Description Sets / returns the current state of the document.

Syntax `DocState As CDRDocState (read/write)`

EdgeCount

Description Returns the number of vertical edges found in a document.

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

Parameters

edgeSide:	Flag to distinguish between left and right edges.

ErrorDescription

Description Sets / returns an error description.

Syntax `ErrorDescription As String (read/write)`

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
```

FieldColor

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

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

Parameters *FieldValid:* If set to TRUE it specifies the color for valid Fields or it specifies the color for invalid Fields if FALSE.

Fields

Description Provides access to all Fields of a document.

Syntax `Fields As ISCBCdrFields (read only)`

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

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

Filename

Description Contains the database ID of the Workdoc itself. Returns the database workdoc ID/Name.

Note: To retrieve the filename of the image from which the workdoc was created please use the DocFileName property found above!

Syntax `Filename As String (read only)`

Folder

Description Access the Folder to which the Workdoc belongs to.

Syntax `Folder As ISCBCdrFolder (read only)`

FolderIndex

Description Provides the index of Folder a Workdoc belongs to.

Syntax `FolderIndex As Long (read only)`

ForceClassificationReview

Description In the application, the PostClassify event has been extended so that it can force a manual classification review even if the classification succeeded.

Attribute Read/Write

Example The script sample below shows how the manual classification process can be forced from custom script event "PostClassify".

```
Private Sub ScriptModule_PostClassify(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc)
If pWorkdoc.DocClassName = "VeryImportantClass" Then
pWorkdoc.ForceClassificationReview = True
End If
End Sub
```

GetEdge

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

Syntax

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

Parameters

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.

GetFileSizeKB

Description Retrieve the file size of an image/document via custom script.

Syntax

```
GetFileSizeKB(pWorkdoc As SCBCdrWorkdoc) As Integer
```

Example

```
Private Function GetFileSizeKB(pWorkdoc As SCBCdrWorkdoc) As Integer
Dim FSO As FileSystemObject
Dim ImageFile As File
On Error GoTo ErrHandler
Set FSO = New FileSystemObject
Set ImageFile = FSO.GetFile(pWorkdoc.DocFileName(0))
GetFileSizeKB = Round(ImageFile.Size/1024)
Exit Function
ErrHandler:
GetFileSizeKB = -1
End Function
```

GetWorktextForPageArea

Description A function which returns a worktext object from a specific location on a document. The worktext object will contain text and positional information relating to the area specified in GetWorktextForPageArea. This can be considered as a temporary

zone to read a piece of information via script and review the returned result for that area.

The area to search will start from Left and Top coordinates and finish at Width and Height coordinares, provided in pixels. These are the same coordinates that would be entered for a reading zone (see Designer User Guide, Setting up Zone Analysis).

The scripter may test their page area coordinates using a zone.

Syntax

```
GetWorktextForPageArea(Page, Left, Top, Width, Height,
IncludePartial)
```

Parameters

Page:	page number of the image. 0 represents the first page of a multi page document.
Left:	left coordinate of the page area
Top:	top most coordinate of the page area
Width:	width (length) of the area
Height:	height of the area
includePartial:	Boolean flag. > False – restricts reading of worktext to specified area > True - completes words that appear partially in the specified area with outside information

Example of includePartial

The word "Invoice" exists on the page, but our page area only captures "Inv". Setting *includePartial* to *False* will return only "Inv", setting *includePartial* to *True* will return the entire word "Invoice".

Example of the code to use

The example below takes the OCR results of the top left page area and places the result into the first row table cell.

```
Dim ptrWorkText As SCBCroWorktext
Set ptrWorkText = New SCBCroWorktext
Set ptrWorkText = pWorkdoc.GetWorktextForPageArea(0, 100, 100, 300,
300,True)
pWorkdoc.Fields.ItemByName("TableField").Table(0).CellWorktext(0,0) =
ptrWorkText
```

HighlightCandidate

Description Set / returns the position of highlighted Candidate.

Syntax

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

HighlightField

Description Sets / returns the position of the highlighted Field.

Syntax

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

HighlightMode

Description Sets / returns the current mode of highlighting.

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

Image

Description Returns an Image object for the specified DocPage of the Workdoc.

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

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

IsPlainText

Description Sets or returns if worktext is plain text or not.

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

Language

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

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

LineColor

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

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

Load

Description Loads a file from given root path and this root path is not the absolute path of the file.

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

Parameters *Filename:* Name of the file.

ImageRootPath: Relative path of the file.

PageCount

Description Returns the number of displayable DocPages of the Workdoc.

Syntax

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

Example

```
intImageCount=pWorkdoc.PageCount     'Get the number of pages in TIF
```

Pages

Description Returns the single DocPages of the Workdoc.

Syntax

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

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

Paragraph

Description Provides access to the paragraph array of the Workdoc.

Syntax

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

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

ParagraphCount

Description Returns the number of paragraphs in the document.

Syntax

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

PDFExport

Description Generates a PDF file from Workdoc based on the given export type.

Syntax

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

Parameters *FileName:* Name of the PDF file exported.

PDFGetInfoType

Description Returns the export type of given Page of PDF file.

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

Parameters *PageIdx:* Page number of PDF.

pExportStyle: Type of Export.

PDFSetInfoType

Description Sets the type of export of PDF.

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

Parameters *PageIdx:* Zero-based DocPage Number.

ExportStyle: Type of export

ReadZone

Description It is a part of the OCR-on-demand concept.

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

Parameters

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.

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

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.

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

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

Syntax

```
Refresh ()
```

RenameDocFile

Description To change the name of the CIDoc or Image at given DocIndex by the given new name.

Syntax

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

Parameters

DocIndex:	Specifies the zero-based CIDoc or Image Index.
NewName:	New name given to the document at DocIndex.

ReplaceFirstImage

Description Replaces first image in Workdoc.

Syntax

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

Parameters

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

Save

Description Saves a Workdoc with given filename and its DocFiles relatively at given ImageRootPath

Syntax

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

Parameters

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.

SetDocPageIndex

Description This method has been added to allow the script implementation of the page merging workflow step.

Example The following short script example shows how this new public method can be used to append one document to another.

```
: 0 To thePreviousWorkdoc.PageCount -1 Step 1

      theNextWorkdoc.InsertPage (thePreviousWorkdoc, j, True,
theNextWorkdoc.PageCount)

      theNextWorkdoc.Pages (theNextWorkdoc.PageCount -
1).SetDocPageIndex(0, j + 1)

   End If
```

ShowTooltips

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

Syntax `ShowTooltips As Boolean (read/write)`

SkipTrainingWithEngine

Description Identifies whether the specified trainable engine has to skip this document in the training process.

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

Parameters *bstrEngineName:* Name of classification engine.

Table

Description Returns a Table for given index of the Workdoc.

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

Parameters *Index:* Specifies the index of the Table. Valid indices are from 0 to TableCount-1.

TableCount

Description Returns the number of Table objects stored within the Workdoc.

Syntax `TableCount As Long (read only)`

TextBlock

Description Returns TextBlock by index of the Workdoc.

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

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

Textline

Description Returns text line by index of the Workdoc.

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

Parameters *Index:* Zero-based index.

TextlineCount

Description Retrieves the number of text lines present in a Workdoc.

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

TrainedWithEngine

Description Indicates whether this document is trained with the specified engine.

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

Parameters bstrEngineName: Name of engine.

UnloadDocs

Description Releases all the Images and CIDocs which belong to this Workdoc.

Syntax
```
UnloadDocs ()
```

Word

Description Provides access to the Word array of the Workdoc.

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

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

WordColor

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

Syntax `WordColor As OLE_COLOR (read/write)`

WordCount

Description Returns the number of Words of the Workdoc.

Syntax `WordCount As Long (read only)`

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
```

WordSegmentationChars

Description Sets / returns a string which contains the characters used for the segmentation of Words.

Syntax `WordSegmentationChars As String (read/write)`

Worktext

Description Provides access to the raw OCR results represented by the SCBCroWorktext object.

Syntax `Worktext As ISCBCroWorktext (read only)`

2.2 SCBCdrFields

2.2.1. Description

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

2.2.2. Methods and Properties

Add

Description Adds a new Field with the specified name to the Field Collection.

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

Parameters

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

Description Removes all items from the Collection and releases their reference count.

Syntax
```
Clear ()
```

Collection

Description Returns the Collection which is internally used to store the Fields.

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

Count

Description Returns the number of items within the Field Collection.

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

Example
```
Dim cindex as long, count as long, id as long
```

Item

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

ISCBCdrFields Collection.

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

Parameters *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

Description Returns an item from the Collection specified by index.

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

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

Example

```
strClassName = theProject.AllClasses.ItemByIndex(intClass).Name
```

ItemByName

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

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

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

Example

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

If
pWorkdoc.Fields.ItemByName("InteractiveTableExtractionAllowed").Text
= "No" Then

Project.AllClasses.ItemByName(pWorkdoc.DocClassName).Fields.ItemByNam
e("LineItems").AllowInteractiveExtraction = False

Else

Project.AllClasses.ItemByName(pWorkdoc.DocClassName).Fields.ItemByNam
e("LineItems").AllowInteractiveExtraction = True

End If

End Sub
```

ItemExists

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

Syntax `ItemExists (Name As String) As Boolean`

Parameters *Name:* Name of item to search for.

ItemIndex

Description The index of an item specified by name is returned.

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

Parameters *Name:* Name specifying an item in the Collection.

ItemName

Description The name of an item is returned specified by index.

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

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

MoveItem

Description Moves an item specified by OldIndex from OldIndex to NewIndex.

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

Parameters *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

Description Removes the specified item from the Collection and releases the reference count to this item.

Syntax `Remove (ItemName As String)`

Parameters *ItemName:* [in] Name of item to remove.

RemoveByIndex

Description Removes the specified item from the Collection and releases the

reference count to this item.

Syntax `RemoveByIndex (Index As Long)`

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

Rename

Description Renames the item specified by Oldname from OldName to NewName.

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

Parameters *OldName:* [in] Name of item to rename

NewName: [in] New name of item in Collection.

Tag

Description To store a variant for each item of the Collection.

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

Parameters *Index:* Specifies the item index, valid range from 1 to Count.

2.3 SCBCdrField

2.3.1. Description

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

2.3.2. Type Definitions

CDRFieldState

Enumeration containing the state of the Field.

Available Types	Description
CDRFieldStateAnalyzed	Field is analyzed
DRFieldStateEvaluated	Field is evaluated
CDRFieldStateFormated	Field is formatted
CDRFieldStateReset	Initial state of a Field
CDRFieldStateValid	Validity state of Field

2.3.3. Methods and Properties

ActiveTableIndex

Description Reads the position where the Table is activated or activate the Table at given zero-based index.

Syntax

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

Example

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

AddCandidate

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

Syntax

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

Parameters

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.

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
```

AddCandidate2

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

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

Parameters

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

Description Adds a Table into the Table array of this Field.

Syntax `AddTable ()`

BoostDigitsOnly

Description Sets/returns whether only digits should be boosted.

Syntax `BoostDigitsOnly as Boolean`

BoostField

Description Sets/returns whether a field should be boosted.

Syntax `BoostField as Boolean`

Candidate

Description Returns a Candidate of the Field. Returns the number of Candidates

of the Field

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

Parameters

Index:	Index of the Candidate. Valid indices are 0 to CandidateCount-1
Count:	CandidateCount As Long (read only)

CandidateByFilterID

Description Finds the first candidate by specified filter ID or creates a new one if no such candidate found.

Syntax
```
CandidateByFilterID (ByVal FilterID As Long, ByVal CreateNew As Boolean, pCandidateIndex As Long) as ISCBCdrCandidate
```

Parameters

Filter ID:	ByVal FilterID As Long
	ByVal CreateNew As Boolean
	pCandidateIndex As Long

CandidateCount

Description Returns the number of candidates for a field.

Syntax `CandidateCount As Long`

Changed

Description 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.

Syntax `Changed As Boolean (read/write)`

CustomDetailsString

Description Sets / returns CustomDetailsString

Syntax `CustomDetails as String`

CustomStatusLong

Description Sets / returns CustomStatusLong

Syntax

```
CustomStatus As Long
```

DeleteLine

Description Deletes a line from specific index position.

Syntax

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

Parameters *LineIndex*: Index of Line, zero-based indexing

Example

```
'This loop deletes the existing line objects in the field:
Dim lngLineCounter As Long
For lngLineCounter = (pField.LineCount - 1) To 0 Step -1
pField.DeleteLine(lngLineCounter)
Next
'Then add as many lines as required and populate with the required
string:
pField.InsertLine(0)
pField.Line(0)="Line1"
pField.InsertLine(1)
pField.Line(1)="Line2"
```

DeleteTable

Description Deletes a Table from the Table array of this Field.

Syntax

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

Parameters *TableIndex:* Zero-based Index of the Table

ErrorDescription

Description Stores the reason if a script validation could not be performed successfully.

Syntax

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

Example

```
Private Sub Number_Validate(pField As SCBCdrField, pWorkdoc As
```

```
SCBCdrWorkdoc, pValid As Boolean)
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
```

ExternalText

Description Sets / returns external text

Syntax `ExternalText As String`

FieldID

Description This read-only property returns the internally used FieldID.

Syntax `FieldID As Long (read only)`

FieldState

Description Sets / returns the current execution state of the Field.

Syntax `FieldState As CDRFieldState (read/write)`

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
```

FieldVersion

Description Returns the field data of the specified version.

Syntax

```
FieldVersion As String (ByVal index As Long)
```

Parameters

Index:	ByVal index As Long

FindCandidate

Description Searches inside the list of Candidates if there is a Candidate based on the specified WordID.

Syntax

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

Parameters

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.

FindCandidateByPos

Description This is a method to find a candidate by its position.

Syntax

```
FindCandidateByPos (ByVal Page as Long, ByVal Param1 as
Long, ByVal Left as Long, ByVal Top as Long, ByVal
Width as Long, By Val Height as Long, CandidateIndex as
Long) as ISCBCdrCandidate
```

Parameters

ByVal Page:	Long
ByVal Param1:	Long
ByVal Left:	Long
ByVal Top:	Long
ByVal Width:	Long
ByVal Height:	Long
CandidateIndex	Long

:

FormattedText

Description This property cannot be used. The contents can be formatted via the FormatForExport field event. For details see section 1.4.4 FormatForExport.

GetFirstCandidatePropsByPage

Description This is a method to get the first candidate's properties by page.

Syntax

```
CandidatePropsByPage (ByVal Page As Long, ByVal Param1
As Long, ByVal Left As Long, ByVal Top As Long, ByVal
Width As Long, ByVal Height As Long, ByVal Text As
String, ByVal Weight As Double) as Long
```

Parameters *CandidateProps:*

ByVal Page As Long
ByVal Param1 As Long
ByVal Left As Long
ByVal Top As Long
ByVal Width As Long
ByVal Height As Long
ByVal Text As String
ByVal Weight As Double

GetNextCandidatePropsByPage

Description This is a method to get the next candidate's properties by page.

Syntax

```
CandidatePropsByPage (ByVal Left As Long, ByVal Top As
Long, ByVal Width As Long, ByVal Height As Long, ByVal
Text As String, ByVal Weight As Double) as Long
```

Parameters *CandidateProps:*

ByVal Left As Long
ByVal Top As Long
ByVal Width As Long
ByVal Height As Long
ByVal Text As String

ByVal Weight As Double

GetUniqueEntryId

Description Retrieves other column values for the specified pool entry.

Syntax

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

Parameters

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

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

Example

```
Public Function GetASSAInfo (pworkdoc as SCBCdrPROJLib.SCBCdrWorkdoc,
cand as SCBCdrWkDocLib.SCBCdrCandidate) As String

'Function input: Workdoc, ASSA Candidate

Dim lNumericIdHigh As Long

Dim lNumericIdLow As Long

GetASSAInfo=""

If cand.IsIDAlphNum = True Then

GetASSAInfo = cand.UniqueID

Else

GetASSAInfo = Cand.GetUniqueEntryID(lNumericIDhigh, lnumericIdLow)

End If

End Function
```

Height

Description Sets / returns the height of the Field in pixel.

Syntax

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

Example

```
'copy the positional information to the new object
pCopyField.Height = pField.Height
```

InsertLine

Description Insert a line at given LineIndex in a Field.

Syntax

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

Parameters

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

Example The following script code should be used when attempting to insert new lines to a field in custom script:

```
'This loop deletes the existing line objects in the field:
Dim lngLineCounter As Long
For lngLineCounter = (pField.LineCount - 1) To 0 Step -1
pField.DeleteLine(lngLineCounter)
Next
'Then add as many lines as required and populate with the required
string:
pField.InsertLine(0)
pField.Line(0)="Line1"
pField.InsertLine(1)
pField.Line(1)="Line2"
Attempting to use pfield.text="Line1" + VbCrLf & "Line2" will not
work.
```

IsIDAlphNum

Description Sets / returns whether a unique ID is alphanumeric.

Syntax `IsIDAlphNum As Boolean`

LastModificationEndDate

Description Sets / returns LastModificationEndDate.

Syntax `LastModificationEndDate As Date`

LastModificationEndDateAsFileTimeUtc

Description Sets / returns the height of the Field in pixel.

Syntax `Height As Long (read/write)`

Example

