Oracle® Fusion Middleware

Oracle WebCenter Forms Recognition Scripting User's Guide

12c Release 1 (12.2.1.4.0)

E93588-02

September 2019

Documentation for WebCenter Forms Recognition project developers that describes how to develop scripts for creating and customizing projects.

ORACLE®

Oracle Fusion Middleware Oracle WebCenter Forms Recognition Scripting User's Guide, 12c Release 1 (12.2.1.4.0)

E93588-02

Contents

1 Script Event Reference

1.1 Description: VerifierFormLoad Event

To implement the script handler of this event, complete the following steps:

1. Start the WebCenter Forms Recognition Designer application.
2. Load the project file.
3. Select the project node in *Definition Mode*.
4. Open the *Script Editor*.
5. Select the **Script Module** object, and click the **VerifierFormLoad** item in the **Proc** drop-down list.

For example, the following simple implementation of the `VerifierFormLoad` event, (in this simple case non-optionally) replaces the standard form `Form_Invoices_1` with a custom `Form_Invoices_2` defined for the same document class.

Example:
```
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, the Verifier application will always load the simple second form specified in the script.

If the script modifies the form and the form's class references incorrectly, a warning message displays to the Verifier user. For example, an incorrect script modification can occur when a reference is made to a non-existing verification form of a class or when the form does not exist in the specified class.

Example:
```
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 displays the following warning message:

```
The default form was requested to be reloaded from script, but the requested
form appeared to be inconsistent.

Default form name: Form_Invoices_1
Default form class: Invoices
Requested form name: Non-existing form name
Requested form class: Non-existing class name
Please contact your system administrator to adjust the project
configuration appropriately.
```

The application then loads the standard verification form, the one that the application would load anyway if the script handler of `VerifierFormLoad` event did not exist, instead of the wrong one proposed by the custom script.

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

If you enable **Allow Firing of VerifierFormLoad event when in Verifier Test/Train modes**, the system triggers this event in Designer Verifier Test and Train modes.

In Designer, if you enable **Allow firing of FocusChanged event when loading the verification form**, the system fires the document-class `FocusChanged` event with the `Reason` parameter set to `CdrBeforeFormLoaded`. The system fires the event after the `VerifierFormLoad` event occurs (described above) and before the desired verification form loads.

Below is an example of a script that shows how you can implement the handler of this extended reason in the WebCenter Forms Recognition custom script.

Example:

```
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

You can use the features described in this section for many different purposes, including the following examples:

- To optionally load a non-standard verification form in accordance with some parameters of the processed document.
- To dynamically translate the content of verification forms into a different language, or simply to 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

1.2.1 ScriptModule Event Interface

Project events are specific for one WebCenter Forms Recognition project. However, within a project, all documents and fields share the same implementation of these events. This means that they are document class independent. As the project events belong to the sheet `ScriptModule`, all events start with the prefix **ScriptModule**.

1.2.2 Create a Script for Document Export

The following script writes classified images to subdirectories in the export directory. Each subdirectory holds documents from one class only; the subdirectory name corresponds to the class name.

1. In Designer, switch to *Definition Mode*.
2. On the toolbar, click **Show/hide script** button.
3. In the **Script View for** Project dialog box, from the **Object** list, select **ScriptModule**.
4. From the **Proc** list, select **ExportDocument**.
   This generates the outline of a subroutine.
5. Add the following code.

```
Dim sNewPath As String
Dim MyImage As SCBCroImage
Dim NewFileName As String
sNewPath=ExportPath & "\" & pWorkdoc.DocClassName 'Set directory name
Set MyImage=pWorkdoc.Image(0) 'Access the image file to the current
WorkDoc
On Error GoTo Skip 'Skip next step if directory exists
MkDir sNewPath 'Create directory
Skip:
NewFileName=Mid(MyImage.Filename, InStrRev(MyImage.Filename,"\")) 'Set
file name
MyImage.SaveFile sNewPath & NewFileName 'Save file to directory
```

6. Close the dialog box.
7. Switch to Runtime Mode and test the script.

1.2.3 Methods and Properties

1.2.3.1 AppendWorkDoc

This event is fired at the time of processing a document separation workflow step, at Runtime Server. It can be used to append a given workdoc after the last workdoc on the base of *CdrMPType*.

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

Parameter	Description
`pLastWorkdoc`	The last workdoc object.
`pCurrentWorkdoc`	The current workdoc object.
`pAppendType`	Defines the possible results of the multipage classification. For example, **CdrAttachmentPage** would mean that the engine has classified the page as an attachment or **CdrFirstPage** would mean that the engine has classified this page as the start of a new document.

1.2.3.2 BatchClose

This event launches when the Verifier user exits a batch in one of the following methods:

- When verifying a batch and returning to the batch list.
- Batch verification completion.
- Partial batch verification completion.
- The user quits the Verifier application while in a batch.

The event is triggered in the Verifier and 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)
```

Parameter	Description
`UserName`	The username of the currently logged-in user who has opened the batch.
`BatchDatabaseID`	The unique batch ID within the database. For batches stored in the file system, this batch ID is not used. The batch ID displays as a numeric value. For example, for Batch 00000061, the value **61** is returned.
`ExternalGroupID`	The group ID that can be assigned to a batch. The group ID that can be used with the scripting security methods that enable the developer to assign a batch a security group. Only those users belonging to the same group ID are able to access batches. For example, a batch belonging to group ID 80 is only accessible by a user who is assigned to group **80**.
`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, such as archive ID or a storage box ID.
`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 or a storage box ID.
`WorkflowType`	Corresponds to *CDRDatabaseWorkflowTypes* data type.
`BatchState`	The current status of the batch being closed, such as status **550**.
`BatchReleaseAction`	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 the user always makes a selection. If a registry value is used to hide the batch release dialog box in Verifier, then the last action taken prior to the dialog box being hidden is the one that shows in this parameter. The developer can set an override value to this parameter, such as every time batch verification completes.

1.2.3.3 BatchOpen

This event triggers 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)`

Parameter	Description
`UserName`	The username of the currently logged-in user who has opened the batch.
`BatchDatabaseID`	The unique batch ID within the database. For batches stored in the file system, this batch ID is not used. The batch ID displays as a numeric value. For example, for Batch 00000061, the value **61** is returned.
`ExternalGroupID`	The group ID that can be assigned to a batch. The group ID that can be used with the scripting security methods that enable the developer to assign a batch a security group. Only those users belonging to the same group ID are able to access batches. For example, a batch belonging to group ID 80 is only accessible by a user who is assigned to group **80**.
`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, such as archive ID or a storage box ID.
`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 or a storage box ID.
`WorkflowType`	Corresponds to *CDRDatabaseWorkflowTypes* data type.
`BatchState`	The current status of the batch being closed, such as status **550**.

1.2.3.3.1 Sample Code

The following sample code logs the Batch ID and User name that opened a batch with date and time.

LogMessage is a custom function that 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_Log")
 End Sub
```

1.2.3.4 ExportDocument

This event allows a project developer to implement a customer-specific export of all extracted data.

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

Parameter	Description
`pWorkdoc`	The workdoc object that should be exported.
`ExportPath`	Export path that was configured within the Runtime Server settings.
`pCancel`	Set this variable to `True` to cancel the export.

1.2.3.4.1 Create a Script for Document Export

The following script writes classified images to subdirectories in the export directory. Each subdirectory holds documents from one class only; the subdirectory name corresponds to the class name. To create the script, complete the following steps:

1. In Designer, switch to **Definition Mode**.
2. On the toolbar, click **Show/hide script**.
3. In the **Script View for Project** dialog box, from the **Object** list, select **ScriptModule**.
4. From the **Proc** list, select **ExportDocument**.
5. Add the following code:

```
Example Dim sNewPath as String
 Dim MyImage as SCBCroImage
 Dim NewFileName as String
 sNewPath=ExportPath & "\" & pWorkdoc.DocClassName  'Set directory name
 Set MyImage=pWorkdoc.Image(0)  'Access the image file to the current WorkDoc
 On Error GoTo Skip  'Skip next step if directory exists
 MkDir sNewPath      'Create directory
 Skip:
 NewFileName=Mid(MyImage.Filename,  InStrRev(MyImage.Filename,"\")) 'Set file
name
 MyImage.SaveFile sNewPath & NewFileName 'Save file to directory
```

6. Close the dialog box.
7. Switch to **Runtime Mode** and test the script.

1.2.3.5 Initialize

This event is called when a batch is opened for processing.

Syntax: `ScriptModule_Initialize(ModuleName As String)`

Parameter	Description
`ModuleName`	Name of the current module. Valid values are **Server**, **Designer**, **Verifier** or **Thin Client Verifier**.

1.2.3.5.1 Sample Code

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

1.2.3.6 MoveDocument

This event is launched when the Verifier user places a document in an exception state, and the document is moved out of the batch.

The *ScriptModule* provides the 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 triggers for each document that is placed into exception within a single batch.

After placing a document to an exception state, the event is 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)`

Parameter	Description
`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. The only reason implemented at this point is for the document moved to exception.
`DocState`	The workflow state of the document.

1.2.3.6.1 Sample Code

The following sample code 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
```

1.2.3.7 PostClassify

This event is called after all defined classification methods are executed by the project.