```
'copy the positional information to the new object
pCopyField.Height = pField.Height
```

Left

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

Syntax `Left As Long (read/write)`

Line

Description Sets / returns the text of a single line.

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

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

LineCaption

Description 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.

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

Parameters *Index:* Index of the line, must be from 0 to LineCount-1

LineCount

Description Returns the number of lines of a multi-line header field. This equals the number of Worktext objects (In Perceptive Intelligent Capture, each line of a multi-line header field is represented by a separate individual Worktext object).

Can also be used to set the number of lines of a Field.

Syntax `LineCount As Long (read/write)`

LineWorktext

Description Provides access to the Worktext of each single line of the Field. The line index corresponds to the Worktext object.

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

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

MultilineText

Description Sets or returns multiline text for all lines at once that are separated with line break chars (same as "vbCrLf" in WinWrap script).

Syntax `MultilineText As String (read/write)`

Name

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

Syntax `Name As String (read only)`

PageNr

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

Syntax `PageNr As Long (read/write)`

PutUniqueEntryId

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

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

Parameters

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

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

Example

```
Candidate As long
Dim lngUniqueID As Long
lngUniqueID =
pWorkdoc.Fields("VendorASSA").Candidate(intNewCandidate).FilterID
pWorkdoc.Fields("VendorASSA").PutUniqueEntryId(0, lngUniqueID)
```

RemoveCandidate

Description Removes a Candidate from the Candidate array.

Syntax `RemoveCandidate (CandIndex As Long)`

Parameters *CandIndex:* Zero-based Candidate Index.

SkipTrainingWithEngine

Description Identifies whether the specified trainable engine has to skip this field in the training process.

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

Parameters *bstrEngineName:* Name of the extraction engine.

Table

Description Retrieves the Table object from an array of Tables of this Field at a specified index.

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

Parameters *Index:* Position of a Table in an array of Tables, zero-based indexing

TableCount

Description Returns the number of Tables according to the Field.

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

Tag

Description To store an arbitrary variant in the Field.

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

Text

Description To read and write the text of the Field. In case of multi-line Fields, the Text property refers to all lines at once as one single string, combining lines with spaces in between.

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

Top

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

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

TrainedWithEngine

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

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

Parameters bstrEngineName: Name of the Engine

Valid

Description Sets / returns the valid state of the Field.

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

Width

Description Sets / returns the width of the Field in pixel.

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

Worktext

Description Provides access to the Worktext of the Field. In case of multi-line Fields, the Worktext property refers to the first Worktext the header field consists of, which represents the first line of the multi-line header field.

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

2.4 SCBCdrCandidate

2.4.1. Description

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

2.4.2. Methods and Properties

Attractor

Description Returns the attractor of the Candidate by a zero-based index.

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

```
only)
```

Parameters *Index:* Specifies the index in the attractor array, must be between 0 and AttractorCount - 1.

AttractorCount

Description Returns the number of attractors for this Candidate.

Syntax

```
AttractorCount As Long (read only)
```

CopyToField

Description To copy all required properties from the Candidate to the Field result.

Syntax

```
CopyToField (pField As ISCBCdrField)
```

Parameters *pField:* Reference to the Field containing the Candidate. States which field should get the values from the Candidate.

FilterID

Description This is the FilterID value as it was specified by the AddCandidate method of the Field.

Syntax

```
FilterID As Long (read only)
```

Example

```
Candidate As long
Dim lngUniqueID As Long
lngUniqueID =
pWorkdoc.Fields("VendorASSA").Candidate(intNewCandidate).FilterID
pWorkdoc.Fields("VendorASSA").PutUniqueEntryId(0, lngUniqueID)
```

FormatConfidence

Description Sets / returns the confidence of the string match algorithm performed by the format search engine that has created the Candidate.

Syntax

```
FormatConfidence As Double (read/write)
```

Height

Description Returns the height of the Candidate in pixel.

Syntax `Height As Long (read only)`

KeepSpaces

Description It specifies if the text created from several Words should keep the spaces between these Words or not.

Syntax `KeepSpaces As Boolean (read/write)`

Left

Description Returns the left border of the Candidate in pixel.

Syntax `Left As Long (read only)`

Line

Description Returns the text of a single line. A Candidate can consist of one or more lines.

Syntax `Line (index As Long) As String (read only)`

Parameters *Index:* Index of the Line, must be from 0 to LineCount-1.

LineCaption

Description 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.

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

Parameters *Index:* Index of the line, must be from 0 to LineCount – 1

LineCount

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

Syntax `LineCount As Long (read/write)`

LineWordCount

Description Returns number of words of the specified line.

Syntax `LineWordCount (index As Long) As Long (read only)`

Parameters *Index:* Index of the line.

LineWordID

Description Returns the Word ID of the specified Line and Word index.

Syntax
```
LineWordID (LineIndex As Long, WordIndex As Long) As
Long (read only)
```

Parameters *LineIndex:* Index of the Line, must be from 0 to LineCount-1.

WordIndex: Index of the Word within the Line.

LineWorktext

Description Returns the Worktext object of the single line specified by the zero-based index within a multi-line Field

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

Parameters *Index:* Zero-based index of single line

PageNr

Description Returns the DocPage number where the Candidate is located.

Syntax `PageNr As Long (read only)`

Example
```
Private Sub RestoreFieldPosition(pField As SCBCdrField, pCopyField As
SCBCdrField)
'write the saved fields positional data back to the original field
pField.PageNr = pCopyField.PageNr
End Sub
```

RemoveAttractor

Description Removes the attractor specified by index.

Syntax
```
RemoveAttractor (AttractorIndex As Long)
```

Parameters

AttractorIndex:	Index of attractor to be removed, valid range from 0 to AttractorCount-1.

Text

Description Returns the text of the Candidate.

Syntax
```
Text As String (read only)
```

Top

Description Returns the top border of the Candidate in pixel.

Syntax
```
Top As Long (read only)
```

Weight

Description Sets / returns the result of the evaluation which is between 0 and 1.

Note: the value can be higher than 1 (1 equals 100 %) in case the sum of different single candidate weights resulting from position and environment of the candidate exceeds 100 %. Candidates with more than 100 % will also be accounted for selection.

Syntax
```
Weight As Double (read/write)
```

Width

Description Returns the width of the Candidate in pixel.

Syntax
```
Width As Long (read only)
```

WordCount

Description Returns the Word count of the Candidate.

Syntax `WordCount As Long (read only)`

WordID

Description Returns the Word ID of the specified Word index within the first line.

Syntax `WordID (index As Long) As Long (read only)`

Parameters *Index:* Zero-based index of the Word within the line.

Worktext

Description Returns the Worktext object of the first line.

Syntax `Worktext As ISCBCroWorktext (read only)`

2.5 SCBCdrTable

2.5.1. Descriptions

The Cedar Table object represents a logical Table in a Document which is assigned to a Cedar Field of a Workdoc.

2.5.2. Type Definitions

CDRTableHighlightMode

Enumeration containing the highlighting mode of a Table.

Available Types	Description
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.

Available Types	Description
CDRLocationBottom	Bottom corner coordinate
CDRLocationLeft	Left corner coordinate
CDRLocationRight	Right corner coordinate
CDRLocationTop	Top corner coordinate

2.5.3. Methods and Properties

AddColumn

Description Adds a new column to a Table. Returns the index of the new column (zero-based).

Syntax

```
AddColumn (ColumnName As String) As Long
```

Parameters *ColumnName:* [in] Name of column

AddRow

Description Adds a new row to a Table. Returns the index of the new row (zero-based).

Syntax

```
As Long
```

AddUMColumn

Description Adds a new unmapped column to a Table. Returns the index of the new unmapped column.

Syntax

```
AddUMColumn (pUMColumnIndex As Long)
```

Parameters *pUMColumnIndex:* The method returns the zero-based index of the new column to this parameter.

AppendRows

Description Appends new rows over the specified range within the document.

Syntax
```
AppendRows (top As Long, height As Long, PageNumber As Long)
```

Parameters

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

Description Sets / returns the color of the Table cell.

Syntax
```
CellColor (IsValid As Boolean) As OLE_COLOR (read/write)
```

Parameters

IsValid:	Flag indicating if color refers to valid or invalid Table cells

CellLocation

Description Sets / returns the location of the Table cell.

Syntax
```
CellLocation (Column As Variant, RowIndex As Long, Location As CDRLocation)
As Long (read/write)
```

Parameters

Column:	Zero-based index or name of column
RowIndex:	Zero-based index of row
Location:	Location parameter

CellText

Description Sets / returns the text of the Table cell

Syntax
```
CellText (Column As Variant, RowIndex As Long) As String (read/write)
```

Parameters

Column:	Zero-based index or name of column
RowIndex:	Zero-based index of row

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
```

CellValid

Description Sets / returns the validity flag of the Table cell.

Syntax

```
CellValid (Column As Variant, RowIndex As Long) As
Boolean (read/write)
```

Parameters

Column: Zero-based index of name of column

RowIndex: Zero-based index of row

Example

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

CellValidationErrorDescription

Description Sets / returns the ErrorDescription for the cell validation.

Syntax

```
CellValidationErrorDescription (Column As Variant,
RowIndex As Long) As String (read/write)
```

Parameters

Column: Zero-based index or name of column

RowIndex: Zero-based index of row

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
```

CellVisible

Description Sets / returns Visible flag of the Table cell (currently not used).

Syntax `CellVisible (Column As Variant, RowIndex As Long) As Boolean (read/write)`

Parameters

Column: Zero-based index of name of column

RowIndex: Zero-based index of row

CellWorktext

Description Sets / returns the Worktext object of the cell.

Syntax `CellWorktext (Column As Variant, RowIndex As Long) As ISCBCroWorktext (read/write)`

Parameters

Column: Zero-based index or name of column

RowIndex: Zero-based index of row

CellWorktextChanged

Description Sets / returns a flag indicating whether the cell Worktext has changed.

Syntax `CellWorktextChanged (Column As Variant, RowIndex As Long) As Boolean (read/write)`

Parameters

Column: Zero-based index or name of column

	RowIndex:	Zero-based index of row

Clear

Description	Clears the content of the Table (i.e. removes all columns and all rows and resets all Table attributes).
Syntax	`Clear ()`

ClearColumn

Description	Clears the content of an existing column.	
Syntax	`ClearColumn (Column As Variant)`	
Parameters	*Column:*	Zero-based index or name of column

ClearRow

Description	Clears the content of an existing row.	
Syntax	`ClearRow (RowIndex As Long)`	
Parameters	RowIndex:	Zero-based index of row.

ClearUMColumn

Description	Clears the content of an unmapped column.	
Syntax	`ClearUMColumn (UMColumnIndex As Long)`	
Parameters	*UMColumnIndex:*	Zero-based index of unmapped column to be cleared.

ColumnColor

Description	Sets / returns the color of a column.	
Syntax	`ColumnColor (IsValid As Boolean) As OLE_COLOR (read/write)`	
Parameters	IsValid:	Flag indicating if color refers to valid or invalid columns

ColumnCount

Description Returns the number of the columns.

Syntax
```
ColumnCount As Long (read only)
```

ColumnExportEnable

Description Sets / returns the ExportEnable flag of a column.

Syntax
```
ColumnExportEnable (Column As Variant) As Boolean
(read/write)
```

Parameters

Column:	Zero-based index or name of column

ColumnIndex

Description Returns the column index for the name of a column.

Syntax
```
ColumnIndex (ColumnName As String) As Long (read
only)
```

Parameters

ColumnName:	name of the column

ColumnLabelLocation

Description Sets / returns the location of a column label (referring to first label line in case of multi-page Tables).

Syntax
```
ColumnLabelLocation (Column As Variant, Location As
CDRLocation) As Long (read/write)
```

Parameters

Column:	Zero-based index or name of column
Location:	Location parameter

ColumnLabelText

Description Sets / returns the column label.

Syntax
```
ColumnLabelText (Column As Variant) As String
(read/write)
```

Parameters

Column:	Zero-based index or name of column

ColumnLocation

Description Sets / returns the location of the column.

Syntax
```
ColumnLocation (Column As Variant, PageNr As Long,
Location As CDRLocation)
As Long (read/write)
```

Parameters

Column:	Zero-based index or name of column
PageNr:	DocPage number
Location:	Location parameter

ColumnMapped

Description Sets / returns a flag indicating whether a column has been mapped.

Syntax
```
ColumnMapped (Column As Variant) As Boolean
(read/write)
```

Parameters

Column:	Zero-based index or name of column

ColumnName

Description Returns the name of a column.

Syntax
```
ColumnName (ColumnIndex As Long) As String (read
only)
```

Parameters

ColumnIndex:	Zero-based Index of column

ColumnValid

Description 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.

Syntax
```
ColumnValid (Column As Variant) As Boolean
(read/write)
```

Parameters

Column:	Zero-based index or name of column

ColumnVisible

Description Sets / returns the visible flag of a column. (affects visibility of

column in *Verifier*).

Syntax `ColumnVisible (Column As Variant) As Boolean (read/write)`

Parameters *Column:* Zero-based index or name of column

Example
```
theTableSettings.ColumnVisible(2) = True    'Set the Column visible to True to show, False to hide.
```

DeleteColumn

Description Deletes a column specified by its name or by index.

Syntax `DeleteColumn (Column As Variant)`

Parameters *Column:* Zero-based index or name of column

DeleteRow

Description Deletes a row specified by index.

Syntax `DeleteRow (RowIndex As Long)`

Parameters *RowIndex:* Zero-based index of row

DeleteUMColumn

Description Deletes an unmapped column specified by index.

Syntax `DeleteUMColumn (UMColumnIndex As Long)`

Parameters *UMColumnIndex:* Zero-based index of unmapped column to be deleted

FieldName

Description Sets / returns the name of the CdrField to which the CdrTable object belongs to.

Syntax `FieldName As String (read/write)`

FillColumn

Description 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.

Syntax

```
FillColumn (left As Long, top As Long, width As
Long, height As Long,
PageNumber As Long, Column As Variant)
```

Parameters

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

Description Sets / returns the location of the Table footer.

Syntax

```
FooterLocation (Location As CDRLocation) As Long
(read/write)
```

Parameters

Location:	Location parameter

FooterPageNr

Description Sets / returns the DocPage number of the Table footer.

Syntax

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

FooterText

Description Sets / returns the text of the Table footer.

Syntax

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

HeaderLocation

Description Sets / returns the location of the Table header.

Syntax `HeaderLocation (Location As CDRLocation) As Long (read/write)`

Parameters *Location:* Location parameter

HeaderPageNr

Description Sets / returns the DocPage number of the Table header.

Syntax `HeaderPageNr As Long (read/write)`

HeaderText

Description Sets / returns the text of the Table header.

Syntax `HeaderText As String (read/write)`

HighlightColumnIndex

Description Sets / returns the index of the column to be highlighted.

Syntax `HighlightColumnIndex As Long (read/write)`

HighlightMode

Description 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)

Syntax `HighlightMode As CDRTableHighlightMode (read/write)`

HighlightRowIndex

Description Sets / returns the index of the row to be highlighted.

Syntax `HighlightRowIndex As Long (read/write)`

HighlightUMColumnIndex

Description Sets / returns the zero-based index of an unmapped column to be highlighted.

Syntax `HighlightUMColumnIndex As Long (read/write)`

InsertColumn

Description Inserts a new column after by ColumnIndex specified column.

Syntax `InsertColumn (ColumnIndex As Long, ColumnName As String)`

Parameters

ColumnIndex:	Zero-based index of existing column, behind which new column is to be inserted.
ColumnName:	Name of new column

InsertRow

Description Inserts a new row after specified RowIndex.

Syntax `InsertRow (RowIndex As Long)`

Parameters

RowIndex:	Zero-based index of existing row, below which new row is to be inserted.

InsertUMColumn

Description Inserts new unmapped column.

Syntax `InsertUMColumn (UMColumnIndex As Long)`

Parameters

UMColumnIndex:	Zero-based index of new column.

LabellinePageNr

Description Sets / returns the DocPage number of the label line (first occurrence in case of multi-page Tables).

Syntax `LabellinePageNr As Long (read/write)`

LocationExplicit

Description Sets / returns LocationExplicit flag.

Syntax `LocationExplicit As Boolean (read/write)`

MapColumn

Description Maps an unmapped column, i.e. transfers content of unmapped source column to specified target column.

Syntax `MapColumn (UMColumnIndex As Long, Column As Variant)`

Parameters

UMColumnIndex:	Zero-based index of unmapped source column
Column:	Zero-based index or name of destination column

MergeRows

Description Merges two rows specified by two indices.

Syntax `MergeRows (RowIndex1 As Long, RowIndex2 As Long)`

Parameters

RowIndex1:	Zero-based index of row 1
RowIndex2:	Zero-based index of row 2

RemoveAllColumns

Description This method removes all mapped table columns.

Syntax `RemoveAllColumns ()`

RemoveAllRows

Description This method removes all table rows.

Syntax `RemoveAllRows ()`

RemoveAllUMColumns

Description This method removes all unmapped table columns.

Syntax

```
RemoveAllUMColumns ()
```

RowColor

Description Sets / returns the color of the row.

Syntax

```
RowColor (IsValid As Boolean) As OLE_COLOR
(read/write)
```

Parameters

IsValid: Flag indicating if color refers to valid or invalid rows

RowCount

Description Returns the number of the rows.

Syntax

```
RowCount As Long (read only)
```

RowLocation

Description Sets / returns the location of the row.

Syntax

```
RowLocation (RowIndex As Long, Location As
CDRLocation) As Long (read/write)
```

Parameters

RowIndex: Zero-based index of row

Location: Location parameter

RowNumber

Description This property sets or returns the actual number of row.

Syntax

```
RowNumber (RowIndex As Long) As Long (read/write)
```

Parameters

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)
Dim nCurrentRow, nRow, nLine As Integer
While (nLine < pTable.RowCount) And (nRow = nCurrentRow)
nRow = pTable.RowNumber(nLine)
nLine = nLine + 1
Wend
End Sub
```

RowPageNr

Description Sets / returns the DocPage number of a row.

Syntax `RowPageNr (RowIndex As Long) As Long (read/write)`

Parameters *RowIndex:* Zero-based index of row

RowValid

Description Sets / returns a validity flag of a row.

Syntax `RowValid (RowIndex As Long) As Boolean (read/write)`

Parameters RowIndex: Zero-based index of row

RowValidationErrorDescription

Description Sets / returns an ErrorDescription for a row validation.

Syntax `RowValidationErrorDescription (RowIndex As Long) As String (read/write)`

Parameters *RowIndex:* Zero-based index of 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
```

Significance

Description Sets / returns the significance for corresponding evaluation property of the Table.

Syntax
```
Significance (EvalPropIndex As Long) As Double (read/write)
```

Parameters *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

Description Swaps the two specified columns.

Syntax
```
SwapColumns (ColumnIndex1 As Long, ColumnIndex2 As Long)
```

Parameters *ColumnIndex1:* Zero-based index of column 1

ColumnIndex2: Zero-based index of column 2

TableColor

Description Sets / returns the color of the Table.

Syntax
```
TableColor (IsValid As Boolean) As OLE_COLOR (read/write)
```

Parameters *IsValid:* Flag indicating if color refers to a valid or an invalid Table.

TableFirstPage

Description Sets / returns the DocPage number of the beginning of a Table (must be set after creation of a Table, but cannot chang afterwards).

Syntax
```
TableFirstPage As Long (read/write)
```

TableLastPage

Description 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 change afterwards).

Syntax

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

TableLocation

Description Sets / returns the location of a Table.

Syntax

```
TableLocation (PageNr As Long, Location As
CDRLocation) As Long (read/write)
```

Parameters

PageNr:	DocPage number
Location:	Location parameter

TableValid

Description Sets / returns a validity flag of the Table.

Syntax

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

TableValidationErrorDescription

Description Sets / returns an ErrorDescription for the Table validation.

Syntax

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

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
```

Tag

Description Sets / returns a tag associated with the Table.

Syntax `Tag As String (read/write)`

TotalSignificance

Description Sets / returns the total significance of the Table.

Syntax `TotalSignificance As Double (read/write)`

UMCellColor

Description Sets / returns the color of an unmapped Table cell.

Syntax `UMCellColor As OLE_COLOR (read/write)`

UMCellLocation

Description Sets / returns the location of an unmapped Table cell

Syntax

```
UMCellLocation (UMColumnIndex As Long, RowIndex As
Long, Location As CDRLocation) As Long (read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column
RowIndex:	Zero-based index of unmapped row
Location:	Location parameter

UMCellText

Description Sets / returns the text of an unmapped Table cell.

Syntax
```
UMCellText (UMColumnIndex As Long, RowIndex As
Long) As String (read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column
RowIndex:	Zero-based index of row

UMCellVisible

Description Sets / returns a Visible flag of an unmapped Table cell.

Syntax
```
UMCellVisible (UMColumnIndex As Long, RowIndex As
Long) As Boolean (read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column
RowIndex:	Zero-based index of row

UMCellWorktext

Description Sets / returns the Worktext Object of an unmapped cell.

Syntax
```
UMCellWorktext (UMColumnIndex As Long, RowIndex As
Long) As ISCBCroWorktext (read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column
RowIndex:	Zero-based index of row

UMColumnColor

Description Sets / returns the color of an unmapped column.

Syntax
```
UMColumnColor As OLE_COLOR (read/write)
```

UMColumnCount

Description Returns the number of unmapped columns.

Syntax
```
UMColumnCount As Long (read only)
```

UMColumnLabelLocation

Description Sets / returns the location of an unmapped column label.

Syntax
```
UMColumnLabelLocation (UMColumnIndex As Long,
Location As CDRLocation)
As Long (read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column
Location:	Location parameter

UMColumnLabelText

Description Sets / returns the text of label of an unmapped column.

Syntax
```
UMColumnLabelText (UMColumnIndex As Long) As String
(read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column

UMColumnLocation

Description Sets / returns the location of an unmapped column.

Syntax
```
UMColumnLocation (UMColumnIndex As Long, PageNr As
Long, Location As CDRLocation) As Long (read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column
PageNr:	DocPage number
Location:	Location parameter

UMColumnVisible

Description Sets / returns a Visible flag of an unmapped column (currently not used).

Syntax
```
UMColumnVisible (UMColumnIndex As Long) As Boolean
(read/write)
```

Parameters

UMColumnIndex:	Zero-based index of unmapped column

UnMapColumn

Description Unmaps column, i.e. transfers content of specified source

column to new unmapped column.

Syntax `UnMapColumn (Column As Variant) As Long`

Parameters *Column:* Zero-based index or name of source column

WeightingFactor

Description Sets / returns a Weighting Factor for a corresponding evaluation property.

Syntax `WeightingFactor (EvalPropIndex As Long) As Double (read/write)`

Parameters *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)

2.6 SCBCdrTextblock

2.6.1. Description

This object represents a TextBlock on a Document. A TextBlock may contain one or more lines.

2.6.2. Methods and properties

Color

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

Syntax `Color As OLE_COLOR (read/write)`

Height

Description Returns the height of the TextBlock in pixel.

Syntax `Height As Long (read only)`

Description Returns the left border of the TextBlock in pixel.

Syntax `Left As Long (read only)`

PageNr

Description Returns the number of the DocPage where the TextBlock is located.

Syntax `PageNr As Long (read only)`

Text

Description The whole text of the TextBlock is returned.

Syntax `Text As String (read only)`

Top

Description Returns the top border of the TextBlock in pixel.

Syntax `Top As Long (read only)`

Visible

Description Controls if the highlighted rectangle of the TextBlock should be visible if the TextBlock highlighting is enabled.

Syntax `Visible As Boolean (read/write)`

Weight

Description This property returns the block weight.

Syntax `Weight As Double (read only)`

Width

Description The width of the TextBlock is returned in pixel.

Syntax `Width As Long (read only)`

WordCount

Description The number of Words belonging to the TextBlock is returned.

Syntax `WordCount As Long (read only)`

WordID

Description It can be used as index for the Word array of the Workdoc.

Syntax `WordID (index As Long) As Long (read only)`

Parameters	*Index:*	Index of Word inside the TextBlock. Must be between 0 and WordCount -1

2.7 SCBCdrWord

2.7.1. Description

This object represents a textual Word of a Document.

2.7.2. Methods and Properties

Color

Description	The color that will be used for highlighting checked Words is set / returned
Syntax	`Color As OLE_COLOR (read/write)`

Height

Description	Returns the height of the Word in pixel.
Syntax	`Height As Long (read only)`

Left

Description	Returns the left border of the Word in pixel.
Syntax	`Left As Long (read only)`

PageNr

Description	Returns the number of the DocPage where the Word is located.
Syntax	`PageNr As Long (read only)`

StartPos

Description	Returns the index of the first character of the Word inside the Worktext attached to the Workdoc.
Syntax	`StartPos As Long (read only)`

Text

Description	The text of the Word is returned.
Syntax	`Text As String (read only)`

TextLen

Description The number of characters of the Word is returned.

Syntax
```
TextLen As Long (read only)
```

Tooltip

Description Sets / returns a tooltip string which will be displayed in the checked Words highlight mode

Syntax
```
Tooltip As String (read/write)
```

Top

Description Returns the top border of the Word in pixel.

Syntax
```
Top As Long (read only)
```

Visible

Description Sets / returns if the highlighted rectangle of the Word should be visible if the Word highlighting for checked Words is enabled.

Syntax
```
Visible As Boolean (read/write)
```

Width

Description Returns the width of the Word in pixel.

Syntax
```
Width As Long (read only)
```

Worktext

Description Returns the Worktext object of the Word.

Syntax
```
Worktext As ISCBCroWorktext (read only)
```

2.8 SCBCdrDocPage

2.8.1. Description

An object representing a single DocPage within a Workdoc.

2.8.2. Type Definitions

CDRPageSource

Enumeration containing the Page source.

Available Types	Description
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.

Available Types	Description
CroLinesDir_Horizontal	Horizontal line
CroLinesDir_Vertical	Vertical line

CroLinesKooType

Further information about a line.

Available Types	Description
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

2.8.3. Methods and Properties

DisplayImage

Description	Specifies the index of the Image, which should be displayed if the DocPage is visible inside a Viewer.
Syntax	`DisplayImage As Long (read/write)`

DocIndex

Description	Specifies the index of the document inside the Workdoc that this DocPage belongs to.
Syntax	`DocIndex (ImageIndex As Long) As Long (read only)`
See also	DocFileName and DocFileType property of the SCBCdrWorkdoc object

Parameters	*ImageIndex:*	ImageIndex of the DocPage. Valid indices are 0 to ImageCount-1.

DocPageIndex

Description	Specifies the DocPage offset inside the document where this DocPage belongs to.	
Syntax	`DocPageIndex (ImageIndex As Long) As Long (read only)`	
Parameters	*ImageIndex:*	Index of the Image of the DocPage. Valid indexes are 0 to ImageCount-1.

GetResolution

Description	Returns the resolution of the specified Image in pixel.	
Syntax	`GetResolution (ImageIndex As Long, pXRes As Long, pYRes As Long)`	
Parameters	*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

Description	Returns the height of the DocPage in millimeter.
Syntax	`Height As Double (read only)`

Image

Description	Returns an Image object for the specified index of the DocPage.	
Syntax	`Image (index As Long) As ISCBCroImage (read only)`	
Parameters	*Index:*	Index of the Image of the DocPage. Valid indices are 0 to ImageCount-1.

ImageCount

Description	Returns the number of Images available for the DocPage.
Syntax	`ImageCount As Long (read only)`

Line

Description Returns some specific property of line, viz. starting X ect., of some specific index and direction.

Syntax

```
Line (LineIndex As Long, LineDir As
CroLinesDir, KooType As CroLinesKooType)
As Long (read only)
```

Parameters

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

Description Returns the number of horizontal or vertical Lines present in a document.

Syntax

```
LinesCount (LinesDir As CroLinesDir) As Long
(read only)
```

Parameters

LinesDir: Direction of Line (Horizontal or Vertical).

OriginalDocumentFileName

Description This property allows the Scripter to access the page property to examine what the original file name was for the image. This could be useful for the Scripter if attempting to track original filenames for pages when a document is split/merged via Verifier / Web Verifier or via the Page Separation engine.

Syntax

```
pWorkdoc.Pages(0).OriginalDocumentFileName
```

Example

```
e As SCBCdrDocPage

Dim originalFilename As String

Set myPage = pWorkdoc.Pages(0)   'Get First page

originalFilename = myPage.OriginalDocumentFileName
MsgBox "Page - " & originalFilename
```

PageSource

Description Sets / returns a source of a DocPage. At the time of scanning, a DocPage can be directly assigned to Workdoc.

Syntax

```
PageSource As CDRPageSource (read/write)
```

Example

Rotate

Description	Rotates the underlying Images by the specified angle.	
Syntax	`Rotate (angle As Double)`	
Parameters	*Angle:*	Specifies the rotation angle in a range of -180.0 to +180.0.

Rotation

Description	Returns the rotation angle as it was applied by Rotate method.
Syntax	`Rotation As Double (read only)`

Text

Description	Returns the text of the DocPage if OCR was already executed.
Syntax	`Text As String (read only)`

Width

Description	Returns the width of the DocPage in millimeter.
Syntax	`Width As Double (read only)`

2.9 SCBCdrFolder

2.9.1. Description

A Folder may represent an array of Workdocs within a Batch. A Folder may contain one or more Workdocs. During classification and extraction it is possible to access all Workdocs of the same Folder from script.

2.9.2. Methods and Properties

AddDocument

Description	Add a Workdoc into a Folder at the last position and also returns the position where the Workdoc is appended.	
Syntax	`AddDocument (pWorkdoc As ISCBCdrWorkdoc, pNewIndex As Long)`	
Parameters	*pWorkdoc:*	[in] Added Workdoc Object

	pNewIndex:	[out] Index position in a Folder where Workdoc is inserted

Clear

Description Frees all the allocated memorie by Folder.

Syntax

```
Clear ()
```

Document

Description Returns a Workdoc from the specified index of the document array of the Folder.

Syntax

```
Document (Index As Long) As ISCBCdrWorkdoc (read only)
```

Parameters

Index:	The index of the Workdoc within the Folder. Must be from 0 to DocumentCount-1.

DocumentCount

Description The number of Workdocs within the Folder is returned.

Syntax

```
DocumentCount As Long (read only)
```

FolderData

Description Provides the possibility to store and load a variable number of strings using any string as index key.

Syntax