Syntax: `ScriptModule_PostClassify(pWorkdoc As SCBCdrPROJLib.SCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	The workdoc object that has been classified.

1.2.3.7.1 Sample Code

```
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 works only 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.DocFileName(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
```

1.2.3.7.2 Sample Code

```
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, lngTagID, "Test")
End Sub
```

1.2.3.8 PostImport

In general, this event triggers after the import.

Processing for failed documents imported from the file system.

- The event only triggers if **Perform advanced import failure processing** is active in Runtime Server.
- The event does not trigger if the `pCancel` parameter was set to true in the PreImport event.

Processing for failed documents imported from external batches.

- The event always triggers

Syntax: `ScriptModule_PostImport(pWorkdoc As SCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	Workdoc object to import.

See also "Activate advanced import failure processing" in the Windows Forms Recognition Runtime Server Guide.

1.2.3.9 PostImportBatch

This is an event that is triggered by the Runtime Server after it has finished importing the batch. Use this event to set batch properties such as batch name and external group ID.

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)`

Parameter	Description
`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 that cannot be modified.
`BatchName`	The batch name that is assigned by the Runtime Server instance. The name is taken from the *Import* settings of the Runtime Server instance. Read or Write Parameter that can be modified to any string value.
`Priority`	The batch priority that is assigned by the Runtime Server instance. The priority is taken from the *Import* settings of the Runtime Server instance. Read or Write Parameter that can be modified to any long value between **1** to **9**.
`State`	The batch state that is assigned by the Runtime Server instance. The status is taken from the *Workflow* settings of the Runtime Server instance.

	Read or Write Parameter that can be modified to any long value between **100** and **999**.
`ExternalGroupID`	The group ID that can be assigned to a batch. The group ID that can be used with the scripting security methods that enable the developer to assign a batch a security group. Only those users belonging to the same group ID are able to access batches. For example, a batch belonging to group ID 80 is only accessible by a user who is assigned to group **80**.
`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, such as archive ID or a storage box ID.
`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 or a storage box ID.
`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 archive ID or a storage box ID. Read or Write Parameter that can be modified to any long value.

1.2.3.9.1 Sample Code

The following sample code updates the batch priorities after the import is done. It changes the name, state and adds a group ID as well as a transaction type and an ID.

```
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
```

See Also

- SecurityUpdateStart
- SecurityUpdateAddUserGroup
- SecurityUpdateCommit

1.2.3.10 PostOCR

This event triggers after the OCR process.

Syntax: `ScriptModule_ PostOCR(pWorkdoc As SCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	The workdoc object.

1.2.3.11 PreClassify

This event is called before any defined classification method is executed by the project. During this event, it is possible to apply an existing name of a document class to the workdoc.

Syntax: `ScriptModule_PreClassify(pWorkdoc As SCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	The workdoc object that should be classified.

1.2.3.11.1 Sample Code

```
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
```

1.2.3.12 PreClassifyAnalysis

This event is fired between the *PreClassify* and *PostClassify* events that identify the beginning and end of the *Classification* workflow step for a particular document. Using this event, the custom script can clean-up and extend classification results before the final decision is made by the system and before the final classification matrix is built.

Syntax: `ScriptModule_PreClassifyAnalysis(pWorkdoc As SCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	The workdoc object that should be classified.

1.2.3.13 PreImport

This event triggers before the import occurs.

Syntax: `ScriptModule_ PreImport(pWorkdoc As SCBCdrWorkdoc, FilePath As String, FileType As CDRDocFileType, pCancel As Boolean)`

Parameter	Description
`pWorkdoc`	Workdoc object.
`FilePath`	Path of the document to import.
`FileType`	File type See CDRDocFileType for valid values.

pCancel	▪ **True**: Skip the import of the current document. When importing documents from an external batch, the PostImport event triggers. When importing documents from the file system, the PostImport event does not trigger. ▪ **False**: Import the current document.

1.2.3.14 PreOCR

This event triggers before the OCR process occurs.

Syntax: `ScriptModule_ PreOCR(pWorkdoc As SCBCdrWorkdoc, pCancel As Boolean)`

Parameter	Description
pWorkdoc	The batch object that is being processed.
pCancel	Set this parameter to True to skip the OCR process for the current document.

1.2.3.15 ProcessBatch

This event is launched when the Runtime Server instance begins processing during the *Custom Processing* workflow step.

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

Parameter	Description
pBatch	The batch object that is being processed.
InputState	The input state of the batch when *Custom Processing* was activated on it.
DesiredOutputState Succeeded	The output state of the batch if the workflow step succeeds.
DesiredOutputState Failed	The output state of the batch if the workflow step failed.

Use the corresponding Terminate event script instead to delete these empty batches. Do not use both scripts within one project, because the Terminate event script makes it impossible to load the ProcessBatch script.

1.2.3.15.1 Sample Code

Add the following sample code to the very beginning of the ProcessBatch event to stop an 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 program uses the default value, which is zero.

```
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 & "]"
             "Detected batch with zero foldersdirectories: [" & pBatch.BatchID &
"]"
             pBatch.BatchState = 987
          End If
       On Error Resume Next
       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
            For lFolderIndex = 0 To pBatch.FolderCount-1 Step 1
               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
                  ' 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
               ' 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 successfully, 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
```

1.2.3.16 RouteDocument

This event is launched when a document has been extracted.

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

Parameter	Description

pWorkdoc	The workdoc object that was classified and extracted
State	This parameter contains the current state that is assigned to the workdoc. The value can be changed from the script.

1.2.3.16.1 Sample Code

```
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 directory is shared between multiple organizations (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 that is country based (such as, GB is status 550, Germany is status 551).

```
Private Sub ScriptModule_RouteDocument(pWorkdoc as SCBCdrPROJLib.SCBCdrWorkdoc,
State as Integer)
   'If the batch state is 550 and document is not in verifier
   If State = 550 And Not fnIsVerifier() Then       'Check country code and set
batch status
     Select Case CountryCode
       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
     'Save the work doc after changing document status
     pWorkdoc.Save(pWorkdoc.Filename,"")
   End If
End Sub
```

1.2.3.17 Terminate

This event is called before a batch is closed after processing.

Syntax: `ScriptModule_Terminate(ModuleName as String)`

Parameter	Description
ModuleName	The name of the current WebCenter Forms Recognition module. Possible values are: ▪ **Designer** ▪ **Verifier** ▪ **Server**

1.2.3.17.1 Sample Code

```
Private Sub ScriptModule_Terminate(ByVal ModuleName as String)
    DB.Close
    Set DB = nothing
 End Sub
This script when added to one of the real projects triggers the Terminate event
in Runtime Server. This script erases all state 0 batches that contain zero
directories. Do not use this script piece together with the corresponding
ProcessBatch event script within one project, because this Terminate event
script makes 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("Job name", "Zero Folder Batch
Terminator", False)
    pBatchRoot.Connect("Job name", "", "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
```

1.2.3.18 UpdateSystemSecurity

This event is triggered when the Runtime Server is configured to run with security updates.

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)`

Parameter	Description

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

1.2.3.18.1 Sample Code

The following sample code updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the 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, "VER|SET", "BDomain"
    Project.SecurityUpdateAddUserGroup "User6", 777, "VER|FLT", "BDomain"
    Project.SecurityUpdateAddUserGroup "User7", 333, "AEB", "BDomain"
    Project.SecurityUpdateAddUserGroup "User10", 777, "ADM", "BDomain"
    Project.SecurityUpdateCommit
End Sub
```

For additional information, see *SecurityUpdateStart*, *SecurityUpdateCommit*, SecurityUpdateUserPrameter and *PostImportBatch*.

1.2.3.19 VerifierClassify

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

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

Parameter	Description
`pWorkdoc`	Reference to the currently processed document.
`Reason`	The reason why the script routine decided to reject or accept the document. Possible value: `CdrVerifierClassifyReason`
`ClassName`	The name of the document class to which it is classified manually.

1.2.3.20 VerifierException

This event triggers when a document or batch is moved to exception state in Verifier.

Syntax: `ScriptModule_ VerifierException(pWorkdoc As SCBCdrWorkdoc, Reason As SCBCdrPROJLib.CDRVerifierExceptionReason, CreateNewBatch As Boolean, BatchName As String, BatchDocumentState As Long, BatchPriority As Long, BatchFolderName As String, ApplyExceptionHandling As Boolean)`

Parameter	Description
`pWorkdoc`	Reference to the currently processed document.
`Reason`	Exception reason. Possible value: `CDRVerifierExceptionReason`

`CreateNewBatch`	`True`: Create a new exception batch. `False`: Do not create a new exception batch.
`BatchName`	Name of the new exception batch.
`BatchDocumentState`	State of the new exception batch.
`BatchPriority`	Priority of the new exception batch.
`BatchFolderName`	Name of the folder in the exception batch.
`ApplyExceptionHandling`	`True`: Move the document/batch to exception. `False`: Do not move the document/batch to exception.

1.2.3.21 VerifierFormLoad

This event is triggered before the Verifier form is loaded. It enables the script 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. Refer to the *DisplayPage* feature for additional information. It can also be used to modify the form before it gets displayed to the user. This event is not triggered in the Designer application.