```
FolderData (Index As String) As String (read/write)
```

Parameters

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

Description Inserts a Workdoc into a Folder at some given position.

Syntax `InsertDocument (Index As Long, pWorkdoc As ISCBCdrWorkdoc)`

Parameters

Index:	Index at which Workdoc is to be inserted, zero-based indexing
pWorkdoc:	Workdoc object

MoveDocument

Description To move a Workdoc from one position to another position in a Folder.

Syntax `MoveDocument (FromIndex As Long, ToIndex As Long)`

Parameters

FromIndex:	Zero-based Index from where Workdoc is moved
ToIndex:	Zero-based index where Workdoc is to be placed

RemoveDocument

Description To remove a Workdoc from a given index from a Folder.

Syntax `RemoveDocument (index As Long)`

Parameters

Index:	Zero-based index in a Folder from where Workdoc is to be removed

Chapter 3 Cedar Project Object Reference (SCBCdrPROJLib)

3.1 Description

The Cedar Project object represents a complete Project definition including all Document Classes, Field Definitions, and used classification and extraction methods.

3.2 Type Definitions

CDRClassifyMode

This type defines the algorithms for how the results of several classification engines can be combined.

Available Types	Description
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.

Available Types	Description
CdrDoNothing	Let Verifier user decide to skip special processing altogether.
CdrDoSmartDecision	Make a smart decision[1], 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.

Available Types	Description
CdrForceValDefault	CdrForceValidationModeDefault: ForceValidationMode inherited

[1] The system will decide which one is the right DocClass based on an algorithm that compares the results of the associative search and the template classification. This feature can be selected from the Supervised Learning tab in Designer application.

CdrForceValForbidden	CdrForceValidationModeForbidden: ForceValidation (3*return) not allowed
CdrForceValPermitted	CdrForceValidationModePermitted: ForceValidation (3*return) allowed

CdrLicenseCounter

The data type definitions for all available license counters to be interrogated in script.

Available Types	Description
TLCFineReaderRemainingUnits	Remaining page units available to be processed by the FineReader8 licensing scheme. Integer Value = 18
TLCPeriodDocumentsClassified	Documents classified within the licensing period. Integer Value = 10
TLCPeriodDocumentsExported	Documents exported within the licensing period. Integer Value = 14
TLCPeriodDocumentsExtracted	Documents extracted within the licensing period. Integer Value = 12
TLCPeriodDocumentsOCRed	Documents OCRed within the licensing period. Integer Value = 8
TLCPeriodDocumentsProcessed	Documents processed within the licensing period. Integer Value = 2
TLCPeriodDocumentsValidatedVerifier	Documents validated in Verifier within the licensing period. Integer Value = 16
TLCPeriodPagesImported	Pages imported within the licensing period. Integer Value = 4
TLCPeriodPagesOCRed	Pages OCRed within the licensing period.

	Integer Value = 6
TLCPeriodPagesProcessed	Pages Processed within the licensing period.
	Integer Value = 0
TLCTotalDocumentsClassified	Total Overall Classified documents.
	Integer Value = 11
TLCTotalDocumentsExported	Total Overall Exported documents.
	Integer Value = 15
TLCTotalDocumentsExtracted	Total Overall Extracted documents.
	Integer Value = 13
TLCTotalDocumentsOCRed	Total Overall OCRed documents.
	Integer Value = 9
TLCTotalDocumentsProcessed	Total Overall Processed documents.
	Integer Value = 3
TLCTotalDocumentsValidatedVerifier	Total Overall documents validated in verifier.
	Integer Value = 17
TLCTotalPagesImported	Total Overall Pages Imported documents.
	Integer Value = 5
TLCTotalPagesOCRed	Total Overall Pages OCRed documents.
	Integer Value = 7
TLCTotalPagesProcessed	Total Overall Pages Processed documents.
	Integer Value = 1

CdrLicenseFeatureName

The data type definitions for all available license features to be interrogated in script.

Each data type item below is represented in the license file and may appear. If the item

appears in the license file, that the feature is licensed and available for usage.

Available Types	Description
CDRfnA2iACheckReader	The A2iA Check Reader License Feature. Integer Value = 30
CDRfnA2iAFieldReaderCustom	The A2iA Field Reader custom License Feature. Integer Value = 29
CDRfnA2iAFieldReaderSingleField	The A2iA Field Reader Single Field License Feature. Integer Value = 28
CDRfnAddressAnalysisEngine	The Address Analysis Engine License Feature. Integer Value = 57
CDRfnAddressAnalysisEngine2	The Address Analysis2 Engine License Feature. Integer Value = 58
CDRfnASSAClassifyEngine	The ASSA Classification Engine License Feature. Integer Value = 51
CDRfnAssociativeSearchEngine	The Associative Search Engine Field License Feature. Integer Value = 63
CDRfnAutomaticLearningProcessing	The Automatic Learning Processing License Feature. Integer Value = 64
CDRfnAutomaticLearningSupervising	The Learnset Manager License Feature. Integer Value = 65
CDRfnBrainwareClassifyEngine	The Brainware Classifier License Feature. Integer Value = 46
CDRfnBrainwareExtraction	The Brainware Extraction evaluation engine License Feature. Integer Value = 61
CDRfnBrainwareFieldExtraction	The Brainware Field Extraction License Feature. Integer Value = 45

CDRfnBrainwareLayoutClassification	The Brainware Layout Classifier engine License Feature. Integer Value = 54
CDRfnBrainwareTableExtraction	The Brainware Table Extraction engine License Feature. Integer Value = 60
CDRfnCairoImage	The Cairo Image License Feature. Integer Value = 21
CDRfnCairoOMR	The Cairo OMR License Feature. Integer Value = 33
CDRfnCaptureService	The Capture Service License Feature. Integer Value = 68
CDRfnCleqsBarcode	The Cleqs Barcode OCR License Feature. Integer Value = 34
CDRLfnCloseLicensingPeriodBySlaveServer	Integer Value = 9
CDRfnConcurrentVerifierSessionCount	The Web Verifier session count License Feature. Integer Value = 1
CDRfnCustomer	The customer name License Feature. Integer Value = 15
CDRfnCustomerID	The customer ID License Feature. Integer Value = 16
CDRfnDesignerDesignLicense	The Designer application module License Feature. Integer Value = 70
CDRfnDisableUpdateForVerifier	The ability to disable an update for verifier License Feature. Integer Value = 11
CDRfnEMailsImporting	The EMail Importing License Feature. Integer Value = 66
CDRfnFineReader	The FineReader4 License Feature. Integer Value = 22

CDRfnFineReader5	The FineReader5 License Feature. Integer Value = 27
CDRfnFineReader7	The FineReader7 License Feature. Integer Value = 32
CDRfnFineReader8	The FineReader8 License Feature. Integer Value = 36
CDRLfnFirmwareHDSerialNumber	The Hard Disk Serial Number License Feature. Integer Value = 13
CDRLfnFormatAnalysisEngine	The Format Analysis engine License Feature. Integer Value = 56
CDRLfnFormsClassifyEngine	The Forms Classifier engine License Feature. Integer Value = 50
CDRLfnHardwareBindingEnabled	The HW binding enabled License Feature. Integer Value = 4
CDRLfnImageSizeClassification	The Image Size classifier engine License Feature. Integer Value = 49
CDRLfnIMailBasicComponents	The Imail components License Feature. Integer Value = 69
CDRLfnISIS	The ISIS driver License Feature. Integer Value = 41
CDRLfnKadmos	The Kadmos OCR License Feature. Integer Value = 24
CDRLfnKadmos4	The Kadmos4 OCR License Feature. Integer Value = 25
CDRLfnKofax	The Kofax driver License Feature. Integer Value = 44
CDRLfnLanguageClassifyEngine	The Language Classifier Engine License Feature. Integer Value = 53

CDRLfnLicenseCountingByReprocessing	The License Counting when reprocessing documents License Feature. Integer Value = 10
CDRLfnLicenseExpirationDate	The License expiration date License Feature. Integer Value = 18
CDRLfnLicenseVersion	The License version License Feature. Integer Value = 17
CDRLfnLicensingPeriodInDays	The License period in days License Feature. Integer Value = 7
CDRLfnMasterLicenseHexID	The License HexID License Feature. Integer Value = 74
CDRLfnNonImageDocumentsProcessing	The electronic document processing License Feature. Integer Value = 67
CDRLfnNonImageDocumentsProcessing	The electronic document processing License Feature. Integer Value = 67
CDRLfnOverallVerifierSessionCount	The overall verifier session count License Feature. Integer Value = 2
CDRLfnPeriodDocumentsClassified	The documents classified count License Feature. Integer Value = 95
CDRLfnPeriodDocumentsExported	The documents exported count License Feature. Integer Value = 99
CDRLfnPeriodDocumentsExtracted	The documents extracted count License Feature. Integer Value = 97
CDRLfnPeriodDocumentsOCRed	The documents OCRed count License Feature. Integer Value = 93
CDRLfnPeriodDocumentsProcessed	The documents Processed count License Feature.

	Integer Value = 87
CDRLfnPeriodDocumentsValidatedVerifier	The documents validated in verifier License Feature. Integer Value = 101
CDRLfnPeriodPagesImported	The Pages imported License Feature. Integer Value = 89
CDRLfnPeriodPagesOCRed	The Pages OCRed License Feature. Integer Value = 91
CDRLfnPeriodPagesProcessed	The Pages Processed License Feature. Integer Value = 85
CDRLfnPhraseClassifyEngine	The Phrase Classifier engine License Feature. Integer Value = 48
CDRLfnPrimaryDongleID	The Primary Dongle ID License Feature. Integer Value = 5
CDRLfnProcessedDocumentsPerDay	The Processed Documents Per Day License Feature. Integer Value = 6
CDRLfnQualitySoftBarcode	The QualitySoft Barcode OCR engine License Feature. Integer Value = 37
CDRLfnQualitySoftBarcodeDM	The QualitySoft Barcode DM OCR engine License Feature. Integer Value = 38
CDRLfnQualitySoftBarcodePDF417	The QualitySoft Barcode PDF OCR engine License Feature. Integer Value = 39
CDRLfnRecognita	The Recognita OCR engine License Feature. Integer Value = 23
CDRLfnRecognitaBarcode	The Recognita Barcode OCR engine License Feature. Integer Value = 35
CDRLfnRecoStar	The RecoStar OCR engine License Feature.

	Integer Value = 26
CDRLfnSecondaryDongleID	The Secondary Dongle ID License Feature. Integer Value = 3
CDRLfnSecondaryHDSerialNumber	The Secondary Hard Disk Serial Number License Feature. Integer Value = 12
CDRLfnSecondaryMACAddress	The Secondary MAC Address License Feature. Integer Value = 14
CDRLfnSelfLearningManager	The Learnset Manager Module License Feature. Integer Value = 73
CDRLfnSERSCSI	The SCSI Driver License Feature. Integer Value = 40
CDRLfnServer	The RTS Server Module License Feature. Integer Value = 71
CDRLfnServerCount	The RTS Server count License Feature. Integer Value = 19
CDRLfnSupplierExtraction	The supplier extraction License Feature. Integer Value = 62
CDRLfnSVRS	The SVRS driver License Feature. Integer Value = 43
CDRLfnTableAnalysisEngine	The Table Analysis engine License Feature. Integer Value = 59
CDRLfnTemplateClassifyEngine	The Template Classifier engine License Feature. Integer Value = 47
CDRLfnTWAIN	The Twain Driver License Feature. Integer Value = 42
CDRLfnVerifier	The Verifier application module License Feature. Integer Value = 72

CDRLfnVerifierCount	The Verifier application count License Feature. Integer Value = 20
CDRLfnZoneAnalysisEngine	The Zone Analysis engine License Feature. Integer Value = 55

CdrMessageType

This type defines the different message types.

Available Types	Description
CDRTypeInfo	An informational message.
CDRTypeWarning	A warning message.
CDRTypeError	An error message.

CdrMessageSeverity

This type defines the different message severities.

Available Types	Description
CDRSeverityLogFileOnly	Store the message to the application 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. This option is applicable when the call is executed from within the Runtime Server application only.
CDRSeverityEmailNotificatio	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. This option is applicable when the call is executed from within the Runtime Server application only.

3.2.1. Methods and Properties

ActivateLicensing

Description	This method is used as a call to enable license activation in the custom script. The call is used as a prerequisite prior to retrieving information for the licensing utilization. By calling activate licensing, the script creates a connection to the active license being utilized.
Syntax	`ActivateLicensing (ModuleName as text, LicensePath as`

text)

Parameters

ModuleName: A text that represents the application activating licensing. Any value may be entered here.

LicensePath: A text that contains the location of the license share file that will be queried.

The licensePath must be accessible from the location of the script execution.

The license path must point to the Runtime.lic file explicitly.

See Also ReportLicensingStatus, GetLicenseValueByName, GetLicenseValueByID

Example Code to retrieve licensing utilization information for active licensing counters.

```
ect represents the project library object.
   Dim theProject As New SCBCdrPROJLib.SCBCdrProject


   'The location of the shared license file that is being updated.
   Dim LicenseShareLocation As String
   LicenseShareLocation="\\MasterRTS\License\Runtime.lic"


   'Activate licensing within the code for project.  This enables you
   to reference the license in the next command.
   theProject.ActivateLicensing("CustomEXE", LicenseShareLocation)
   'Call the License Reporting function, this has several options
   available
   theProject.ReportLicensingStatus(True,
   SCBCdrPROJLib.CDRMessageSeverity.CDRSeverityLogFileOnly)
```

AllClasses

Description Returns a Collection of all defined DocClasses of this Project.

Syntax `AllClasses As ISCBCdrDocClasses (read only)`

See also ISCBCdrDocClasses and ISCBCdrDocClass for further information

BaseClasses

Description Returns a Collection containing all defined BaseDocClasses.

Syntax `BaseClasses As ISCBCdrDocClasses (read only)`

See also ISCBCdrDocClasses and ISCBCdrDocClass for further information

ClassificationMode

Description Returns the used classification mode.

Syntax

```
ClassificationMode As CDRClassifyMode (read/write)
```

DefaultClassifyResult

Description Returns the default DocClass name to which a document is redirected if no other DocClass fits.

Syntax

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

DefaultLanguage

Description Returns the language used as default.

Syntax

```
DefaultLanguage As String (read only)
```

Example

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

'Set the table column to be invisible, check that the verifier form
hasn't been loaded yet.

If Reason=CdrBeforeFormLoaded Then

'The Table Setting to use to set table properties.

Dim theTableSettings As
SCBCdrBrainwareTableEngineLib.SCBCdrTableSettings

Dim theAnalysisSettings As Object

Project.AllClasses.ItemByName("Invoices").GetFieldAnalysisSettings(
"Table", Project.DefaultLanguage, theAnalysisSettings)    'Get the
table settings for the TABLE field.

Set theTableSettings = theAnalysisSettings

theTableSettings.ColumnVisible(2) = True          'Set the Column
visible to True to show, False to hide.

End If

End Sub
```

Filename

Description Returns the filename of the Project including the directory path

Syntax `Filename As String (read only)`

ForceValidation

Description 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.

Syntax `ForceValidation As CdrForceValidationMode (read/write)`

GetVerifierProject

Description Returns the *Verifier* Project.

Syntax `GetVerifierProject (ppVal As Object)`

Parameters *ppVal:* [out] Verifier Project Object

LastAddressPoolUpdate

Description Returns the time when the address pool was updated for the last time.

Syntax `LastAddressPoolUpdate As Date (read only)`

Lock

Description This property locks the Project for updating.

Syntax `Lock ()`

LogScriptMessage

Description This method enables the developer to utilize the new in-built functionality to automate custom script error notification directly to the core product logs, MMC, or system monitoring notification.

Syntax `LogScriptMessage(Type As Long, Code As Long, MessageText As String)`

Parameters

Type:	The CdrMessageType option to determine whether the message is classified to either an Information, a Warning, or an Error.
Code:	Represents the severity code of the message. Reference CdrMessageSeverity for additional information on options. This option will depict where the message will appear (Log, System Monitoring, or as an EMail).
MessageText:	The message text to display/send.

See Also CdrMessageType, CdrMessageSeverity

Example The example below writes a custom script message to the core product log file (H_RTSInstanceName).

```
   Project.LogScriptMessage(CDRTypeInfo, CDRSeverityLogFileOnly, _
sification process has been started for document " &
   pWorkdoc.Filename)
```

The above script can be placed in the PreClassify event and would provide a entry in the log similar to this:

```
[Info]   |30| 01:59:33.312 |  3108 | 668184k/1428344k |
514004k/3520792k | 57176k/67252k | 238 | 38/43 | The Classification
process has been started for document  c:\slw demo
us\batches\00000000\00000478.wdc
```

MinClassificationDistance

Description Sets / returns the minimal distance of classification results.

Syntax `MinClassificationDistance As Double (read/write)`

MinClassificationWeight

Description Sets / returns the minimal classification weight.

Syntax `MinClassificationWeight As Double (read/write)`

MinParentClsDistance

Description Sets / returns the minimal distance between the classification weight of the parent and the derived DocClasses.

Syntax `MinParentClsDistance As Double (read/write)`

MinParentClsWeight

Description Sets / returns minimal parent classification weight. This value is used as threshold during parent classification.

Syntax
```
MinParentClsWeight As Double (read/write)
```

MoveDocClass

Description Moves a DocClass specified by its Name to a new ParentDocClass specified by NewParentName.

Syntax
```
MoveDocClass (Name As String, NewParentName As String)
```

Parameters

Name:	Name of moved DocClass
NewParentName:	Name of new ParentDocClass

NoUI

Description Sets or returns NoUI. If NoUI set to true, then no login dialog is displayed.

Syntax
```
NoUI As Boolean (read/write)
```

Page

Description Returns Cairo Page object of current Project.

Syntax
```
Page As ISCBCroPage (read only)
```

ParentWindow

Description Sets the parent window of the login dialog.

Syntax
```
ParentWindow As Long (write only)
```

Parameters

lhWnd:	[in] Window handles of windows operating system.

PerformScriptCommandRTS

Description

The method allows the developer to restart, or stop, the Runtime Server via custom script.

This could be used to perform a Stop on a Runtime Server should a third party system, such as SAP, be unavailable.

The method stops the currently running Runtime Server instance executing the script to either stop or restart.

Syntax

```
PerformScriptCommandRTS (CommandID As Long,
MessageType As Long, UserCode As Long,
MessageDescription As String)
```

Parameters

CommandID: Identifier of the command to execute on the RTS instance.

Two commands that are currently supported:

. Forcing the RTS instance to stop document processing (with the "CommandID" parameter set to "0").
. Restarting the RTS instance (with the "CommandID" parameter set to "1").

MessageType: The type of message to log when the command executes: "0" for informational message, "1" for warnings and "2" for error messages. Note that error messages are additionally forwarded to MMC administration console of the Runtime Server.

UserCode: User error code of the message. This error code can be defined by the developer as any custom error number.

MessageDescription: The 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

Two examples depicting a stop and a restart of the RTS instance executing project code.

```
ript 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"
```

ReportLicensingStatus

Description

The method is used to retrieve either all license counter information, or just the active license counter information.

An active counter license is the document or page limit licensing that is present in the license file.

Reference the Product Licensing guide for further details on licensing counters present/available in the license file.

This method returns what the current utilization figures are on the server.

If running outside of the Runtime Server, the information will be saved in the U_ log file.

Syntax

```
ReportLicensingStatus (ReportActiveLicensingOnly As
Boolean, Severity As
SCBCdrPROJLib.CDRMessageSeverity)
```

Parameters

ReportActiveLicensingOnly: A Boolean flag to indicate if all licensing counters should be outputted (False), or if only the license counters active in the license file should be outputted (True).

Severity: The location of the utilization output to be sent to. This relates to the defined types shown in CdrMessageSeverity type definition (Log File, Email, or RTS System Monitoring).

An example of a log file output is:

Requested current licensing status for license "Internal" with ID 00999-D7CDV811. License updated last time at 2007-11-16 21:02:55. Current licensing period is [2] of 30 days. Project was started at 2007-10-17 15:20:31.

License status for [Processed Pages per Day = 500] (active). Current utilization: 0.65%. Units processed: 97 in period of 1 day(s). Units credit: 14903.

See Also

ActivateLicensing, GetLicenseValueByName, GetLicenseValueByID

Example

Code to retrieve licensing utilization information for all licensing counters.

```
ect represents the project library object.
   Dim theProject As New SCBCdrPROJLib.SCBCdrProject


   'The location of the shared license file that is being updated.
   Dim LicenseShareLocation As String
   LicenseShareLocation="\\MasterRTS\License\Runtime.lic"
```

```
'Activate licensing within the code for project.  This enables you
to reference the license in the next command.

theProject.ActivateLicensing("CustomEXE", LicenseShareLocation)

'Call the License Reporting function, this has several options
available

theProject.ReportLicensingStatus(False,
SCBCdrPROJLib.CDRMessageSeverity.CDRSeverityLogFileOnly)
```

ShowValidationTemplates

Description Display the validation templates and their settings in a given container.

Syntax

```
ShowValidationTemplates (pContainer As
ISCBCdrPPGContainer)
```

Parameters

pContainer: Container used to save the validation templates and their settings.

SLWDifferentResultsAction

Description Sets or returns the action to be done if a template classification and supplier extraction has different results.

Syntax

```
SLWDifferentResultsAction As
CdrSLWDifferentResultsAction (read/write)
```

SLWSupplierInvalidIfDifferentClsResults

Description Sets or returns if a Supplier Field is made invalid when the template classification and supplier extraction have different results.

Syntax

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

Unlock

Description This method unlocks the Project after updating.

Syntax

```
Unlock ()
```

UpdateAddressPool

Description To update the address analysis pool.

Syntax `UpdateAddressPool ()`

ValidationSettingsColl

Description Returns a collection of all activated validation engines.

Syntax `ValidationSettingsColl As ISCBCroCollection (read only)`

ValidationTemplates

Description Returns a collection of all available validation templates.

Syntax `ValidationTemplates As ISCBCroCollection (read only)`

VersionCount

Description Returns the number of versions available for specified filename.

Syntax `VersionCount (Filename As String) As Long (read only)`

Parameters *Filename:* Name of the file.

WordSegmentationChars

Description Sets / returns a string containing all characters used for Word segmentation.

Syntax `WordSegmentationChars As String (read/write)`

3.3 SCBCdrDocClasses

3.3.1. Description

This Collection contains all defined DocClass objects of the Cedar Project.

3.3.2. Methods and Properties

Collection

Description Returns the Collection which is internally used to store the

DocClasses.

Syntax

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

Count

Description

Returns the number of items within the Collection.

Syntax

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

IgnoreAnalysisFailures

Description

If set to 'True', any errors occuring during extraction analysis phase will be ignored. Errors will not cause a sudden termination of the extraction process. Instead, traces will be left in the component logs for the CdrProj library (at tracing level 1, i.e. Error):

0|0|13:10:14.840|LErr:0|hRes:0x80005141|cdrproj\scbcdrdocclass.cpp|Wed Sep 12 13:07:13 2012|2416|F|Error preprocessing zone ! Zone rectangle out of image.|||
0|0|13:10:14.840|LErr:0|hRes:0|cdrproj\scbcdrdocclass.cpp|Wed Sep 12 13:07:13 2012|2416||Level2||SAVINGS|

By default, this option is switched off. It can be activated at any time, for example in the PreExtract event.

Syntax

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

Example

```
' Cedar Document Class Script for Class "Level2"

Private Sub SAVINGS_PreExtract(pField As
SCBCdrPROJLib.ISCBCdrField, pWorkdoc As
SCBCdrPROJLib.ISCBCdrWorkdoc)

   pWorkdoc.NamedProperty("IgnoreAnalysisFailures") = True

End Sub
```

Item

Description

Returns a specified item from the Collection

Syntax

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

Parameters

Index: [in] The index can either be a Long value specifying the index within the collection or a String specifying the item

by name.

ItemByIndex

Description	Returns an item from the Collection specified by index.	
Syntax	`ItemByIndex (Index As Long) As ISCBCdrDocClass (read only)`	
Parameters	*Index:*	[in] Index of the item to retrieve from the Collection, valid range from 1 to Count

ItemByName

Description	Returns an item from the Collection specified by name.	
Syntax	`ItemByName (Name As String) As ISCBCdrDocClass (read only)`	
Parameters	*Name:*	[in] Name of the item to retrieve from the Collection.

ItemExists

Description	Returns TRUE if an item with specified name exists inside the Collection or FALSE is returned.	
Syntax	`ItemExists (Name As String) As Boolean`	
Parameters	*Name:*	[in] Name of item to search for.

ItemIndex

Description	The index of an item specified by name is returned.	
Syntax	`ItemIndex (Name As String) As Long (read only)`	
Parameters	*Name:*	[in] Name specifying an item in the Collection.

ItemName

Description	The name of an item specified by index is returned.

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

Parameters *Index:* [in] Index specifying an item in the Collection, valid range from 1 to Count

Tag

Description To store a variant for each item of the Collection.

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

Parameters *Index:* Specifies the item index, valid from 1 to Count

3.4 SCBCdrDocClass

3.4.1. Description

A Cedar DocClass object represents a single document class within a Cedar project class hierarchy.

3.4.2. Type Definitions

CdrFocusChangeReason

This enumeration defines the reason for the focus change of a Verifier field edit.

Available Types	Description
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

CdrVerifierClassifyReason

The reason for the classification of the document.

Available Types	Description
CdrChangedReason	The user selected a new class without leaving the classification view.
CdrInitReason	Manual classification view has just been displayed.
CdrValidatedReason	The document class has been changed.

CDRsiModule

This type defines the module in which the smart index definition should be used.

Available Types	Description
CDRsiModulePerceptive Intelligent Capture	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.

Available Types	Description
CdrForceValDefault	CdrForceValidationModeDefault: ForceValidationMode inherited
CdrForceValForbidden	CdrForceValidationModeForbidden: ForceValidation (3*return) not allowed
CdrForceValPermitted	CdrForceValidationModePermitted: ForceValidation (3*return) allowed

3.4.3. Methods and Properties

ClassificationField

Description	To read or to write the name of the field that is used for the classification.
Syntax	`ClassificationField As String (read/write)`

ClassificationRedirection

Description Returns the name of the target DocClass.

Syntax
```
ClassificationRedirection As String (read/write)
```

ClassifySettings

Description Collection of chosen classification engines and their settings for this DocClass.

Syntax
```
ClassifySettings As ISCBCroCollection (read only)
```

DerivedDocClasses

Description Returns a collection of all DocClasses derived directly from this DocClass.

Syntax
```
DerivedDocClasses As ISCBCdrDocClasses (read only)
```

DisplayName

Description Sets / returns the display name of the DocClass currently not used, if nothing inserted here the DocClass name are used.

Syntax
```
DisplayName As String (read/write)
```

Fields

Description Provides access to FieldDefs of a DocClass.

Syntax
```
Fields As ISCBCdrFieldDefs (read only)
```

ForceSubtreeClassification

Description Sets / returns if the classification to the sub tree of this DocClass is forced.

Syntax
```
ForceSubtreeClassification As Boolean (read/write)
```

ForceValidation

Description 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

Syntax

```
ForceValidation As CdrForceValidationMode (read/write)
```

GetFieldAnalysisSettings

Description Returns the analysis settings for the document class.

Syntax

```
GetFieldAnalysisSettings (FieldName As String, Language
As String, ppAnalysisSettings As
ISCBCdrAnalysisSettings)
```

Parameters

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

Description Specifies if the DocClass should be visible in the Project designer.

Syntax

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

InitField

Description Reinitializes a required field in workdoc.

Syntax
```
InitField (pWorkdoc As ISCBCdrWorkdoc, pField As ISCBCdrField)
```

Parameters

pWorkdoc: Current workdoc.

pField: Field to be cleared.

ManualTableTrainingMode

Description Sets or returns the option for manual Table Extraction training mode

Syntax
```
ManualTableTrainingMode As Boolean (read/write)
```

Name

Description Reads or writes the name of the Document Class.

Syntax
```
Name As String (read/write)
```

Page

Description Returns the Page object of this DocClass with all defined zones and their OCR settings.

Syntax
```
Page As ISCBCroPage (read only)
```

Parent

Description Returns the parent DocClass of the actual DocClass.

Syntax
```
Parent As ISCBCdrDocClass (read only)
```

ShowClassValidationDlg

Description Displays the property page of validation settings for this document class.

Syntax `ShowClassValidationDlg (pContainer As ISCBCdrPPGContainer)`

Parameters

pContainer:	Container in which the property page should be displayed.

ShowFieldValidationDlg

Description Displays the property page of the validation settings for the specified field name.

Syntax `ShowFieldValidationDlg (FieldName As String, pContainer As ISCBCdrPPGContainer)`

Parameters

FieldName:	Field for which the dialog is shown.
pContainer:	Container in which the property page should be displayed.

ShowGeneralFieldPPG

Description Starts field settings property page specifying the active tab

Syntax

```
ShowGeneralFieldPPG (FieldName As String,
TabIndexActive As Long,
pContainer As ISCBCdrPPGContainer)
```

Parameters

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

Description Returns the minimal distance to the classification weight of the derived DocClasses.

Syntax `SubtreeClsMinDist As Double (read/write)`

SubtreeClsMinWeight

Description Sets / returns the minimal classification weight of the derived DocClasses.

Syntax `SubtreeClsMinWeight As Double (read/write)`

UseDerivedValidation

Description Sets or returns a Boolean value, when derived validation rules are used.

Syntax `UseDerivedValidation As Boolean (read/write)`

ValidationSettingsColl

Description Returns a collection of all activated validation engines.

Syntax `ValidationSettingsColl As ISCBCroCollection (read only)`

ValidationTemplateName

Description Sets or returns the name of the validation template.

Syntax `ValidationTemplateName As String (read/write)`

ValidClassificationResult

Description Sets / returns if this DocClass is a valid classification result or if it is omitted for classification.

Syntax `ValidClassificationResult As Boolean (read/write)`

VisibleInCorrection

Description This property determines if a project class is available for classification correction.

In version 4.x, 5.2, and 5.3 this property was read only.

In version 5.3 SP1 and above, this property can be modified prior to classification correction for a Verifier.

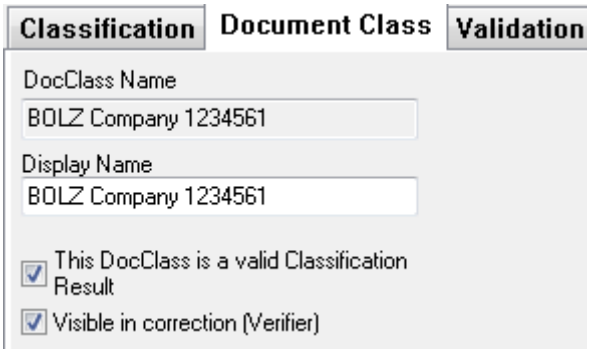
Setting the property to

- **True**: the class is available for classification correction.
- **False**: the class is unavailable for classification correction.

Dynamic modification of this property can be managed through

the ScriptModule_VerifierClassify event.

Dynamic modification of the class visibility overrides the default Designer class property.



Attribute `Read/write`

Syntax `VisibleInCorrection As Boolean (read/write)`

Example The script sample below shows how to dynamically modify the property of classes prior to showing the classification view.

The example below hides Invoices, BOLZ and UNICOM classes from verification availability.

```
Public Function fnShouldHideClass(ByVal strClassNameToCheck As String, pWorkdoc As
           SCBCdrPROJLib.SCBCdrWorkdoc) As Boolean
   Select Case UCase (strClassNameToCheck)
      Case "BOLZ COMPANY 1234561"
         fnShouldHideClass = False
      Case "UNICOM CORPORATION 1234563"
         fnShouldHideClass = False
      Case "INVOICES"
         fnShouldHideClass = False
      Case Else
         fnShouldHideClass = True
   End Select
End Function

Private Sub ScriptModule_VerifierClassify(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc, ByVal
Reason As SCBCdrPROJLib.CdrVerifierClassifyReason, ClassName As String)
   Dim i As Long
   Dim strNextClassName As String
   If Reason = CdrInitReason Then
      For i = 1 To Project.AllClasses.Count Step 1
         strNextClassName = Project.AllClasses.ItemName(i)
         Project.AllClasses.ItemByIndex(i).VisibleInCorrection =
fnShouldHideClass(strNextClassName, pWorkdoc)
      Next i
   End If

End Sub
```

FillRectangle

Description Allows the developer to draw a square on the image (White/Black) which can be used to blank out a certain area on the invoice.

By utilizing the FillRectangle method of the SCBCroImage object, we can perform image redaction

Parameters

Color to use	0: black , 1: white

Left, Top, Width, Height: Dimensions of the rectangle

3.5 SCBCdrFieldDefs

3.5.1. Description

This Collection contains all defined FieldDef objects of a single DocClass.

3.5.2. Methods and Properties

Collection

Description Returns the Collection which is internally used to store the FieldDefs.

Syntax

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

Count

Description Returns the number of items within the FieldDef Collection.

Syntax

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

Item

Description Returns a specified item from the Collection.

Syntax

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

Parameters

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

Description Returns an item from the Collection specified by index.

Syntax

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

Parameters

Index: [in] Index of the item to retrieve from the Collection.

ItemByName

Description Returns an item from the Collection specified by name.

Syntax
```
ItemByName (Name As String) As ISCBCdrFieldDef
(read only)
```

Parameters

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

ItemExists

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

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

Parameters

Name:	[in] Name of item to search for.

ItemIndex

Description The index of an item specified by name is returned.

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

Parameters

Name:	[in] Name specifying an item in the Collection.

ItemName

Description The name of an item specified by index is returned.

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

Parameters

Index:	[in] Index specifying an item in the Collection, valid range from 1 to Count

Tag

Description To store a variant for each item of the Collection.

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

Parameters

Index:	Specifies the item index, valid range from 1 to Count

3.6 SCBCdrFieldDef

3.6.1. Description

A Cedar FieldDef object represents the definition of a single FieldDef inside a Cedar DocClass

3.6.2. Type Definitions

CdrFieldFormat

This type defines the default format of a certain field. (Not yet implemented)

Available Types	Description
CdrFieldFormatCurrency	CdrFieldFormatCurrency
CdrFieldFormatDate	CdrFieldFormatDate
CdrFieldFormatExtNumber	CdrFieldFormatExtNumber
CdrFieldFormatNone	CdrFieldFormatNone
CdrFieldFormatNumber	CdrFieldFormatNumber

CDRFieldType

This type defines the type of a FieldDef.

Available Types	Description
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.

Available Types	Description
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.

Available Types	Description
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.

3.6.3. Methods and Properties

AlwaysValid

Description Sets / returns if the content of this FieldDef is always valid.

Syntax
```
AlwaysValid As Boolean (read/write)
```

AnalysisTemplate

Description Returns the name of the analysis template if used.

Syntax
```
AnalysisTemplate (Language As String) As String (read only)
```

Parameters *Language:* Language parameter

AppendListItem

Description Adds a new list item and returns a new item index for it.

Syntax
```
AppendListItem (bstrItem As String) As Long
```

Parameters *bstrItem:* String inserted into the list.

ColumnCount

Description Returns the number of Table columns if FieldType is Table.

Syntax `ColumnCount As Long (read only)`

ColumnName

Description Returns the name of a Table column if FieldType is Table.

Syntax `ColumnName (ColumnIndex As Long) As String (read only)`

Parameters *ColumnIndex:* Zero-based index of the Table column

DefaultValidationSettings

Description Returns the validation settings with default language.

Syntax `DefaultValidationSettings As ISCBCdrValidationSettings (read only)`

Parameters *ppValSettings:* ValidationSettings object for the default language

Derived

Description Returns TRUE if the FieldDef properties are derived from an upper DocClass.

Syntax `Derived As Boolean (read only)`

DisplayName

Description 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.

Syntax `DisplayName As String (read/write)`

EvalSetting

Description Sets / returns activated evaluation engine and its settings.

Syntax `EvalSetting (Language As String) As Object (read/write)`

Parameters *Language:* Language parameter

EvalTemplate

Description Returns the name of the evaluation template if used.

Syntax
```
EvalTemplate (Language As String) As String (read only)
```

Parameters *Language:* Language of Project

FieldID

Description This read-only property returns the internally used FieldID.

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

FieldType

Description Sets or returns the type of the FieldDef.

Syntax
```
FieldType As CDRFieldType (read/write)
```

ForceValidation

Description Sets or returns the mode for the ForceValidation.

Syntax
```
ForceValidation As CdrForceValidationMode (read/write)
```

ListItem

Description Sets or returns a list item string for a given index.

Syntax
```
ListItem (lIndex As Long) As String (read/write)
```

Parameters *lIndex:* Zero-based index.

ListItemCount

Description Returns the number of strings in the ListItem list.

Syntax `ListItemCount As Long (read only)`

Example

```
Dim lngItem As Long

For lngItem =
Project.AllClasses.ItemByName("Invoice").Fields("Currency").ListIte
mCount - 1 To 0 Step -1
```

MaxLength

Description Returns the maximal number of characters permitted for this FieldDef.

Syntax `MaxLength As Long (read/write)`

MinLength

Description Sets / returns the minimal number of characters for this FieldDef.

Syntax `MinLength As Long (read/write)`

Name

Description Sets / returns the name of the FieldDef.

Syntax `Name As String (read/write)`

NoRejects

Description Sets / returns if rejects are permitted.

Syntax `NoRejects As Boolean (read/write)`

OCRConfidence

Description Sets / returns the confidence level for OCR. The value must be between 0 and 100.

Syntax `OCRConfidence As Long (read/write)`

RemoveListItem

Description Removes a list item by its index.

Syntax `RemoveListItem (lIndex As Long)`

Parameters *lIndex:* Index of entry to be removed from the list.

Example

```
Project.AllClasses.ItemByName("Invoice").Fields("Currency").RemoveListItem(lngItem)
```

SmartIndex

Description Contains all definitions about smart indexing.