You can use the VerifierFormLoad event for different purposes, including the following examples:

- To switch verification forms between different types of classes.
- To optionally load a non-standard verification form in accordance with some parameters of the processed document.
- To translate the content of verification forms dynamically into a language different from Windows Region and Language settings.
- To load the required verification form according to the Windows Region and Language settings of the current system.
- To default Verifier to display a specific page of a document instead of the first one.
- To modify the form before it displays to the user

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

Parameter	Description
`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 WebCenter Forms Recognition custom script to point to a different document class, in case the desired verification form is located in this different class.

1.2.3.21.1 Sample Code

The following code example demonstrates how to replace the standard class name and the form name that Verifier loads.

Note: If the VerifierFormLoad event script handler is not implemented or the script assigns a non-existing class name or a non-existing form name, the application loads the standard verification form.

```
Private Sub ScriptModule_VerifierFormLoad(pWorkdoc as
SCBCdrPROJLib.SCBCdrWorkdoc, FormClassName as String, FormName as String)
    Select Case UCase(FormClassName)
       Case "BASH"
          FormClassName = "Invoices"
          FormName = "Form_Invoices_2"
       Case "CONTACT"
          FormClassName = "Invoices"
          FormName = "Form_Invoices_1"
    End Select
End Sub
```

For additional information, see Implement an Event, DisplayPage, Invisible.

1.3 Document Events

1.3.1 DocClass Event Interface

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

1.3.1.1 FocusChanged

This event is fired each time 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 a specified field instead of the originally selected field.

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

Parameter	Description
`pWorkdoc`	Reference to the currently displayed workdoc.
`Reason`	Reason (`CdrFocusChangeReason`) of the current focus change, which can be: ▪ **[Tab]** key. ▪ **[Enter]** key. ▪ Mouse click. ▪ Initial loading.
`OldFieldIndex`	Index of the current select field. In case of initial loading this **-1**.

pNewFieldIndex	Index of the field that should be selected now. This parameter can be modified during the script event to keep the focus in the previous field or set it to another field.

1.3.1.1.1 Sample Code

The following script example demonstrates how to skip the table data validation in Verifier for a table with two columns.

```
Private Sub Document_FocusChanged(pWorkdoc as SCBCdrWorkdoc, Reason as
CdrFocusChangeReason, OldFieldIndex as Long, pNewFieldIndex as Long)
    Dim theEmptyTable as SCBCdrPROJLib.SCBCdrTable
    Dim theEmptyTableField as SCBCdrPROJLib.SCBCdrField
    'Initializes table and field references
    Set theEmptyTable = _

pWorkdoc.Fields("EmptyTable").Table(pWorkdoc.Fields("EmptyTable").ActiveTableInd
ex)
    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
```

1.3.1.1.2 Sample Code

To trigger the event before the assigned verification form loads, but after the VerificationFormLoad event, enable the "Allow firing of FocusChanged event when loading the verification form" in Designer. See *Specify compatibility settings* in the *WebCenter Forms Recognition Designer Guide*.

The following sample code shows how to implement the handler for the `CdrBeforeFormLoaded` reason:

```
Private Sub Document_FocusChanged(pWorkdoc as SCBCdrPROJLib.SCBCdrWorkdoc, ByVal
Reason as SCBCdrPROJLib.CdrFocusChangeReason, ByVal OldFieldIndex as Long,
pNewFieldIndex as Long)
    If Reason = CdrBeforeFormLoaded Then
       ' Do something...
    End If
End Sub
```

1.3.1.2 OnAction

This event is triggered if any of the configured actions was caused by the user. Actions have to be configured in Designer in *Verifier Design Mode*. Actions can either be caused if a user pressed a button or any of the configured keyboard short cuts.

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

Parameter	Description

pWorkdoc	Reference to the currently displayed workdoc.
ActionName	Name of the action that was assigned to the pressed button or short cut key.

1.3.1.2.1 Sample Code

```
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
```

1.3.1.3 PostExtract

The *PostExtract* event is called after all defined analysis or evaluation methods are executed by the document class. During this event, it is possible to examine and change the results of one or more fields of the document.

You can also use this event 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 to a different class.

Syntax: `Document_PostExtract(pWorkdoc As ISCBCdrWorkdoc)`

Parameter	Description
pWorkdoc	Reference to the current workdoc object.

1.3.1.3.1 Sample Code

```
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
```

1.3.1.4 PreExtract

The *PreExtract* event is called before any defined analysis or evaluation method is executed by the document class.

Syntax: `Document_PreExtract(pWorkdoc As ISCBCdrWorkdoc)`

Parameter	Description
pWorkdoc	Reference to the current workdoc object.

1.3.1.4.1 Sample Code

```
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
```

1.3.1.5 PreVerifierTrain

This event is called at the point when an application starts learning for a document in the Supervised Learning Workflow.

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

Parameter	Description
`pMode`	*Reserved for future use.*

1.3.1.5.1 Sample Code

The following script example demonstrates how the new script event can be used to apply a substitution of the primary Associative Search 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
```

1.3.1.6 Validate

Use the *Validate* event to perform validation at document level. At this point, the validation of all single fields is executed. If one of the fields is still invalid, *pValid* is **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. 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)`

Parameter	Description
`pWorkdoc`	Reference to the current workdoc object.

pValid	Parameter containing the current valid state of the Workdoc.

1.3.1.6.1 Sample Code

```
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
```

1.3.1.7 VerifierTrain

After a document processed in Verifier has been checked to see whether it should 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)`

Parameter	Description
pWorkdoc	Reference to the current workdoc object.
ProposedClassName	The proposed class name.
WillTrain	Boolean value for the current learning state. **True** when the document is going to be learned and **False** when it will not be learned.
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.

1.3.2 <Field$_n$> FieldDef Event Interface

Field events are specific for each field of each document class. Field events appear within the script sheet of their document class. That means all events for the field **Number** of the document class *Invoice* must be implemented within the script sheet of the document class **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, **<Field$_n$>** is used as a placeholder for the name of the field. The *Validate* event is named here as **<Field$_n$>_Validate**.

1.3.2.1 CellChecked

This event occurs when a checkbox cell of the table is checked or unchecked by the user.

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

Parameter	Description
`pTable`	Current table object.
`pWorkdoc`	Reference to the 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**.

1.3.2.1.1 Sample Code

```
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
```

1.3.2.2 CellFocusChanged

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

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

Parameter	Description
`pTable`	Current table object.
`pWorkdoc`	Reference to the current workdoc object.
`Reason`	Parameter that contains the kind of focus change that has occurred. `CdrTableFocusChangeReason`

`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 (set back to *OldRow* value), to forbid an action, such as double-clicking on the special column.
`pNewColumn`	This parameter contains the index of the destination column. This value can be changed (set back to *OldColumn* value), to forbid an action, such as double-clicking on the special column.

1.3.2.2.1 Sample Code

```
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, 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 sets the 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 program 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
```

1.3.2.3 Format

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 field's content, use the *FormatForExport* event.

Syntax: `<Field_n>_Format(pField As ISCBCdrField)`

Parameter	Description
`pField`	Reference to the field object.

1.3.2.3.1 Sample Code

```
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
```

1.3.2.4 FormatForExport

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: `<Field_n>_FormatForExport(pField As ISCBCdrField)`

Parameter	Description
`pField`	Reference to the field object.

1.3.2.4.1 Sample Code

```
Private Sub Amount_FormatForExport(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. Then
```

```
use pField.FormattedText also for the Export, instead of pField.Text
       pField.FormattedText = NewAmount
    End if
End Sub
```

1.3.2.5 PostAnalysis

The `PostAnalysis` event is called after the analysis step is performed. It is possible to examine the list of all candidates and to add further candidates to the field.

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

Parameter	Description
`pField`	Reference to the field object.
`pWorkdoc`	Reference to the current workdoc object.

1.3.2.5.1 Sample Code

```
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
```

1.3.2.6 PostEvaluate

The `PostEvaluate` event is called after the evaluation step is performed. It is possible to examine the list of all candidates and to change their weights.

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

Parameter	Description
`pField`	Reference to the field object.
`pWorkdoc`	Reference to the current workdoc object.

1.3.2.6.1 Sample Code

```
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
```

1.3.2.7 PreExtract

The `PreExtract` event is called before any defined analysis or evaluation method for this field is executed by the document class.

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

Parameter	Description
`pField`	Reference to the field object.
`pWorkdoc`	Reference to the current workdoc object.

1.3.2.7.1 Sample Code

```
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
```

1.3.2.8 SmartIndex

The `SmartIndex` event is be called for the field where the smart indexing was defined. This field usually provides the key for the *SELECT* statement.

Syntax: `<Field_n>_SmartIndex(pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc)`

Parameter	Description
`pField`	Reference to the field object.
`pWorkdoc`	Reference to the current workdoc object.

1.3.2.8.1 Sample Code