Syntax `SmartIndex As ISCBCdrSmartIndex (read/write)`

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
```

UseDerivedOCRSettings

Description Sets / returns if OCR settings of the parent DocClass are used.

Syntax `UseDerivedOCRSettings As Boolean (read/write)`

UseDerivedValidation

Description Sets / returns if the derived validation rules are used for validation of this FieldDef.

Syntax `UseDerivedValidation As Boolean (read/write)`

UseMaxLen

Description Sets / returns if the maximal number of characters is limited to the value given by MaxLength.

Syntax `UseMaxLen As Boolean (read/write)`

UseMinLen

Description Sets / returns if the usage of the minimal number of characters given by the property MinLength is activated.

Syntax

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

ValidationSettings

Description Sets or returns the chosen validation engine and its settings.

Syntax

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

Parameters

Language: Defines the language for classification, extraction and validation.

ValidationTemplate

Description Returns the name of validation template.

Syntax

```
ValidationTemplate (Language As String) As String
(read only)
```

Parameters

Language: Defines the language for classification, extraction and validation.

ValidationType

Description Returns the type of validation.

Syntax

```
ValidationType As CdrValFieldType (read only)
```

VerifierColumnWidth

Description Sets /returns the width of the specified column of the Table.

Syntax

```
VerifierColumnWidth (ColumnIndex As Long) As Long
(read/write)
```

Parameters

ColumnIndex: Zero-based Index of the Table column

3.7 SCBCdrSettings

3.7.1. Description

The Cedar Settings object stores arbitrary strings for usage in script.

3.7.2. Methods and Properties

ActiveClient

Description Sets / returns name of the currently active client.

Syntax
```
ActiveClient As String (read/write)
```

AddClient

Description Adds a new client with the specified name to the current Settings object.

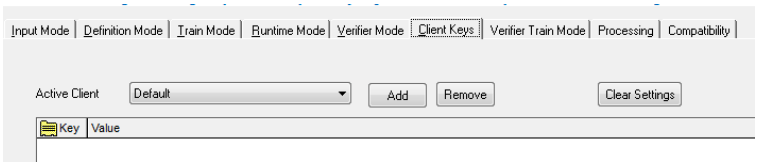
Syntax
```
AddClient (newVal As String)
```

Parameters

newVal:	Name of new client

AddKey

Description Adds a new key specified by its name and its Parent. Refer to Designer User Guide for more information.



Syntax
```
AddKey (newVal As String, Parent As String)
```

Parameters

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

Description Clears all clients and keys from the Settings object.

Syntax
```
Clear ()
```

Client

Description Returns the name of the specified client.

Syntax
```
Client (Index As Long) As String (read only)
```

Parameters

Index:	Zero-based client index.

ClientCount

Description Returns the number of clients.

Syntax `ClientCount As Long (read only)`

GlobalLearnsetPath

Description Sets or returns the global Learnset path.

Syntax `GlobalLearnsetPath As String (read/write)`

Key

Description Returns the key name specified by index.

Syntax `Key (Index As Long) As String (read only)`

Parameters

Index:	Zero-based index of the key.

KeyCount

Description Returns the number of keys.

Syntax `KeyCount As Long (read only)`

KeyIcon

Description Sets new value for the specified key or returns the key's value.

Syntax `KeyIcon (Key As String) As String (read/write)`

Parameters

Key:	Name of the key.

KeyParent

Description Returns the parent name of specified key index.

Syntax `KeyParent (Index As Long) As String (read only)`

Parameters

Index:	Zero-based key index.

MoveKey

Description Moves a key specified by its name to the NewParent specified by its name.

Syntax `MoveKey (Key As String, NewParent As String)`

Parameters

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

Description Sets or returns the file name of the Project.

Syntax
```
ProjectFileName As String (read/write)
```

RemoveClient

Description Removes a client specified by its name.

Syntax
```
RemoveClient (ClientName As String)
```

Parameters *ClientName:* Name of client that should be removed

RemoveKey

Description Removes a key specified by its name.

Syntax
```
RemoveKey (KeyName As String)
```

Parameters *KeyName:* Name of key that is removed.

SupervisedLearningDisabled

Description Sets or returns the state of supervised learning in *Designer* and local *Verifier* workstations.

Syntax
```
SupervisedLearningDisabled As Boolean (read/write)
```

TopDownEventSequence

Description Sets or returns the value of top-down event sequence.

Syntax
```
TopDownEventSequence As Boolean (read/write)
```

Value

Description Returns the value of the specified key.

Syntax
```
Value (Key As String, Parent As String, Client As
String) As String (read/write)
```

Parameters

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, …)
```

3.8 SCBCdrScriptModule

3.8.1. Description

This is a global object at the project level. All script module events occurred at project level belongs to this object.

3.8.2. Methods and Properties

ModuleName

Description

Returns the name of the module that initialized ScriptModule.

The full list of values and under what circumstances they are set are detailed below:

Runtime Server - ModuleName = Server

Web Verifier Client (v5 and above) - ModuleName = Verifier

Verifier Thick Client (v3 and above) - ModuleName = Verifier

Local Verifier Project - ModuleName = LocalVerifier

Learnset Manager Tool - ModuleName = PlainVerifier

Designer Runtime mode = Server

Designer Verifier test mode = Verifier

Designer Verifier train mode = Verifier

Designer Normal train mode = Designer

Designer Definition mode = Designer

Syntax

```
ModuleName As String (read only)
```

Example

```
‘This example sets the global variable gblVerifierAsServer to true
if the Modulename contains VERIFIER
Private Sub Document_PreExtract(pWorkdoc As
SCBCdrPROJLib.SCBCdrWorkdoc)
If InStr(UCase(ScriptModule.ModuleName), “VERIFIER”) Then
gblVerifierAsServer = True
Else
gblVerifierAsServer = False
end if
End Sub


‘This example is a function which returns true if the Modulename
contains VERIFIER
Public Function fnIsVerifier As Boolean
If InStr(UCase(ScriptModule.ModuleName), “VERIFIER”) Then
fnIsVerifier = True
Else
fnIsVerifier = False
end if
End Function
```

ReadZone

Description This method can be used to read a zone on a CroImage object, which settings are saved before in the ScanJobs' definition.

Syntax

```
ReadZone (Image As ISCBCroImage, ZoneName As String)
As String
```

Parameters

Image:	[in] SCBCroImage object
ZoneName:	[in] Name of Zone which is read

ReadZoneEx

Description This method can be used to read a zone on a CroImage object, which settings are saved before in the ScanJobs' definition.

Syntax

```
ReadZoneEx (Image As ISCBCroImage, ZoneName As String,
Result As ISCBCroWorktext)
```

Parameters

Image:	[in] SCBCroImage object
ZoneName:	[in] Name of read zone
Result:	[in] Result of reading returned as SCBCroWorktext object

3.9 SCBCdrScriptProject

3.9.1. Description

3.9.2. Methods and Properties

CurrentClient

Description This property retrieves and sets the "Client" attribute of the batch.

Syntax

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

GetHostProperties

Description The method lets user retrieve information about the current machine, application and Perceptive Intelligent Capture user.

Syntax

```
GetHostProperties(appType as CDRApplicationName,
appSubtype as Long, appInstance as String, appUserName
as String, appIP as String, appMachineName as String,
appLicensee as String)
```

Parameters

appType	Applicationname represents the calling

	application by a CDRApplicationName type. The parameter can be read from script. CDRApplicationName: *TANDesigner:* - represents Perceptive Intelligent Capture Designer *TANLearnSetManager:* - represents Perceptive Intelligent Capture Learn Set Manager *TANLocalVerifier:* - represents Perceptive Intelligent Capture Verifier used as local project for SLW *TANRuntimeServer:* - represents Perceptive Intelligent Capture Runtime Service Instance *TANUnknown:* - unknown application *TANVerifier:* - represents Perceptive Intelligent Capture Verifier *TANWebVerifier:* - represents Perceptive Intelligent Capture Web Verifier
appSubType	Only used for internal purposes
appInstance	The name of the Perceptive Intelligent Capture Runtime Service Instancename, if ApplicationName is TANRuntimeServer. Not used for other applications.
appUsername	Login Name of the current Perceptive Intelligent Capture user Perceptive Intelligent Capture user for Designer, Verifier, LSM, Web Verifier Windows user for Runtime Server
appIP	IP address of the computer
appMachineName	Machine name that is running the script
appLicensee	Customer name of the used license file

Example

The script below calls the GetHostProperties in the initialize event. The method than returns information into variables as to where the script is executed, who is executing it, and what application module is executing it.

```
Private Sub ScriptModule_Initialize(ByVal ModuleName As String)
```

```
        Dim appInstance As String
        Dim appSubtype As Long
        Dim appUserName As String
        Dim appIP As String
        Dim appMachineName As String
        Dim appLicensee As String
        Dim appType As CDRApplicationName

        Project.GetHostProperties(appType, appSubtype, appInstance,
appUserName, appIP, appMachineName, appLicensee)

End Sub
```

3.10 SCBCdrScriptAccess

3.10.1. Description

Perceptive Intelligent Capture provides a new 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 Perceptive Intelligent Capture custom script. Using this interface it is possible to retrieve, modify and dump project and class level scripts.

3.10.2. Methods and Properties

DumpAllPages

Description Dumps all script pages available in the project as a Unicode text file.

Syntax

```
DumpAllPages(FileName As String)
```

Parameters

FileName:	[in] name of the dump file.

Example

```
ccess.DumpAllPages("Script Export_" & CStr(Format(Now, "DDMMYYYY
HHMM")) & ".sax")     'Export all script pages prior to modification
(Project and Classes).
```

ExportAllPages

Description CURRENTLY NOT SUPPORTED. Exports all available script pages in a reimportable format to the specified folder.

Syntax

```
ExportAllPages(FolderName As String)
```

Parameters

FolderName:	[in] name of the folder to save the script pages to.

ExportClassPage

Description CURRENTLY NOT SUPPORTED. Exports the specified script page to a script dump file.

Syntax
```
ExportClassPage(FolderName As String, ClassName As String)
```

Parameters

FolderName:	[in] name of the folder to save the script page to.
ClassName:	[in] name of the class to export the script for.

GetPageCode

Description Retrieves the project or specified class level script code.

Syntax
```
GetPageCode(ClassName As String, ScriptCode As String)
```

Parameters

ClassName:	[in] name of the class.
ScriptCode:	[out] class script code.

ImportAllPages

Description CURRENTLY NOT SUPPORTED. Imports all available script pages using script dumps from the specified folder.

Syntax
```
ImportAllPages(FolderName As String)
```

Parameters

FolderName:	[in] name of the folder to load the script pages from.

ImportClassPage

Description CURRENTLY NOT SUPPORTED. Imports the specified script page from a script dump file.

Syntax
```
ImportClassPage(FolderName As String, ClassName As String)
```

Parameters

FolderName:	[in] name of the folder to load the script page from.
ClassName:	[in] name of the class to import the script for.

SetPageCode

Description Assigns the project or specified class level script code.

Syntax
```
SetPageCode(ClassName As String, ScriptCode As String)
```

Parameters

ClassName:	[in] name of the class.

ScriptCode: [out] class script code.

Example

```
theScriptAccess.SetPageCode(strClassName, "")   'Set new script code
(blank “”)
```

Chapter 4 (CDRADSLib)

4.1 SCBCdrSupExSettings

4.1.1. Description

This collection contains the functions for the Associative Search engine.

4.1.2. Methods and Properties

ClearFilterAttributes

Description Clears all existing filters of the Multi-columnn Attribute Search.

Syntax

```
.ClearFilterAttributes()
```

Example

```
Dim theSupplierSettings As Object
Set theSupplierSettings = FieldAnalysissettings
Dim theAdsSettings As CDRADSLib.SCBCdrSupExSettings
Set theAdsSettings = theSupplierSettings

theAdsSettings.ClearFilterAttributes
```

AddFilterAttributes

Description Adds new filters for a chosen attribute of the Multi-column Attribute search. Choose attributes from the data source of the Associative Search Engine.

Note: The first two attributes are combined as logical OR, and the additional ones that may be added are combined with logical AND.

Syntax

```
.AddFilterAttribute("Attribute Name", "Attribute Value")
```

Parameters

Attribute Name:	*Name of the attribute to be filtered*
Attribute Value	*Value of the attribute that is searched for in the datasource*

Example This example configures the Multi-column Attribute Search for use with the Vendor search button of the Verifier Thick Client. The VendorSearch button in Verifier is related to the Object: General, Process: DialogFunc:

```
Dim theSupplierSettings As Object
Set theSupplierSettings = FieldAnalysissettings
Dim theAdsSettings As CDRADSLib.SCBCdrSupExSettings
Set theAdsSettings = theSupplierSettings

theAdsSettings.ClearFilterAttributes
theAdsSettings.AddFilterAttribute "SupplierName", "VAN"
theAdsSettings.AddFilterAttribute "SupplierName", "VAN3"
```

The following example configures the extension for the filtering with RTS in the VendorName (or VendorASSA) object preExtract event:

```
Private Sub VendorName_PreExtract(pField As
SCBCdrPROJLib.SCBCdrField, pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)


   Dim theSupplierSettings As CDRADSLib.SCBCdrSupExSettings

   Dim theDocClass As SCBCdrDocClass

   Dim theAnalysisSettings As ISCBCdrAnalysisSettings

   Dim theObject As Object

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


   theDocClass.GetFieldAnalysisSettings "VendorName","German",
theAnalysisSettings

   Set theObject = theAnalysisSettings

   Set theSupplierSettings = theObject


   theSupplierSettings.ClearFilterAttributes()

   theSupplierSettings.AddFilterAttribute "SupplierName", "VAN"

   theSupplierSettings.AddFilterAttribute "SupplierName", "VAN3"


End Sub
```

Chapter 5 Analysis Engines Object Reference

5.1 SCBCdrAssociativeDbExtractionSettings

5.1.1. Description

This interface covers all methods and properties that are required for controlling and accessing the new universal format of the ASSA engine's pool.

5.1.2. Type Definitions

CdrAutoUpdateType

This enumeration is used to specify the automatic import property.

Available Types	Description
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.

5.1.3. Method and Properties

AddColumn

Description Adds a new column field to the pool.

Syntax

```
AddColumn (ColumnName As String, IsSearchField As Boolean, NewColumnIndex As Long)
```

Parameters

ColumnName:	[in] Name of the column field.
IsSearchField:	[in] Boolean value that has 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

Description Appends a new phrase to the list of phrases to be used for the address analysis.

Syntax

```
AddPhrase (Phrase As String, IsIncludePhrase As Boolean)
```

Parameters

Phrase:	[in] This string variable 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

Description Updates or inserts the content of the entry data to the specified column.

Syntax `ChangeEntry (ColumnName As String, EntryData As String)`

Parameters

ColumnName:	[in] Name of the column that is changed.
EntryData:	[in] The content of the specified column is updated with this data.

ClassNameFormat

Description Sets or reads the format definition for a document class name.

Syntax `ClassNameFormat As String (read/write)`

ColumnCount

Description Returns the number of columns of currently opened pool.

Syntax `ColumnCount As Long (read only)`

ColumnName

Description Returns or sets the name of the column by its index.

Syntax `ColumnName (ColumnIndex As Long) As String (read/write)`

Parameters

ColumnIndex:	[in] Index of the column to retrieve [zero-based].

CommitAddEntry

Description After execution of StartAddEntry and ChangeEntry changes take effect.

Use this method only in context with the StartUpdate, StartAddEntry, ChangeEntry, Com mitAddEntry and CommitUpdate.

Syntax `CommitAddEntry (NewIndex As Long)`

Parameters

NewIndex:	[out] Index of new entry.

CommitUpdate

Description Closes and saves the currently opened pool.

Syntax `CommitUpdate ()`

EnableCandidateEvaluation

Description Sets / returns if candidate evaluation permitted.

Syntax `EnableCandidateEvaluation As Boolean (read/write)`

EntryCount

Description Returns the number of entries of the pool.

Syntax `EntryCount As Long (read only)`

EvalFirstPageOnly

Description Sets / returns if candidate evaluation is processed only for the first page.

Syntax `EvalFirstPageOnly As Boolean (read/write)`

FieldContentsFormat

Description Sets / returns the format definition for the representation of the engine results.

Syntax `FieldContentsFormat As String (read/write)`

FindLocation

Description Sets / returns if address analysis is enabled. If TRUE the position of the address is found.

Syntax `FindLocation As Boolean (read/write)`

GeneratePool

Description Imports the pool from specified source by the property AutomaticImportMethod.

Syntax `GeneratePool ()`

GeneratePoolFromCsvFile

Description Removes the previous pool and generates a new one using CSV file designed in the new format.

Syntax `GeneratePoolFromCsvFile ()`

GeneratePoolFromODBC

Description Removes previous pool and generates a new one using ODBC source using the parameters set on the property page.

Syntax
```
GeneratePoolFromODBC ()
```

GetClassNameByID

Description Returns the formatted document class name for the pool entry specified by its unique ID.

Syntax
```
GetClassNameByID (IDHigh As Long, IDLow As Long, ClassName As String)
```

Parameters

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

Description Returns the content of a field that is specified by its index and the column name.

Syntax
```
GetEntry (Index As Long, FieldName As String) As String
```

Parameters

Index:	[in] Index of the entry to be retrieved.
FieldName:	[in] Name of the column to be retrieved.

GetFormattedValueByID

Description Returns the formatted entry representation for the pool entry specified by its unique ID.

Syntax
```
GetFormattedValueByID (IDHigh As Long, IDLow As Long, FormattedValue As String)
```

Parameters

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

Description Returns unique ID of an entry by index.

Syntax
```
GetIDByIndex (Index As Long, IDHigh As Long, IDLow As Long)
```

Parameters

Index:	[in] Zero-based index.

	IDHigh:	[out] Upper part of 64-bit unique ID.
	IDLow:	[out] Lower part of 64-bit unique ID.

GetIndexByID

Description Returns index of an entry by its unique ID.

Syntax
```
GetIndexByID (IDHigh As Long, IDLow As Long, Index As Long)
```

Parameters	*IDHigh:*	[in] Upper part of 64-bit unique ID.
	IDLow:	[in] Lower part of 64-bit unique ID.
	Index:	[out] Zero-based index.

GetSearchArea

Description Returns area on the document to search in

Syntax
```
GetSearchArea (SearchAreaIndex As Long, Left As Long, Top As Long, Width As Long, Height As Long)
```

Parameters	*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

Description Sets /returns the name of column of unique ID.

Syntax
```
IdentityColumn As String (read/write)
```

ImportFieldNames

Description Sets / returns if column names are taken from first line of CSV file.

Syntax
```
ImportFieldNames As Boolean (read/write)
```

ImportFileName

Description Sets / returns the name of CSV file that should be imported.

Syntax
```
ImportFileName As String (read/write)
```

ImportFileNameRelative

Description Sets / returns if the name of CSV file should be stored relative to the path of project file.

Syntax `ImportFileNameRelative As Boolean (read/write)`

IsPhraseIncluded

Description Sets / returns if a phrase to find address is sufficient.

Syntax `IsPhraseIncluded (PhraseIndex As Long) As Boolean (read/write)`

Parameters *PhraseIndex:* [in] Index of phrase [zero-based].

IsSearchField

Description Sets / returns if a field is used for associative search.

Syntax `IsSearchField (ColumnIndex As Long) As Boolean (read/write)`

Parameters *ColumnIndex:* [in] Index of column [zero-based]

LastImportTimeStamp

Description Returns the timestamp for the last import.