```
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
```

1.3.2.9 TableHeaderClicked

This event occurs when a user clicks on one of the table header buttons. There are three different table header buttons:

- Row header button.
- Column header button.

- Table header button.

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

Parameter	Description
`pTable`	Current table object.
`pWorkdoc`	Reference to the current workdoc object.
`ClickType`	The click type of the mouse depends on the place where the click occurred either for the column header, row header, or table header, and the type of click that occurred, such as a single click, double-click, or right-click. `CdrTableHeaderClickType`
`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`.

1.3.2.9.1 Sample Code

```
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
```

1.3.2.10 Validate

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

Syntax: `<Field_n>_Validate(pField As ISCBCdrField, pWorkdoc As ISCBCdrWorkdoc, pValid As Boolean)`

Parameter	Description
`pField`	Reference to the field object.
`pWorkdoc`	Reference to the current workdoc object.
`pValid`	Boolean parameter containing the current valid state of the field.

1.3.2.10.1 Sample Code

```
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
```

1.3.2.11 ValidateCell

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

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

Parameter	Description
`pTable`	Current table object.
`pWorkdoc`	Reference to the current workdoc object.

Row	The row location of the cell to be validated.
Column	The column location of the cell to be validated.
pValid	Boolean parameter containing the current valid state of the table cell.

1.3.2.11.1 Sample Code

```
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
```

1.3.2.12 ValidateRow

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

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

Parameter	Description
pTable	Current table object.
pWorkdoc	Reference to the current workdoc object.
Row	The given row of the table to be validated.
pValid	Boolean parameter containing the current valid state of the table row.

1.3.2.12.1 Sample Code

```
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
```

1.3.2.13 ValidateTable

Implements a validation rule for the entire table.

Syntax: `<Field_n>_ValidateTable(pTable As ISCBCdrTable, pWorkdoc As ISCBCdrWorkdoc, pValid As Boolean)`

Parameter	Description
`pTable`	Current table object.
`pWorkdoc`	Reference to the current workdoc object.
`pValid`	Boolean parameter containing the current valid state of the table.

1.3.2.13.1 Sample Code

```
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
```

2 Workdoc Object Reference (SCBCdrWorkdocLib)

2.1 SCBCdrWorkdoc

2.1.1 Description

The `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

2.1.2.1 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.

2.1.2.2 CDRDocFileType

This data type is the enumeration that contains 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.

2.1.2.3 CDRDocState

This definition 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.

2.1.2.4 CdrEdgeSide

This is the definition that determines the type of alignment or edges.

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

2.1.2.5 CdrExportType

This data type determines which data from the current document will export. It is used in the method *ExportToXML*.

Available Types	Description
CDRExportTypeOCRData	Exports OCR data of words and characters of a Workdoc.

2.1.2.6 CDRFieldState

This enumeration contains 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.

2.1.2.7 CDRHighlightMode

This definition is the highlighting mode for the workdoc that displays for the user, such as highlight candidates, highlight fields only, and so on.

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.
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 tooltips for all words and fires all mouse events for all words.
CDRHighlightTextLines	Text lines highlighting.

CDRHighlightTextLinesAdvanced	Highlights text lines according their block number, show tooltips 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.

2.1.2.8 CDRLocation

This table lists the enumerations that contain the location of a row, column, or cell in a table.

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

2.1.2.9 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 CI Doc to workdoc.
CDRPageAssignNewPage	First Page of Image or CI Doc appended as last DocPage to workdoc.
CDRPageAssignNoPage	No DocPages assigned to workdoc.

2.1.2.10 CDRPageSource

The following table shows the enumeration that contains 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.

2.1.2.11 CDRPDFExportStyle

This data type is responsible for specifying the export type of PDF image out of WebCenter Forms Recognition.

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.

2.1.2.12 CDRTableHighlightMode

This lists the enumerations that contains 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.

2.1.2.13 CDRTranslationLanguage

This type defines the transliteration approach.

Available Types	Description
CDRTranslationLanguageDefault	Internal language translation is turned off.
CDRTranslationLanguageGreek	One to one character translation type representing former non-western languages support approach.
CDRTranslationLanguageRussian	Translation through extended CJKT methods, but leaving all one to many chars unchanged.
CDRTranslationLanguageCJKT	Full CJKT type of language translation with one to many encoding for both CJKT and non-western languages.
CDRTranslationLanguageCombined	Transformation for all Unicode characters.

2.1.2.14 CroLinesDir

This table shows the enumeration that specifies the direction of a line.

Available Types	Description
CroLinesDir_Horizontal	Horizontal line.
CroLinesDir_Vertical	Vertical line.

2.1.2.15 CroLinesKooType

This table shows the enumeration that specifies coordinate types for 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.

2.2 SCBCdrCandidate

2.2.1 Description

Candidates are generated during the analysis step and represent possible results of a field.

2.2.2 Methods and Properties

2.2.2.1 Attractor

This property returns the attractor of the candidate by a zero-based index.

Syntax: `Attractor(Index As Long) As ISCBCdrAttractor`

Parameter	Description
`Index`	Specifies the zero-based index in the attractor array. This value must be between **0** and *AttractorCount*-1

2.2.2.2 AttractorCount

This property returns the number of attractors for this candidate.

Syntax: `AttractorCount As Long`

2.2.2.3 CopyToField

Use this method to copy all required properties from the candidate to the field result.

Syntax: `CopyToField(pField As ISCBCdrField)`

Parameter	Description
`pField`	Reference to the field object containing the candidate. States which field should get the values from the candidate.

2.2.2.4 FilterID

This is the *FilterID* value as it was specified by the *AddCandidate* method of the field.

Syntax: `FilterID As Long`

2.2.2.4.1 Sample Code

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

2.2.2.5 FormatConfidence

This property sets or returns the confidence of the string match algorithm performed by the format search engine that has created the candidate.

Syntax: `FormatConfidence As Double`

2.2.2.6 Height

This property returns the height of the candidate in pixels.

Syntax: `Height As Long`

2.2.2.7 KeepSpaces

This property specifies if the text created from several words should keep the spaces between these words or not.

Syntax: `KeepSpaces As Boolean`

2.2.2.8 Left

This read-only property returns the left border of the candidate in pixels.

Syntax: `Left As Long`

2.2.2.9 Line

This read-only property returns the text of a single line. A candidate can consist of one or more lines.

Syntax: `Line(Index As Long) As String`

Parameter	Description
`Index`	The index of the line. This value must be between **0** and *LineCount*-1.

2.2.2.10 LineCaption

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`

Parameter	Description
`Index`	The index of the line. This value must be between **0** and *LineCount*-1.

2.2.2.11 LineCount

This property 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`

2.2.2.12 LineWordCount

This read-only property returns the number of words of the specified line.

Syntax: `LineWordCount(Index As Long) As Long`

Parameter	Description
`Index`	The index of the line. This value must be between **0** and *LineCount*-1.

2.2.2.13 LineWordID

This read-only property returns the word ID of the specified line and word index.

Syntax: `LineWordID(LineIndex As Long, WordIndex As Long) As Long`

Parameter	Description
`LineIndex`	The index of the line. This value must be between **0** and *LineCount*-1.
`WordIndex`	The index of the word within the line.

2.2.2.14 LineWorktext

This property returns the *Worktext* object of the single line specified by the zero-based index within a multiline field.

Syntax: `LineWorktext(Index As Long) As ISCBCroWorktext`

Parameter	Description
`Index`	The index of the line. This value must be between **0** and *LineCount*-1.

2.2.2.15 PageNr

This read-only property returns the *DocPage* number where the candidate is located.

Syntax: `PageNr As Long`

2.2.2.15.1 Sample Code

```
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
```

2.2.2.16 RemoveAttractor

This method removes the attractor specified by index.

Syntax: `RemoveAttractor(AttractorIndex As Long)`

Parameter	Description

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

2.2.2.17 Text

This read-only property returns the text of the candidate.

Syntax: `Text As String`

2.2.2.18 Top

This property returns the top border of the candidate in pixels.

Syntax: `Top As Long`

2.2.2.19 Weight

This property sets or 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`

2.2.2.20 Width

This read-only property returns the width of the candidate in pixels.

Syntax: `Width As Long`

2.2.2.21 WordCount

This read-only property returns the word count of the candidate.

Syntax: `WordCount As Long`

2.2.2.22 WordID

This read-only property returns the word ID of the specified word index within the first line.

Syntax: `WordID(Index As Long) As Long`

Parameter	Description
`Index`	Zero-based index of the word within the line.

2.2.2.23 Worktext

This read-only property returns the *Worktext* object of the first line.

Syntax: `Worktext As ISCBCroWorktext`

2.3 SCBCdrDocPage

2.3.1 Description

An object that represents a single document page within a workdoc.

2.3.2 Methods and Properties

2.3.2.1 DisplayImage

This property specifies the index of the image, which is displayed if the DocPage is visible inside the viewer.

Syntax: `DisplayImage As Long`

2.3.2.2 DocIndex

This property specifies the index of the document inside the workdoc to which this DocPage belongs.

Syntax: `DocIndex(ImageIndex As Long) As Long`

Parameter	Description
`ImageIndex`	Image index of the DocPage. Valid indices are **0** to *ImageCount*-1.

For additional information, refer to *DocFileName* and *DocFileType* properties of the *SCBCdrWordoc* object.

2.3.2.3 DocPageIndex

This read-only property specifies the DocPage offset inside the document where this DocPage belongs.

Syntax: `DocPageIndex(ImageIndex As Long) As Long`

Parameter	Description
`ImageIndex`	Image index of the DocPage. Valid indices are **0** to *ImageCount*-1.

2.3.2.4 GetResolution

This method returns the resolution of the specified image in pixels.

Syntax: `GetResolution(ImageIndex As Long, pXRes As Long, pYRes As Long)`

Parameter	Description
`ImageIndex`	Image index of the DocPage. Valid indices are **0** to *ImageCount*-1.
`pXRes`	Returns the *x* resolution after execution of the method.

pYRes	Returns the *y* resolution after execution of the method.

2.3.2.5 Height

This read-only property returns the height of the DocPage in millimeters.

Syntax: `Height As Double`

2.3.2.6 Image

This read-only property returns an image object for the specified index of the DocPage.

Syntax: `Image(Index As Long) As ISCBCroImage`

Parameter	Description
Index	Image index of the DocPage. Valid indices are **0** to *ImageCount*-1.

2.3.2.7 ImageCount

This read-only property returns the number of images available for the DocPage.

Syntax: `ImageCount As Long`

2.3.2.8 Line

This read-only property returns some specific property of a line, of some specific index, direction and coordinate type.

Syntax: `Line(LineIndex As Long, LineDir As CroLinesDir, KooType As CroLinesKooType) As Long`

Parameter	Description
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.)

2.3.2.9 LinesCount

This read-only property returns the number of horizontal or vertical lines present in a document.

Syntax: `LinesCount(LinesDir As CroLinesDir) As Long`

Parameter	Description
LinesDir	Direction of line (horizontal or vertical).

2.3.2.10 OriginalDocumentFileName

This property allows the project developer to access the page property to examine the original file name for the image. This is useful when attempting to track original file names for pages when a document is split or merged through Verifier or Web Verifier, or through the Page Separation engine.

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

2.3.2.10.1 Sample Code

```
Private Sub CreateCollectionofPageOrgFileName(pWorkdoc as
SCBCdrPROJLib.ISCBCdrWorkdoc)
    Dim WdcPageCount as Long ' Total Number of pages associated to the WorkDoc
    Dim CurPage as Long ' Current Page Number
    Dim OrgFilename as String ' Original File Name of the selected page
    Dim OrgFilenames() as String ' Array of Original File Name of all Pages of
the WorkDocument
    WdcPageCount = pWorkdoc.PageCount
    ReDim OrgFilenames(WdcPageCount)
    For CurPage=0 To WdcPageCount-1
       OrgFilenames(CurPage) = pWorkdoc.Pages(CurPage).OriginalDocumentFileName
    Next CurPage
    ' Write the original file name of all pages to log.
    For CurPage=0 To WdcPageCount-1
       OrgFilename = OrgFilenames(CurPage)
       Project.LogScriptMessageEx CDRTypeInfo, CDRSeverityLogFileOnly, "
Original File Name of Page: " & CStr(CurPage+1) & " is [" & OrgFilename & "]"
    Next CurPage
End Sub
```

2.3.2.11 PageSource

This property sets or returns a source of a DocPage. At the time of scanning, a DocPage can be directly assigned to workdoc.

Syntax: `PageSource As CDRPageSource`

2.3.2.12 Rotate

This method rotates the underlying images by the specified angle.

Syntax: `Rotate(Angle As Double)`

Parameter	Description
`Angle`	Specifies the rotation angle in a range of **-180.0** to **180.0**.

2.3.2.13 Rotation

This read-only property returns the rotation angle as it was applied by the *Rotate* method.

Syntax: `Rotation As Double`

2.3.2.14 Text

This read-only property returns the text of the DocPage if OCR was already executed.

Syntax: `Text As String`

2.3.2.15 Width

This read-only property returns the width of the DocPage in millimeters.

Syntax: `Width As Double`

2.4 SCBCdrEmailProperties

2.4.1 Description

When importing a `.msg` file into a workdoc, the most important properties of the email are stored in the workdoc and available in the custom script via the `ISCBCdrEmailProperties` interface, that can be queried from the `SCBCdrWorkdoc` interface.

2.4.2 Methods and Properties

2.4.2.1 CC

The List of carbon copy recipients.

2.4.2.2 From

The List of email senders.

2.4.2.3 MessageID

Unique message identifier.

2.4.2.4 Priority

Priority of the email was sent with.

2.4.2.5 Received

Date and time the email was received.

2.4.2.6 Sent

Date and time the email was sent.

2.4.2.7 Subject

Subject of the email.

2.4.2.8 To

List of email recipients.

2.5 SCBCdrField

2.5.1 Description

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

2.5.2 Methods and Properties

2.5.2.1 ActiveTableIndex

This property reads the position where the table is activated, or activates the table at given zero-based index.

Syntax: `ActiveTableIndex As Long`

2.5.2.1.1 Sample Code

```
'Initializes table and field references
 Set theEmptyTable = _

pWorkdoc.Fields("EmptyTable").Table(pWorkdoc.Fields("EmptyTable").ActiveTableInd
ex)
 Set theEmptyTableField = pWorkdoc.Fields("EmptyTable")
```

2.5.2.2 AddCandidate

This method 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)`

Parameter	Description
`WordNr`	Specifies the word index within the word array of the workdoc. Must be within **0** to *pWorkdoc.WordCount* - 1.
`WordCount`	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, etc.
`FilterID`	This parameter can be used to store a filter identifier inside the candidate. Later it is possible to see which filter expression has created the candidate.
`pIndex`	Returns the index of the new candidate within the candidate array.

2.5.2.2.1 Sample Code

```
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
```

2.5.2.3 AddCandidate2

This method adds a new candidate to the field, based on the specified *pWorktext*.

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

Parameter	Description
`pWorktext`	Must be an initialized *Worktext* object as it was created calling a *SCBCroZone.Recognize* method.
`pIndex`	Returns the index of the new candidate within the candidate array.

2.5.2.4 AddTable

This method adds a table into the table array of this field.

Syntax: `AddTable()`

2.5.2.5 BoostDigitsOnly

This property sets or returns whether only digits should be boosted.

Syntax: `BoostDigitsOnly As Boolean`

2.5.2.6 BoostField

This property sets or returns whether a field should be boosted.

Syntax: `BoostField As Boolean`

2.5.2.7 Candidate

This read-only property returns a candidate of the field.

Syntax: `Candidate(Index As Long) As` *`ISCBCdrCandidate`*

Parameter	Description
`Index`	Specifies the index in the attractor array, must be between **0** and *AttractorCount* - 1.

2.5.2.8 CandidateByFilterID

This method finds the first candidate by specified *FilterID*, or creates a new one if no such candidate is found.

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

Parameter	Description
FilterID	The filter ID that is used to find the candidate.
CreateNew	Create a new candidate if set to **True**.
pCandidateIndex	The index of the found candidate.

2.5.2.9 CandidateCount

This read-only property returns the number of candidates for a field.

Syntax: `CandidateCount As Long`

2.5.2.10 Changed

This property returns the changed state of the field. If the changed state becomes **True**, the field must be validated even if it was previously validated.

Syntax: `Changed As Boolean`

2.5.2.11 CustomDetailsString

This property sets or returns the *CustomDetailsString*.

Syntax: `CustomDetailsString As String`

2.5.2.12 CustomStatusLong

This property sets or returns the *CustomStatusLong*.

Syntax: `CustomStatusLong As Long`

2.5.2.13 DeleteLine

This method deletes a line from a specific index position.

Syntax: `DeleteLine(LineIndex As Long)`

Parameter	Description
LineIndex	Zero-based index of the line to be deleted.

2.5.2.13.1 Sample Code

```
'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"
```

2.5.2.14 DeleteTable

This method deletes a table from the table array of this field.

Syntax: `DeleteTable(TableIndex As Long)`

Parameter	Description
`TableIndex`	Zero-based index of the table to be deleted.

2.5.2.15 ErrorDescription

This property stores the reason if a script validation could not be performed successfully.

Syntax: `ErrorDescription As String`

2.5.2.15.1 Sample Code

```
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
```

2.5.2.16 ExternalText

This property sets or returns the extended text.

Syntax: `ExternalText As String`

2.5.2.17 FieldID

This read-only property returns the internally used field ID.

Syntax: `FieldID As Long`

2.5.2.18 FieldState

This property sets or returns the current execution state of the field.

Syntax: `FieldState As` *`CDRFieldState`*

2.5.2.18.1 Sample Code

```
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
```

2.5.2.19 FieldVersion

This property returns the field data of the specified version.

Syntax: `FieldVersion As String (ByVal Index As Long)`

Parameter	Description
`Index`	The zero-based index of the field.

2.5.2.20 FindCandidate

This method searches inside the list of candidates if there is a candidate based on the specified word ID.

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

Parameter	Description
`WordID`	Specifies a word ID inside the word array of the workdoc searched for.
`pCandIndex`	Contains the index of the candidate if one was found, or **-1** if no candidate was found.

2.5.2.21 FindCandidateByPos

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`*

Parameter	Description
`Page`	Long
`Param1`	Long
`Left`	Long
`Top`	Long
`Width`	Long

`Height`	Long
`CandidateIndex`	Contains the index of the candidate if one was found, or **-1** if no candidate was found.

2.5.2.22 FormattedText

This property can be set only in the *FormatForExport* field event. It is emptied before validate field event.

2.5.2.23 GetFirstCandidatePropsByPage

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
```