Syntax `LastImportTimeStamp As Date (read only)`

MaxCandidates

Description Sets / returns the maximum number of results of the associative search engine.

Syntax `MaxCandidates As Long (read/write)`

MinDistance

Description Sets / returns the required minimum distance to next best candidate for a valid result.

Syntax `MinDistance As Double (read/write)`

MinRelevance

Description This property sets or returns the minimum relevance for search results, default value is 0.0.

Syntax `MinRelevance As Double (read/write)`

MinThreshold

Description Sets / returns the required minimum value for a valid engine result.

Syntax `MinThreshold As Double (read/write)`

ODBCName

Description This property sets / returns the name of the ODBC source.

Syntax `ODBCName As String (read/write)`

Password

Description Sets / returns the password of the ODBC source.

Syntax `Password As String (read/write)`

Phrase

Description Sets / returns phrase by its index.

Syntax `Phrase (PhraseIndex As Long) As String (read/write)`

PhrasesCount

Description Returns the number of phrases used for address analysis.

Syntax `PhrasesCount As Long (read only)`

PoolName

Description Sets / returns the name of the associative search pool.

Syntax `PoolName As String (read/write)`

PoolPath

Description Sets / returns the name of path of the associative search pool.

Syntax `PoolPath As String (read/write)`

PoolPathRelative

Description Sets / returns if the pool should be saved relative to the path of the project.

Syntax `PoolPathRelative As Boolean (read/write)`

ProjectPath

Description Returns the path of the project file.

Syntax `ProjectPath As String (read only)`

RemovePhrase

Description Removes a phrase from list of phrases for address analysis specified by its index.

Syntax
```
RemovePhrase (PhraseIndex As Long)
```

Parameters

PhraseIndex:	[in] Index of the phrase that should be deleted [zero-based].

SavePoolInternal

Description Sets / returns if pool should be saved within the project file or as separate files.

Syntax
```
SavePoolInternal As Boolean (read/write)
```

Separator

Description Sets / returns separator, either semicolon or comma, used for csv file.

Syntax
```
Separator As String (read/write)
```

SetSearchArea

Description Sets area on the document to search in.

Syntax
```
SetSearchArea (SearchAreaIndex As Long, Left As Long,
Top As Long,
Width As Long, Height As Long)
```

Parameters

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

Description Sets /returns an SQL statement used to import ODBC source.

Syntax
```
SQLQuery As String (read/write)
```

StartAddEntry

Description Prepares the insertion of a new entry to the associative search pool.

Syntax `StartAddEntry ()`

StartUpdate

Description Generates and opens a new empty pool, or opens an existing pool for the update.

Syntax `StartUpdate (RemoveExistingPool As Boolean)`

Parameters

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 the “AddColumn” method is invoked, the “AddColumn” method will report an error because it must be prohibited to modify the existing column.

Username

Description Sets / returns username required for the login into the ODBC source.

Syntax `Username As String (read/write)`

VendorTypeColumn

Description Sets / returns the column that defines the vendor type. The vendorType column must contain a value in the area of 0-2. 0 means that no class can be created for that vendor via SLW. 1 Allows one document for that vendor to be trained, while 2 allows unlimited training.

Syntax `VendorTypeColumn As String (read/write)`

Chapter 6 StringComp Object Reference (SCBCdrSTRCOMPLib)

6.1 SCBCdrStringComp

6.1.1. Description

This component provides several implementations of string compare algorithms.

6.1.2. Type Definitions

CdrCompareType

String Compare Algorithm

Available Types	Description
CdrTypeLevenShtein	Levenshtein algorithm
CdrTypeRegularExpression	Regular expression
CdrTypeSimpleExpression	Simple expression
CdrTypeStringComp	Exact string compare
CdrTypeTrigram	Trigram algorithm

6.1.3. Methods and Properties

CaseSensitive

Description This option controls if the compare algorithm should work case sensitive.

Syntax

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

CompType

Description Selects the compare algorithm used for the next call of Distance.

Syntax

```
CompType As CdrCompareType (read/write)
```

Distance

Description 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.

Syntax

```
Distance (String As String, Distance As Double)
```

Parameters

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

Description Returns the count of deletions calculated by the last Distance function.

Syntax

```
LevDeletions As Single (read only)
```

LevInsertions

Description Returns the count of insertions calculated by the last Distance function.

Syntax

```
LevInsertions As Single (read only)
```

LevRejects

Description Returns the count of rejects calculated by the last Distance function.

Syntax

```
LevRejects As Single (read only)
```

LevReplacements

Description Returns the count of replacements calculated by the last Distance function.

Syntax

```
LevReplacements As Single (read only)
```

LevSame

Description Returns the count of equal characters calculated by the last Distance function.

Syntax

```
LevSame As Single (read only)
```

LevTraceMatrix

Description Returns the Levenshtein trace matrix calculated by the last Distance function.

Syntax

```
LevTraceMatrix As String (read only)
```

LevTraceResult

Description Returns the Levenshtein trace result calculated by the last Distance function.

Syntax `LevTraceResult As String (read only)`

MatchEndPosition

Description Returns the matching end position calculated by the last Distance function.

Syntax `MatchEndPosition As Single (read only)`

MatchStartPosition

Description Returns the matching start position calculated by the last Distance function.

Syntax `MatchStartPosition As Single (read only)`

SearchExpression

Description Contains the search expression which should be used for the next compare operation.

Syntax `SearchExpression As String (read/write)`

ValidateSearchExpression

Description Performs a syntax check for the specified compare method and search expression.

Syntax
```
ValidateSearchExpression (Type As CdrCompareType,
SearchExpression As String)
As Boolean
```

Parameters

Type:	Compare method which should be used for validation.
SearchExpression:	Search expression which should be validated.

6.2 SCBCdrEmailProperties

6.2.1. Description

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 the "**ISCBCdrEmailProperties**" interface that can be queried from the SCBCdrWorkdoc interface.

6.2.2. Properties

CdrMessageSeverity

This type defines the different message severities.

Available Types	Description

CDRSeverityLogFileOnly	Store the message to the application 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. This option is applicable when the call is executed from within the Runtime Server application only.
CDRSeverityEmailNotificatio	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. This option is applicable when the call is executed from within the Runtime Server application only.

6.3 SCBCdrLicenseInfoAccess

6.3.1. Description

The Licensing Information Access object allows direct retrieval to the active licensing object.

The Developer would be able to directly query any licensing component in custom script.

This object is available from Version 5.x and above.

6.3.2. Methods

GetLicenseCounterByID

Description	Returns the license counter information for any given active/inactive license counter. An active license counter is one that is specifically identified in the license file and is enforced by the licensing mechanism.	
Syntax	`GetLicenseCounterByID(CounterID As SCBCdrPROJLib.CDRLicenseCounter, Count As Long, Active As Boolean)`	
Parameters	*CounterID*:	Depicts which counter to retrieve values for. The ID is determined by the CdrLicenseCounter project data type.
	Count:	The returned utilization value from the licensing mechanism. This stores the value of usage.
	Active:	Identifies if the license counter should be active, or specified in the license file.
See Also	CdrLicenseCounter, CdrLicenseFeatureName, GetLicenseCounterByName, GetLicenseValueByID, GetLicenseValueByName, ActivateLicensing	
Example	An example to retrieve the OCRed count of documents in script.	

```
ensingInterface2 As SCBCdrPROJLib.SCBCdrLicenseInfoAccess
```

```
Dim theObject2 As Object
Dim vValue2 As Long
Dim vValue3 As Variant
Dim LicInfoMsg2 As String
vValue2=0
vValue3=0
Project.ActivateLicensing "Designer","C:\Program Files
(x86)\Brainware\Components\Cairo"
Set theObject2 = Project
Set theLicensingInterface2 = theObject2
' theLicensingInterface2.GetLicenseCounterByID(TLCPeriodPagesOCRed,
vValue2, False)
' theLicensingInterface2.GetLicenseCounterByID(TLCTotalPagesOCRed,
vValue3, False)
'
theLicensingInterface2.GetLicenseCounterByID(TLCFineReaderRemainingUnits,
vValue2, True)
theLicensingInterface2.GetLicenseCounterByName ("Overall OCRed Pages",
vValue2, True)
LicInfoMsg2 = "OCRed count - " & CStr(vValue2)
MsgBox(LicInfoMsg2, vbOkOnly,"Get License Count By ID")
```

GetLicenseCounterByName

Description — Returns the license counter information for any given active/inactive license counter.

An active license counter is one that is specifically identified in the license file and is enforced by the licensing mechanism.

Syntax — `GetLicenseCounterByName(CounterName As String, Count As Long, Active As Boolean)`

Parameters — *CounterName* - Depicts which counter to retrieve values for. The Name is the same as shown in the license file.

Count: The returned utilization value from the licensing mechanism. This stores the value of usage.

Active: Identifies if the license counter should be active, or specified in the license file.

See Also — CdrLicenseCounter, CdrLicenseFeatureName, GetLicenseCounterByID, GetLicenseValueByID, GetLicenseValueByName, ActivateLicensing

Example — An example to retrieve the OCRed count of documents in script.

```
ensingInterface As SCBCdrPROJLib.SCBCdrLicenseInfoAccess
  Dim theObject As Object
  Dim vValue1 As Variant
  Dim LicInfoMsg As String

  Project.ActivateLicensing "Designer",""'

  Set theObject = Project
  Set theLicensingInterface = theObject

  theLicensingInterface.GetLicenseCounterByName("OCRed Pages per
  Day", vValue1, True)

  LicInfoMsg = "OCRed count - " & CStr(vValue1)
```

```
MsgBox(LicInfoMsg, vbOkOnly,"Get License Count By Name")
```

GetLicenseValueByID

Description — Returns the license counter information for any given item in the license file.

Syntax — `GetLicenseValueByID(PropertyID As SCBCdrPROJLib.CDRLicenseFeatureName, Value As Variant)`

Parameters

PropertyID: Depicts which item to retrieve values for. Various options can be found in CdrLicenseFeatureName.

Value: The returned value from the licensing mechanism. The data type varies depending on the item being returned.

See Also — CdrLicenseCounter, CdrLicenseFeatureName, GetLicenseCounterByID, GetLicenseCounterByName, GetLicenseValueByName, ActivateLicensing

Example — An example to retrieve the Email Importing flag in the license file.

```
ensingInterface As SCBCdrPROJLib.SCBCdrLicenseInfoAccess
  Dim theObject As Object
  Dim vValue1 As Variant
  Dim LicInfoMsg As String

  Project.ActivateLicensing "Designer",""'

  Set theObject = Project
  Set theLicensingInterface = theObject

  theLicensingInterface.GetLicenseValueByID(CDRLfnEMailsImporting,
  vValue1)
   LicInfoMsg = "Email Importing - " & CStr(vValue1)

  MsgBox(LicInfoMsg, vbOkOnly,"Get License Value By ID")
```

GetLicenseValueByName

Description — Returns the license counter information for any given item in the license file.

Syntax — `GetLicenseValueByName(PropertyName As String, Value As Variant)`

Parameters

PropertyName: Depicts which item to retrieve values for. Various options can be found in the license file. The text to be entered for this parameter should be the exact same text as appears in the license file.

Value: The returned value from the licensing mechanism. The data type varies depending

on the item being returned.

See Also

CdrLicenseCounter, CdrLicenseFeatureName, GetLicenseCounterByID, GetLicenseCounterByName, GetLicenseValueByName, ActivateLicensing

Example

An example to retrieve the Email Importing flag in the license file.

```
ensingInterface As SCBCdrPROJLib.SCBCdrLicenseInfoAccess
  Dim theObject As Object
  Dim vValue1 As Variant
  Dim LicInfoMsg As String

  Project.ActivateLicensing "Designer",""'

  Set theObject = Project
  Set theLicensingInterface = theObject

  theLicensingInterface.GetLicenseValueByName("Serial", vValue1)

  LicInfoMsg = "Primary Dongle Serial Number - " & CStr(vValue1)

  MsgBox(LicInfoMsg, vbOkOnly,"Get License Value By Name")
```

Chapter 7 Cedar Verifier Component Library

7.1 SCBCdrVerificationForm

7.1.1. Description

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.

7.1.2. Methods and Properties

DefaultLabelFont

Description Sets / returns default font for all label elements available on this verification form.

Syntax

```
DefaultLabelFont As StdFont
```

DefaultLabelFontColor

Description Sets / returns default color for all label elements available on this verification form.

Syntax

```
DefaultLabelFontColor As OLE_COLOR
```

Example

```
Dim clrDefaultColor As OLE_COLOR
clrDefaultColor = -1
theForm.VerificationLabels.ItemByIndex(lNextLabelIndex).FontColor =
clrDefaultColor
```

DefaultLabelBackgroundColor

Description Sets / returns default background color for all label elements available on this verification form.

Syntax

```
DefaultLabelBackgroundColor As OLE_COLOR
```

DefaultFieldFont

Description Sets / returns default font for all verification field elements available on this verification form.

Syntax

```
DefaultFieldFont As StdFont
```

DefaultFieldFontColor

Description Sets / returns default color for all verification field elements available on this verification form.

Syntax

```
DefaultFieldFontColor As OLE_COLOR
```

DefaultElementBackgroundColorValid

Description Sets / returns default color for all valid (valid in terms of validation status) field elements available on this verification form.

Syntax `DefaultElementBackgroundColorValid As OLE_COLOR`

DefaultElementBackgroundColorInvalid

Description Sets / returns default color for all invalid (invalid in terms of validation status) field elements available on this verification form.

Syntax `DefaultElementBackgroundColorInvalid As OLE_COLOR`

FormBackgroundColor

Description Sets / returns background color for the form.

Syntax `FormBackgroundColor As OLE_COLOR`

FormBackgroundColorDI

Description Sets / returns background color for the Direct Input control on the form, i.e. for the area around the Direct Input field.

Syntax `FormBackgroundColorDI As OLE_COLOR`

7.2 SCBCdrVerificationField

7.2.1. Description

This interface is used to identify verification properties specific for header fields' validation elements, like drop down lists, check-boxes, and normal edit fields.

*Note: In order to get the OLE_COLOR object for the types below, add **OLE Automation** as a reference.*

7.2.2. Type Definitions

CdrVerifierFieldType

Description The Verifier Field type.

This type interface is a member of the Cedar Verifier Project library.

Available Types *CDRVerifierFieldTypeCheckbox* – Checkbox field type, value = 2.

CDRVerifierFieldTypeCombobox – Combobox field type, value = 3.

CDRVerifierFieldTypeTableCheckBoxCell – Table Checkbox Cell field type, value = 4.

CDRVerifierFieldTypeTextMultiline – Multiline Text field type, value = 1.

CDRVerifierFieldTypeTextSingleline – Single Line Text field type, value = 0.

7.2.3. Methods and Properties

AutoCompletionEnabled

Description	This property enables / disables the field text Auto Completion for a verification field.
Attribute	Read/Write
Example	The example below turns Auto Completion on for the Invoice Number field.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").AutoCo
mpletionEnabled = True
```

BackgroundColorInvalid

Description	This property sets the color for the verification field to display to the user when the field required manual verification. When the field is Invalid in Verifier, the color that is set will display to the user. By default, the invalid background color of the field is red.
Attribute	Read/Write
See Also	BackgroundColorValid
Example	The example below turns the Invalid color for Invoice Number field to gray if it is Invalid.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationFields.ItemByName("InvoiceNo").BackgroundCo
lorInvalid = RGB (192, 129, 129)
```

BackgroundColorValid

Description	This property sets the color for the verification field to display to the user when the field does not require manual verification. When the field is Valid in Verifier, the color that is set will display to the user. By default, the invalid background color of the field is red.
Attribute	Read/Write
See Also	BackgroundColorInvalid
Example	The example below turns the Invalid color for Invoice Number field to gray if it is Valid.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Backgro
undColorValid = RGB (192, 129, 129)
```

Font

Description	This property sets the Font for the content of the verification field. Note: In order to get the StdFont object, add *OLE Automation* as a reference.
Attribute	Read/Write
See Also	FontColor
Example	The example below sets the Font for Invoice Number field.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim DefaultFieldFont As New StdFont

DefaultFieldFont.Bold = False  'Set Font attributes

  ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Fon
t = DefaultFieldFont
```

FontColor

Description This property sets the Font Color for the content of the verification field.

Attribute Read/Write

See Also Font

Example The example below sets the FontColor for Invoice Number field to gray.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Fon
tColor = RGB (192, 129, 129)
```

Invisible

Description This property determines if the field is visible or hidden from the Verifier / Web Verifier form. The developer uses script options to hide or display the field from the verifier user. For the Web Verifier this method is used in the VerifierFormload event.

Attribute Read/Write

Example The example below hides the Invoice Number field from the verifier user.