Parameter	Description
`Page`	Long
`Param1`	Long
`Left`	Left position of area in pixels.
`Top`	Top of area in pixels.
`Width`	Width of area in pixels.
`Height`	Height of area in pixels.
`Text`	String
`Weight`	Double

2.5.2.24 GetNextCandidatePropsByPage

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
```

Parameter	Description
`Left`	Left position of area in pixels.
`Top`	Top of area in pixels.

`Width`	Width of area in pixels.
`Height`	Height of area in pixels.
`Text`	Left position of area in pixels.
`Weight`	Double

2.5.2.25 GetUniqueEntryID

This method retrieves other column values for the specified pool entry.

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

Parameter	Description
`IdHigh`	Returns the upper part of the 64-bit unique ID.
`IdLow`	Returns the lower part of the 64-bit unique ID.

2.5.2.26 Height

This property sets or returns the height of the field in pixels

Syntax: `Height As Long`

2.5.2.26.1 Sample Code

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

2.5.2.27 InsertLine

This method inserts a line at the given *LineIndex* in a field.

Syntax: `InsertLine(LineIndex As Long)`

Parameter	Description
`LineIndex`	Zero-based line index at which position line should be inserted.

2.5.2.27.1 Sample Code

```
'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" does not work.
```

2.5.2.28 IsIDAlphNum

Sets or returns whether an Associative Search Engine's unique ID is alphanumeric. If **True**, then the field is alphanumeric; if **False** than the field is numeric.

When accessing this attribute from the *CdrWorkDoc* object, the property is taken from the Associative Search Engine configured for the *Classification Field*.

When accessing this attribute from the *CdrField* Object, the property is taken directly from the Associative Search Engine field for the class.

In some, complex, project configurations, the following considerations may apply where direct access to fields is needed, to look directly at the Associative Search Engine field attribute rather than the workdoc.

- When the project hierarchy has a parent class where the *Classification Field* **UniqueID** is of type **A** (e.g. alphanumeric), but the same field on a child class is of type **B** (e.g. numeric).

  In this instance accessing the *workdoc.IsIDAlphNum* will always return the parent setting, thus requiring the project developer to access the field property directly.
- When the project has many Associative Search Engine fields, and the *IsIDAlphNum* is being retrieved or set.

Syntax: `IsIDAlphNum As Boolean`

2.5.2.28.1 Sample Code

```
Dim pFieldDef as SCBCdrFieldDef
 Dim pSettings as SCBCdrSupExSettings
 Dim bIsAlphNum as Boolean
 Set pFieldDef = Project.AllClasses(pWorkdoc.DocClassName).Fields("MyASSA")
 Set pSettings = pFieldDef.AnalysisSetting(Project.DefaultLanguage )
 bIsAlphNum = pSettings.IsIDAlphNum
```

2.5.2.29 LastModificationEndDate

This property sets or returns the date the field was last modified.

Syntax: `LastModificationEndDate As Date`

2.5.2.30 LastModificationEndDateAsFileTimeUtc

This property sets or returns the date the field was last modified in UTC format.

Syntax: `LastModificationEndDateAsFileTimeUtc As Date`

2.5.2.31 Left

This property sets or returns the left border of the field in pixels.

Syntax: `Left As Long`

2.5.2.32 Line

This property sets or returns the text of a single line.

Syntax: `Line(Index As Long) As String`

Parameter	Description
`Index`	The index value of the line. Must be from **0** to *LineCount*-1.

2.5.2.33 LineCaption

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`

Parameter	Description
`Index`	The index value of the line. Must be from **0** to *LineCount*-1.

2.5.2.34 LineCount

This property returns the number of lines of a multi-line header field. This equals the number of *Worktext* objects.

Syntax: `LineCount As Long`

2.5.2.35 LineWorktext

This property 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`

Parameter	Description
`Index`	The index value of the line. Must be from **0** to *LineCount*-1.

2.5.2.36 MultilineText

This property sets or returns multiline text for all lines at once, separated with line break characters.

Syntax: `MultilineText As String`

2.5.2.37 Name

This read-only property returns the name of the field as it was defined within the design environment.

Syntax: `Name As String`

2.5.2.38 PageNr

This property sets or returns the *DocPage* number where the field is located.

Syntax: `PageNr As Long`

2.5.2.39 PutUniqueEntryId

This method sets the 64-bit unique ID for the field content from associative search pool.

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

Parameter	Description
`IdHigh`	The upper part of the 64-bit unique ID.
`IdLow`	The lower part of the 64-bit unique ID.

2.5.2.39.1 Sample Code

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

2.5.2.40 RemoveCandidate

This method removes a candidate from the candidate array.

Syntax: `RemoveCandidate(CandIndex As Long)`

Parameter	Description
`CandIndex`	Zero-based index of the candidate to be removed.

2.5.2.41 SkipTrainingWithEngine

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

Syntax: `SkipTrainingWithEngine(bstrEngineName As String) As Boolean`

Parameter	Description
`bstrEngineName`	The name of the extraction engine.

2.5.2.42 SortCandidatesByWeight

This method sorts evaluated field candidates by their weight.

Syntax: `SortCandidatesByWeight(Ascending as Boolean)`

Parameter	Description
`Ascending`	True: Sort the candidates in ascending order which means by descending weight value. False: Sort the candidates in descending order which means by ascending weight value.

2.5.2.43 Table

This property returns the table object from an array of tables of this field at a specified index.

Syntax: `Table(Index As Long) As ISCBCdrTable`

Parameter	Description
`Index`	Zero-based index of the table in the array.

2.5.2.44 TableCount

This read-only property returns the number of tables according to the field.

Syntax: `TableCount As Long`

2.5.2.45 Tag

Use this property to store an arbitrary variant in the field.

Syntax: `Tag As Variant`

2.5.2.46 Text

Use this property to read and write the text of the field. In case of multiline fields, the *Text* property refers to all lines at once as one single string, combining lines with spaces in between.

Syntax: `Text As String`

2.5.2.47 Top

This property sets or returns the top border of the field in pixels.

Syntax: `Top As Long`

2.5.2.48 TrainedWithEngine

This read-only property returns whether this field is trained with the specified engine.

Syntax: `TrainedWithEngine(bstrEngineName As String) As Boolean`

Parameter	Description
`bstrEngineName`	The name of the engine.

2.5.2.49 UniqueId

This property sets or returns the unique alphanumeric Id for the field content from the associative search pool.

Syntax: `UniqueId As String`

2.5.2.50 Valid

This property sets or returns the valid state of the field.

Syntax: `Valid As Boolean`

2.5.2.51 Width

This property sets or returns the width of the field in pixels.

Syntax: `Width As Long`

2.5.2.52 Worktext

This property provides access to the *Worktext* of the field. In case of multiline fields, the *Worktext* property refers to the first *Worktext* the header field consists of, which represents the first line of the multiline header field.

Syntax: `Worktext As ISCBCroWorktext`

2.5.2.53 XmlExportEnabled

This property sets or returns whether the field is included in the exported XML file.

Set the property to `False` to exclude the field from the exported file.

The default value is `True`.

Syntax: `XmlExportEnabled As ISCBCroWorktext`

2.5.2.53.1 Sample Code

The following sample code disables the XML export of field data for field names that start with `.tmp`.

```
Dim i As Long
 Dim currentField As ISCBCdrField
 For i = 1 To pWorkdoc.Fields.Count
    Set currentField = pWorkdoc.Fields.ItemByIndex(i)
    If Left$(currentField.Name,3) = "tmp" Then
       currentField.XmlExportEnabled = False
    End If
 Next i
```

2.6 SCBCdrFields

2.6.1 Description

This is a collection of all `SCBCdrField` objects contained in the current workdoc object.

2.6.2 Methods and Properties

2.6.2.1 Add

This method adds a new field with the specified name to the collection.

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

Parameter	Description
`NewItem`	Pointer to a *SCBCdrField* object that should be added to the collection.
`ItemName`	Name of the field item inside the collection. This name must be used to access the item inside the collection.

2.6.2.2 Clear

This method removes all items from the collection and releases their reference count.

Syntax: `Clear()`

2.6.2.3 Collection

This read-only property returns the collection that is internally used to store the fields.

Syntax: `Collection As ISCBCroCollection`

2.6.2.4 Count

This read-only property returns the number of items within the field collection.

Syntax: `Count As Long`

2.6.2.5 Item

This read-only property returns a specified item from the collection. The *Item* property is the default property of the *ISCBCdrFields* collection.

Syntax: `Item(Index As Variant) As ISCBCdrField`

Parameter	Description
`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.

2.6.2.6 ItemByIndex

This read-only property returns an item from the collection specified by the index.

Syntax: `ItemByIndex(Index As Long) As ISCBCdrField`

Parameter	Description
`Index`	The index of the item to retrieve from the collection. Valid range from 1 to *Count*.

2.6.2.6.1 Sample Code

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

2.6.2.7 ItemByName

This read-only property returns the field from the collection by the specified field name.

Syntax: `ItemByName(Name As String) As ISCBCdrField`

Parameter	Description
`Name`	The name of the item to retrieve from the collection.

2.6.2.7.1 Sample Code

```
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.ItemByName("LineItem
s").AllowInteractiveExtraction = False
    Else

Project.AllClasses.ItemByName(pWorkdoc.DocClassName).Fields.ItemByName("LineItem
s").AllowInteractiveExtraction = True
    End If
End Sub
```

2.6.2.8 ItemExists

This method returns **True** if an item with the specified name exists inside the collection, otherwise **False** is returned.

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

Parameter	Description
`Name`	The name of the item for which to search.

2.6.2.9 ItemIndex

This read-only property returns the index of an item specified by name.

Syntax: `ItemIndex(Name As String) As Long`

Parameter	Description
`Name`	Name specifying an item in the collection.

2.6.2.10 ItemName

This read-only property returns then name of an item specified by index.

Syntax: `ItemName(Index As Long) As String`

Parameter	Description
`Index`	Index specifying an item in the collection. Valid range from 1 to *Count*.

2.6.2.11 MoveItem

This method moves an item specified by *OldIndex* from *OldIndex* to *NewIndex*.

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

Parameter	Description
`OldIndex`	Index of item to remove. Valid range from 1 to *Count*.
`NewIndex`	New index of the item after the move has occurred. Valid range from 1 to *Count*.

2.6.2.12 Remove

This method removes the specified item from the collection and releases the reference count to this item.

Syntax: `Remove(ItemName As String)`

Parameter	Description
`ItemName`	Name of the item to remove.

2.6.2.13 RemoveByIndex

This method removes the specified item from the collection and releases the reference count to this item.

Syntax: `RemoveByIndex(Index As Long)`

Parameter	Description
`Index`	Index of the item to remove. Valid range from 1 to *Count*.

2.6.2.14 Rename

This method renames the item specified by *OldName* from *OldName* to *NewName*.

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

Parameter	Description
`OldName`	Name of item to rename.
`NewName`	New name of item in collection.

2.6.2.15 Tag

This property stores a variant for each item of the collection.

Syntax: `Tag(Index As Long) As Variant`

Parameter	Description
`Index`	Specifies the item index. Valid range from 1 to *Count*.

2.7 SCBCdrFolder

2.7.1 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 the script.

2.7.2 Methods and Properties

2.7.2.1 AddDocument

This method adds a workdoc into a folder at the last position and returns the position where the workdoc is appended.

Syntax: `AddDocument(pWorkdoc As ISCBCdrWorkdoc, pNewIndex As Long)`

Parameter	Description
`pWorkdoc`	Added workdoc object.
`pNewIndex`	Index position in the folder where `pWorkdoc` is inserted.

2.7.2.2 Clear

This method frees all the allocated memory by the folder object.

Syntax: `Clear()`

2.7.2.3 Document

This read-only property returns a workdoc object from the specified index of the document array of the folder.

Syntax: `Document(Index As Long) As ISCBCdrWorkdoc`

Parameter	Description
`Index`	The index of the workdoc within the folder. Must be from **0** to *DocumentCount*-1.

2.7.2.4 DocumentCount

This read-only property returns the number of workdocs within the folder.

Syntax: `DocumentCount As Long`

2.7.2.5 FolderData

This property provides the possibility to store and load a variable number of strings using any string as an index key.

Syntax: `FolderData(Index As String) As String`

Parameter	Description
`Index`	Any non-empty string that is used as an index key.

2.7.2.5.1 Sample Code

```
'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
```

2.7.2.6 InsertDocument

This method inserts a workdoc into a folder at some given position.

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

Parameter	Description

`Index`	Zero-based index at which *pWorkdoc* is to be inserted.
`pWorkdoc`	Workdoc object to be inserted.

2.7.2.7 MoveDocument

Use this method to move a workdoc from one position to another position in a folder.

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

Parameter	Description
`FromIndex`	Zero-based Index from where the workdoc is moved.
`ToIndex`	Zero-based index where the workdoc is to be placed.

2.7.2.8 RemoveDocument

This method removes a workdoc from a given index from a folder.

Syntax: `RemoveDocument(Index As Long)`

Parameter	Description
`Index`	Zero-based index in a folder from where the workdoc is to be removed.

2.8 SCBCdrTable

2.8.1 Description

The *Table* object represents a logical table in a document that is assigned to a field of a workdoc.

2.8.2 Methods and Properties

2.8.2.1 AddColumn

This method adds a new column to a table. It returns the zero-based index of the new column.

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

Parameter	Description
`ColumnName`	The name of the column to add.

2.8.2.2 AddRow

This method adds a new row to a table. It returns the zero-based index of the new row.

Syntax: `AddRow()`

2.8.2.3 AddUMColumn

This method adds a new unmapped column to a table. It returns the zero-based index of the new unmapped column.

Syntax: `AddUMColumn(pUMColumnIndex As Long)`

Parameter	Description
`pUMColumnIndex`	Returns the zero-based index of the new column.

2.8.2.4 AppendRows

This method appends new rows over the specified range within the document.

Syntax: `AppendRows(Top As Long, Height As Long, PageNumber As Long)`

Parameter	Description
`Top`	Top of region used for creation of new rows.
`Height`	Height of region used for creation of new rows.
`PageNumber`	DocPage number of region.

2.8.2.5 CellColor

This property sets or returns the color of the table cell.

Syntax: `CellColor(IsValid As Boolean) As OLE_COLOR`

Parameter	Description
`IsValid`	Boolean flag indicating if color refers to valid or invalid table cells.

2.8.2.6 CellLocation

This property sets or returns the location of the table cell.

Syntax:
```
CellLocation(Column As Variant, RowIndex As Long, Location As
CDRLocation)
As Long
```

Parameter	Description
`Column`	Zero-based index or name of column.
`RowIndex`	Zero-based index of row.
`Location`	Location parameter.

2.8.2.7 CellText

This property sets or returns the text of the table cell.

Syntax: `CellText(Column As Variant, RowIndex As Long) As String`

Parameter	Description
`Column`	Zero-based index or name of column.
`RowIndex`	Zero-based index of row.

2.8.2.7.1 Sample Code

```
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
```

2.8.2.8 CellValid

This property sets or returns the validity flag of the table cell.

Syntax: `CellValid (Column As Variant, RowIndex As Long) As Boolean`

Parameter	Description
`Column`	Zero-based index or name of column.
`RowIndex`	Zero-based index of row.

2.8.2.8.1 Sample Code

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

2.8.2.9 CellValidationErrorDescription

This property sets or returns the error description for the cell validation.

Syntax: `CellValidationErrorDescription(Column As Variant, RowIndex As Long) As String`

Parameter	Description
`Column`	Zero-based index or name of column.
`RowIndex`	Zero-based index of row.

2.8.2.9.1 Sample Code

```
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
```

2.8.2.10 CellVisible

This property sets or returns visible flag of the table cell. *(Currently not used.)*

Syntax: `CellVisible(Column As Variant, RowIndex As Long) As Boolean`

Parameter	Description
`Column`	Zero-based index or name of column.
`RowIndex`	Zero-based index of row.

2.8.2.11 CellWorktext

This property sets or returns the *Worktext* object of the cell.

Syntax: `CellWorktext(Column As Variant, RowIndex As Long) As ISCBCroWorktext`

Parameter	Description
`Column`	Zero-based index or name of column.
`RowIndex`	Zero-based index of row.

2.8.2.12 CellWorktextChanged

This property sets or returns a flag that indicates whether the cell *Worktext* has changed.

Syntax: `CellWorktextChanged(Column As Variant, RowIndex As Long) As Boolean`

Parameter	Description
`Column`	Zero-based index or name of column.
`RowIndex`	Zero-based index of row.

2.8.2.13 Clear

This method clears the content of the table. It removes all columns and all rows and resets all table attributes.

Syntax: `Clear()`

2.8.2.14 ClearColumn

This method clears the content of an existing column.

Syntax: `ClearColumn(Column As Variant)`

Parameter	Description
`Column`	Zero-based index or name of column.

2.8.2.15 ClearRow

This method clears the content of an existing row.

Syntax: `ClearRow(RowIndex As Long)`

Parameter	Description
`RowIndex`	Zero-based index of row.

2.8.2.16 ClearUMColumn

This method clears the content of an unmapped column.

Syntax: `ClearUMColumn(UMColumnIndex As Long)`

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped column to be cleared.

2.8.2.17 ColumnColor

This property sets or returns the color of the column.

Syntax: `ColumnColor(IsValid As Boolean) As OLE_COLOR`

Parameter	Description
`IsValid`	Flag indicating if color refers to valid or invalid columns.

2.8.2.18 ColumnCount

This read-only property returns the number of columns.

Syntax: `ColumnCount As Long`

2.8.2.19 ColumnExportEnable

This read-only property sets or returns the *ExportEnable* flag of a column.

Syntax: `ColumnExportEnable(Column As Variant) As Boolean`

Parameter	Description
`Column`	Zero-based index or name of column.

2.8.2.19.1 Sample Code