```
 Dim theVerificationProject As
 DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

 Dim theVerificationForm As
 DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

 Project.GetVerifierProject theVerificationProject

 Set theVerificationForm =
 theVerificationProject.AllVerificationForms.ItemByName("Invoices")

 theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").In
 visible = True

' Update the form

theVerificationForm.RepaintControls
```

Left

Description This property provides the left position of the field on the Verifier form.

Attribute Read/Write

See Also Top, Width

Example The example below retrieves the Left position of the Invoice Number

field from Verifier Form.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim LeftPos As Integer

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

LeftPos =
theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Lef
t
```

Name

Description	This property provides the Name of the field on the Verifier form.
Attribute	Read
Example	The example below retrieves the Name of the Invoice Number field from Verifier Form.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim FieldName As String

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

FieldName =
theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Nam
e
```

ReadOnly

Description	This property determines if the verification field on the Verifier / Web Verifier form is editable or Read Only. For the Web Verifier use this method in the VerifierFormLoad event. Setting the property to True will make the field non-editable.
Attribute	Read / Write
Example	The example below sets the Invoice Number field as Read Only on the Verifier Form.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form
```

```
Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Rea
dOnly = True

theVerificationForm.RepaintControls     ' Update the form UI
```

TabIndex

Description	This property allows the scripter to set the tab sequence number of the verification field on the Verifier form. The Tab sequence is typically configured on the verification form in Designer, this script method allows the scripter to change the sequence number to re-ordering TAB sequence of fields.
Attribute	Read / Write
Example	The example below sets the Invoice Number field tab sequence on the Verifier Form.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Tab
Index = 5
```

Top

Description	This property provides the top position coordinates of the field on the Verifier form. The scripter may choose to reorder positional information of the field if another element is being hidden. Using the RepaintControls method, the form UI will be updated with the changes made.
Attribute	Read/Write
See Also	Left, Width, RepaintControls
Example	The example below retrieves the Top position of the Invoice Number field from Verifier Form.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim TopPos As Integer

 ' Request the main form

Project.GetVerifierProject theVerificationProject
```

```
Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

TopPos =
theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Top
```

Type

Description	This property provides the Field Type information of the field on the Verifier form. The scripter may choose to review information of the field type.
Attribute	Read
See Also	CdrVerifierFieldType
Example	The example below retrieves the Field Type Information of the Invoice Number field from Verifier Form.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim FieldInfo As CdrVerifierFieldType

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

FieldInfo =
theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Wid
th
```

Width

Description	This property provides the Width size information of the field on the Verifier form. The scripter may choose to reorder or resize positional information of the field if another element is being hidden. Using the RepaintControls method, the form UI will be updated with the changes made.
Attribute	Read/Write
See Also	Left, Top, RepaintControls
Example	The example below retrieves the Width Information of the Invoice Number field from Verifier Form.

```
Dim theVerificationProject As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim WidthInfo As Integer

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
```

```
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

WidthInfo =
theVerificationForm.VerificationFields.ItemByName("Field_InvoiceNo").Width
```

7.3 SCBCdrVerificationTable

7.3.1. Description

This interface is used to identify verification properties specific for table validation elements.

7.3.2. Methods and Properties

FontFont

Description	Sets / returns font settings for the individual table field element.
Syntax	`FontFont As StdFont`

BackgroundColorValid

Description	Sets / returns background color for the individual verification table element, when the table cell is valid in terms of current validation status.
Syntax	`BackgroundColorValid As OLE_COLOR`

BackgroundColorInvalid

Description	Sets / returns background color for the individual verification table element, when the table cell is invalid in terms of current validation status.
Syntax	`BackgroundColorInvalid As OLE_COLOR`

HeaderFont

Description	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).
Syntax	`HeaderFont As StdFont`

HeaderFontColor

Description	Sets / returns font color for the header buttons of the table field element, including row header buttons and column header buttons.
Syntax	`HeaderFontColor As OLE_COLOR`

HeaderBackgroundColor

Description	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.
Syntax	`HeaderBackgroundColor As OLE_COLOR`

7.4 SCBCdrVerificationButton

7.4.1. Description

This interface is used to set verification properties specific for all custom buttons defined on a verification form.

7.4.2. Methods and Properties

Font

Description	Sets / returns font settings (name, type, and styles) for the individual custom button control.
Syntax	`Font As StdFont`

FontColor

Description	Sets / returns font color for the individual custom button control.
Syntax	`FontColor As OLE_COLOR`

BackgroundColor

Description	Sets / returns background color for the individual custom button control.
Syntax	`BackgroundColor As OLE_COLOR`

7.5 SCBCdrVerificationLabel

7.5.1. Description

This object is part of the Cedar Verifier Component Library. It enables the scripter to manipulate the verifier form.

Cedar Verifier Component Library is not enabled by default. This component can be added to the script references for any project class.

The Cedar Verification Label Object allows for the manipulation of the field for the verifier user (eg Font and color that appears when a user views a field label on the verifier form).

7.5.2. Properties

BackgroundColor

Description	This property sets the color for the verification text label to display to the user.

By default, the background color of the field is gray.

Syntax `BackgroundColor As OLE_COLOR`

Attribute Read/Write

Example The example below turns the color for Invoice Number label to gray.

```
ificationProject As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject
  Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm
   ' Request the main form
  Project.GetVerifierProject theVerificationProject
  Set theVerificationForm =
  theVerificationProject.AllVerificationForms.ItemByName("Invoices")
  theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Backgro
  undColor = RGB (192, 129, 129)
```

Font

Description This property sets the Font for the content of the verification field label.

Note: In order to get the StdFont object, add OLE Automation as a reference.

Syntax `Font As StdFont`

Attribute Read/Write

See Also FontColor

Example The example below sets the Font for Invoice Number field label.

```
ificationProject As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject
  Dim theVerificationForm As
  DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm
  Dim DefaultLabelFont As New StdFont
  DefaultLabelFont.Bold = False  'Set Font attributes
  'Request the main form
  Project.GetVerifierProject theVerificationProject
  Set theVerificationForm =
  theVerificationProject.AllVerificationForms.ItemByName("Invoices")
  theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Font
  = DefaultLabelFont
```

FontColor

Description This property sets the Font Color for the content of the verification field label.

Note: In order to get the OLE_COLOR object, add OLE Automation as a reference.

Syntax `FontColor As OLE_COLOR`

Attribute Read/Write

See Also Font

Example The example below sets the FontColor for Invoice Number field label to blue.

```
ificationProject As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

  Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

   ' Request the main form

  Project.GetVerifierProject theVerificationProject

  Set theVerificationForm =
  theVerificationProject.AllVerificationForms.ItemByName("Invoices")

  theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").FontCo
  lor = RGB (0, 0, 255)
```

Invisible

Description This property determines if the field label is visible or hidden on the Verifier form. The developer may script options to hide or display the field label from the verifier user.

Attribute Read/Write

Example The example below hides the Invoice Number field label from the verifier user.

```
Dim theVerificationProject As
            DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Invisib
le = True

' Update the form

theVerificationForm.RepaintControls
```

Left

Description This property provides the left position of the field on the Verifier form.

Attribute Read/Write

See Also Top, Width

Example The example below retrieves the Left position of the Invoice Number field label from Verifier Form.

```
Dim theVerificationProject As
            DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
```

```
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim LeftPos As Integer

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

LeftPos =
theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Left
```

Name

Description This property provides the Name of the field label on the Verifier form.

Attribute Read

Example The example below retrieves the Name of the Invoice Number Label field from Verifier Form.

```
Dim theVerificationProject As
            DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim FieldName As String

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

FieldName =
theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Name
```

Text

Description This property allows the scripter to set the text of the verification field label on the Verifier form.

Attribute Read / Write

Example The example below sets the Invoice Number field tab sequence on the Verifier Form.

```
Dim theVerificationProject As
            DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Text
= “Invoice Number”
```

Top

Description: This property provides the top position coordinates of the field label on the Verifier form.

The scripter may choose to reorder positional information of the field label if another element is being hidden. Using the RepaintControls method, the form UI will be updated with the changes made.

Attribute: Read/Write

See Also: Left, Width, RepaintControls

Example: The example below retrieves the Top position of the Invoice Number field from Verifier Form.

```
Dim theVerificationProject As
            DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As
DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim TopPos As Integer

 ' Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

TopPos =
theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Top
```

Width

Description: This property provides the Width size information of the field label on the Verifier form.

The scripter may choose to reorder or resize positional information of the field label if another element is being hidden. Using the RepaintControls method, the form UI will be updated with the changes made.

Attribute: Read/Write

See Also: Left, Top, RepaintControls

Example: The example below retrieves the Width Information of the Invoice Number field label from Verifier Form.

```
Dim theVerificationProject As
            DISTILLERVERIFIERCOMPLib.SCBCdrVerificationProject

Dim theVerificationForm As DISTILLERVERIFIERCOMPLib.SCBCdrVerificationForm

Dim WidthInfo As Integer

'Request the main form

Project.GetVerifierProject theVerificationProject

Set theVerificationForm =
theVerificationProject.AllVerificationForms.ItemByName("Invoices")

WidthInfo =
theVerificationForm.VerificationLabels.ItemByName("Label_InvoiceNo").Width
```

Chapter 8 Password Encryption for Database Connection Strings

The application architecture of Perceptive Intelligent Capture makes it very important to be able to hide sensitive security information, such as DB access password, stored in Perceptive Intelligent Capture or custom project configuration files.

The same requirement also applies to the database connection strings in the Perceptive Intelligent Capture project INI files that often contain multiple connection strings to different database instances (like for Visibility reporting or custom databases) with unencrypted password info. These INI files may not reside directly on the local Verifier workstation, but still can be easily accessed by the Verifier users, because at least the read-only access to the Perceptive Intelligent Capture project directory is a requirement for Perceptive Intelligent Capture applications.

Below are the steps to implement password encryption for custom configuration files used when loading Perceptive Intelligent Capture projects:

8.1 Master Project Side (Project Primary Developer)

Prerequisites

Before you start please request a pair of RSA encryption keys from Perceptive Customer Support. In terms of testing you can though use the pair of test keys below. However, do request a new pair before releasing your master project to the others.

Keep your private key safe - do NOT provide to anyone else! Only the public key should be distributed to those who use your project for custom implementations!

Test Public Key

```
<RSAKeyValue><Modulus>vJ+W7SuXuvOrWVoy4tPrbfLCuoHElo750cpTuEzLPk6iz6bHAodPVgLFaOEK+XMMS2G5z+6
961vuQsDGUt+O1Ag1PiTXCa6rrAaeCaaDO4HI8Mmpw0OkUZEfCZpTTYCYQPfZlgokwomF6VDSB9dlUS430IT0gctQY1b5
iM4MqT0=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>
```

Test Private Key

```
<RSAKeyValue><Modulus>vJ+W7SuXuvOrWVoy4tPrbfLCuoHElo750cpTuEzLPk6iz6bHAodPVgLFaOEK+XMMS2G5z+6
961vuQsDGUt+O1Ag1PiTXCa6rrAaeCaaDO4HI8Mmpw0OkUZEfCZpTTYCYQPfZlgokwomF6VDSB9dlUS430IT0gctQY1b5
iM4MqT0=</Modulus><Exponent>AQAB</Exponent><P>8SRHEvT5Bn2paRHSDR9yCQb7WGYE9PbeHzuqwH6iWa0LNYJ
rSrhhUeCEpwlPLQWQq10KmMZgG0+Br4nuBMmMHQ==</P><Q>yD7l9fjB/MJWYaV3LcEzY286Q+Xvo74i6THvHkKqB1NKY
GcN9xF9d8XbiUQNgBZ/4F02T6mFeYDO32KFVRXHoQ==</Q><DP>nRDTFn7nwRmSgfRwi8minkyk5DQ3IFO35EIZ+x3Ao4
Z52ZWkStwDz6/c12vR3XJVg7irkU0NBlzoDK1bklSw5Q==</DP><DQ>B3xieGmORva05/2ZkPpSA3ubAALOjJ6FC5a0S7
tOQ+vXMfdoTD45JIsfA+ipYIp2yVpyt1OtC7fHBA7Y0S95QQ==</DQ><InverseQ>4S1xqlXK9f1rawGCbFWOVp6lz1fC
oQ8RfyDE87/G/pUilHRJV2acBAcngY3c/MRMKrXQb8lx99k7dENUYc8ywQ==</InverseQ><D>KAL6cwkCQKgbuvKFRNS
LZmFOqV2JpB5kI/p1U+0GWAs6Qi4wnPqy+53O3naOa2faPctXLSKJqvlvSz21VDMUCsyphvOSxBtc1cZHJp4ueQPA7u+q
rIJaDY1RhlAVoqNfCJFX6+McVJ+I/X+mZOCtdUaCuAoNn014UYOaMujYDQE=</D></RSAKeyValue>
```

Implementation Guidelines

1. Split the connection string in your configuration files to encrypted and non-encrypted parts.

   a. Example of connection string of "BW Packaged.ini" before splitting:

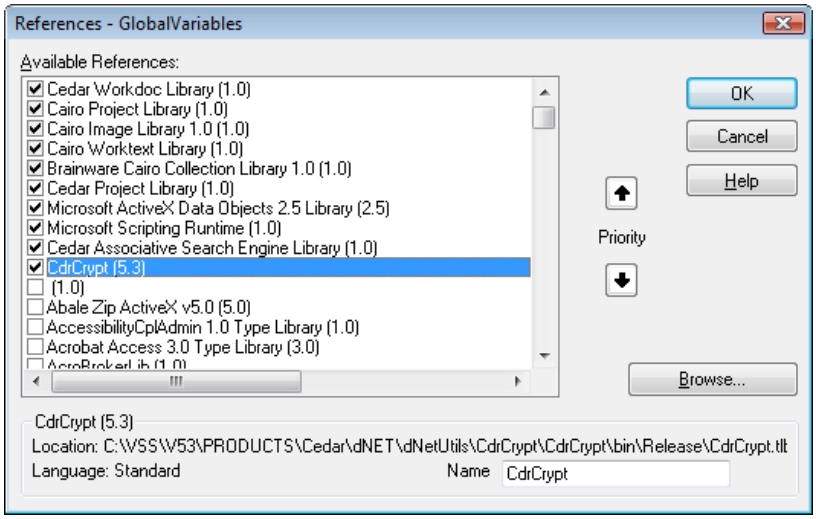
   ```
   SQL_VL_01_ConnectionString=Provider=SQLOLEDB.1;Password=alexey 123456789;Persist
   Security Info=True;User ID=alexey;Initial Catalog=Visibility;Data Source=KIR-AE-NB-
   03\SQLSERVER2008R2
   ```

   b. Example of connection string of "BW Packaged.ini" before splitting (the red part of the example below is now packaged as an extended new variable - see the red part below):

```
SQL_VL_01_ConnectionString=Provider=SQLOLEDB.1;Persist Security Info=True;User
ID=alexey;Initial Catalog=Visibility;Data Source=KIR-AE-NB-03\SQLSERVER2008R2

SQL_VL_01_ConnectionPassword=encrypted_password_is_to_be_placed_here
```

2. Open your master project in Designer, run script editor, open the script page, where you would like to implement connection string encryption and add the Reference to "CdrCrypt (5.3)" type library:

Note: If it does not show up in the list of libraries, click on "Browse..." button, browse to the .\Application\bin and open the CdrCrypt.tlb.



3. At the place of the same script page, where connection string is supposed to be read from the configuration (INI) file and then used further to connect to the database add a script code, similar to the one below:

```
Dim theCedarCryptographyHelper As New CdrCrypt.RSACodecInt

Dim strEncryptedPassword As String

Dim strOpenPassword As String

Dim strPrivateKey As String

strPrivateKey =
"<RSAKeyValue><Modulus>vJ+W7SuXuvOrWVoy4tPrbfLCuoHElo750cpTuEzLPk6iz6bHAodPVgLFaOEK+XMMS
2G5z+6961vuQsDGUt+O1Ag1PiTXCa6rrAaeCaaDO4HI8Mmpw0OkUZEfCZpTTYCYQPfZlgokwomF6VDSB9dlUS430
IT0gctQY1b5iM4MqT0=</Modulus><Exponent>AQAB</Exponent><P>8SRHEvT5Bn2paRHSDR9yCQb7WGYE9Pb
eHzuqwH6iWa0LNYJrSrhhUeCEpwlPLQWQq10KmMZgG0+Br4nuBMmMHQ==</P><Q>yD7l9fjB/MJWYaV3LcEzY286
Q+Xvo74i6THvHkKqB1NKYGcN9xF9d8XbiUQNgBZ/4F02T6mFeYDO32KFVRXHoQ==</Q><DP>nRDTFn7nwRmSgfRw
i8minkyk5DQ3IFO35EIZ+x3Ao4Z52ZWkStwDz6/c12vR3XJVg7irkU0NBlzoDK1bklSw5Q==</DP><DQ>B3xieGm
ORva05/2ZkPpSA3ubAALOjJ6FC5a0S7tOQ+vXMfdoTD45JIsfA+ipYIp2yVpyt1OtC7fHBA7Y0S95QQ==</DQ><I
nverseQ>4S1xqlXK9f1rawGCbFWOVp6lz1fCoQ8RfyDE87/G/pUilHRJV2acBAcngY3c/MRMKrXQb8lx99k7dENU
Yc8ywQ==</InverseQ><D>KAL6cwkCQKgbuvKFRNSLZmFOqV2JpB5kI/p1U+0GWAs6Qi4wnPqy+53O3naOa2faPc
tXLSKJqvlvSz21VDMUCsyphvOSxBtc1cZHJp4ueQPA7u+qrIJaDY1RhlAVoqNfCJFX6+McVJ+I/X+mZOCtdUaCuA
oNn014UYOaMujYDQE=</D></RSAKeyValue>"

strEncryptedPassword = DicVal("01" & "ConnectionPassword", "SQL")

If Len(strEncryptedPassword) > 0 Then

  strOpenPassword = theCedarCryptographyHelper.Decode(strEncryptedPassword,
strPrivateKey)

End If

If Len(strOpenPassword) > 0 Then

  strConnection = strConnection + ";Password=" + strOpenPassword

End If
```

4. Make sure you encrypt the script page that contains the code above via standard script code encryption feature.

   Alternatively, you leave the code above unencrypted, but place the "strPrivateKey" variable and its initialization on another encrypted page available from the code above.

5. When you release your master project to the others, distribute the public key along with the project release - PS representatives who will be installing your project on customer site, will use this public key to encrypt their custom passwords.

Index