```
Dim i As Long
 Dim currentTable As ISCBCdrTable
 Set currentTable = pWorkdoc.Fields.ItemByName("Table").Table(0)
 For i = 0 To currentTable.ColumnCount-1
    If Left$(currentTable.ColumnName(i),3) = "tmp" Then
       currentTable.ColumnExportEnable(i) = False
    End If
Next i
```

2.8.2.20 ColumnIndex

This read-only property returns the column index for the name of a column.

Syntax: `ColumnIndex(ColumnName As String) As Long`

Parameter	Description
`ColumnName`	The name of the column.

2.8.2.21 ColumnLabelLocation

This property sets or returns the location of a column label (referring to first label line in case of multipage tables).

Syntax: `ColumnLabelLocation(Column As Variant, Location As CDRLocation) As Long`

Parameter	Description
Column	Zero-based index or name of column.
Location	Location parameter.

2.8.2.22 ColumnLabelText

This property sets or returns the column label.

Syntax: `ColumnLabelText(Column As Variant) As String`

Parameter	Description
Column	Zero-based index or name of column.

2.8.2.23 ColumnLocation

This property sets or returns the location of the column.

Syntax: `ColumnLocation(Column As Variant, PageNr As Long, Location As CDRLocation)As Long`

Parameter	Description
Column	Zero-based index or name of column.
PageNr	The DocPage number.
Location	Location parameter.

2.8.2.24 ColumnMapped

This property sets or returns a flag that indicates whether a column has been mapped.

Syntax: `ColumnMapped(Column As Variant) As Boolean`

Parameter	Description
Column	Zero-based index or name of column.

2.8.2.25 ColumnName

This read-only property returns the name of a column.

Syntax: `ColumnName(ColumnIndex As Long) As String`

Parameter	Description

`ColumnIndex`	Zero-based index of column.

2.8.2.26 ColumnValid

This property sets or returns a validity flag for a column. If the flag is set to **False**, the invalid state of the table field is not changed automatically.

Syntax: `ColumnValid(Column As Variant) As Boolean`

Parameter	Description
`Column`	Zero-based index or name of column.

2.8.2.27 ColumnVisible

This property sets or returns the visible flag of a column. This method affects the visibility of the column in Verifier.

Syntax: `ColumnVisible(Column As Variant) As Boolean`

Parameter	Description
`Column`	Zero-based index or name of column.

2.8.2.27.1 Sample Code

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

2.8.2.28 DeleteColumn

This method deletes a column specified by its name or by index.

Syntax: `DeleteColumn(Column As Variant)`

Parameter	Description
`Column`	Zero-based index or name of column.

2.8.2.29 DeleteRow

This method deletes a row specified by an index.

Syntax: `DeleteRow(RowIndex As Long)`

Parameter	Description
`RowIndex`	Zero-based index of row.

2.8.2.30 DeleteUMColumn

This method deletes an unmapped column specified by index.

Syntax: `DeleteUMColumn(UMColumnIndex As Long)`

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped column to be deleted.

2.8.2.31 FieldName

This property sets or returns the name of the field to which the table object belongs.

Syntax: `FieldName As String`

2.8.2.32 FillColumn

This method fills the column with words of specified area. If the table is empty, each text line is assigned to a table row. Otherwise, the existing row segmentation is used.

Syntax: `FillColumn(Left As Long, Top As Long, Width As Long, Height As Long, PageNumber As Long, Column As Variant)`

Parameter	Description
`Left`	Left position of area in pixels.
`Top`	Top of area in pixels.
`Width`	Width of area in pixels.
`Height`	Height of area in pixels.
`PageNumber`	DocPage number of area.
`Column`	Zero-based index or name of column.

2.8.2.33 FooterLocation

This property sets or returns the location of the table footer.

Syntax: `FooterLocation(Location As CDRLocation) As Long`

Parameter	Description
`Location`	Location parameter.

2.8.2.34 FooterPageNr

This property sets or returns the *DocPage* number of the table footer.

Syntax: `FooterPageNr As Long`

2.8.2.35 FooterText

This property sets or returns the text of the table footer.

Syntax: `FooterText As String`

2.8.2.36 HeaderLocation

This property sets or returns the location of the table header.

Syntax: `HeaderLocation(Location As CDRLocation) As Long`

Parameter	Description
`Location`	Location parameter.

2.8.2.37 HeaderPageNr

This property sets or returns the *DocPage* number of the table header.

Syntax: `HeaderPageNr As Long`

2.8.2.38 HeaderText

This property sets or returns the text of the table header.

Syntax: `HeaderText As String`

2.8.2.39 HightlightColumnIndex

This property sets or returns the index of the column to be highlighted.

Syntax: `HighlightColumnIndex As Long`

2.8.2.40 HighlightMode

This property sets or returns the *TableHighlightMode* of the table.

Syntax: `HighlightMode As CDRTableHighlightMode`

2.8.2.41 HighlightRowIndex

This property sets or returns the index of the row to be highlighted.

Syntax: `HighlightRowIndex As Long`

2.8.2.42 HighlightUMColumnIndex

This property sets or returns the zero-based index of an unmapped column to be highlighted.

Syntax: `HighlightUMColumnIndex As Long`

2.8.2.43 InsertColumn

This method Inserts a new column after the specified *ColumnIndex*.

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

Parameter	Description
`ColumnIndex`	Zero-based index of the existing column, after which the new column is inserted.
`ColumnName`	The name of the new column.

2.8.2.44 InsertRow

This method inserts a new row after the specified *RowIndex*.

Syntax: `InsertRow(RowIndex As Long)`

Parameter	Description
`RowIndex`	Zero-based index of the existing row, after which the new row is inserted.

2.8.2.45 InsertUMColumn

This method inserts a new, unmapped column.

Syntax: `InsertUMColumn(UMColumnIndex As Long)`

Parameter	Description
`UMColumnIndex`	Zero-based index of new column.

2.8.2.46 LabellinePageNr

This property sets or returns the *DocPage* number of the label line, which is the first occurrence for multipage tables.

Syntax: `LabellinePageNr As Long`

2.8.2.47 LocationExplicit

This property sets and returns the *LocationExplicit* flag.

Syntax: `LocationExplicit As Boolean`

2.8.2.48 MapColumn

This method maps an unmapped column. It transfers the content of an unmapped source column to a specified target column.

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

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped source column.
`Column`	Zero-based index or name of destination column.

2.8.2.49 MergeRows

This method merges two rows specified by two indices.

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

Parameter	Description
`RowIndex1`	Zero-based index of row 1.
`RowIndex2`	Zero-based index of row 2.

2.8.2.50 RemoveAllColumns

This method removes all mapped table columns.

Syntax: `RemoveAllColumns()`

2.8.2.51 RemoveAllRows

This method removes all table rows.

Syntax: `RemoveAllRows()`

2.8.2.52 RemoveAllUMColumns

This method removes all unmapped table columns.

Syntax: `RemoveAllUMColumns()`

2.8.2.53 RowColor

This property sets or returns the color of the row.

Syntax: `RowColor(IsValid As Boolean) As OLE_COLOR`

Parameter	Description
`IsValid`	Boolean flag indicating if color refers to valid or invalid rows.

2.8.2.54 RowCount

This read-only property returns the number of the rows.

Syntax: `RowCount As Long`

2.8.2.55 RowLocation

This property sets or returns the location of the row.

Syntax: `RowLocation(RowIndex As Long, Location As CDRLocation) As Long`

Parameter	Description
`RowIndex`	Zero-based index of row.
`Location`	Location parameter.

2.8.2.56 RowNumber

This property sets or returns the actual number of row.

Syntax: `RowNumber(RowIndex As Long) As Long`

Parameter	Description
`RowIndex`	Zero-based index of row.

2.8.2.56.1 Sample Code

```
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
```

2.8.2.57 RowPageNr

This property sets or returns the *DocPage* number of a row.

Syntax: `RowPageNr (RowIndex As Long) As Long`

Parameter	Description
`RowIndex`	Zero-based index of row.

2.8.2.58 RowValid

This property sets or returns a validity flag of a row.

Syntax: `RowValid (RowIndex As Long) As Boolean`

Parameter	Description
RowIndex	Zero-based index of row.

2.8.2.59 RowValidationErrorDescription

This property sets or returns an error description for a row validation.

Syntax: `RowValidationErrorDescription(RowIndex As Long) As String`

Parameter	Description
RowIndex	Zero-based index of row.

2.8.2.60 Significance

This property sets or returns the significance for the corresponding evaluation property of the table.

Syntax: `Significance(EvalPropIndex As Long) As Double`

Parameter	Description
EvalPropIndex	Index of evaluation property, as follows: 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).

2.8.2.61 SwapColumns

This method swaps two specified columns.

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

Parameter	Description
ColumnIndex1	Zero-based index of column 1.
ColumnIndex2	Zero-based index of column 2.

2.8.2.62 TableColor

This property sets or returns the color of the table.

Syntax: `TableColor(IsValid As Boolean) As OLE_COLOR`

Parameter	Description
IsValid	Boolean flag indicating if color refers to a valid or an invalid table.

2.8.2.63 TableFirstPage

This property sets or returns the *DocPage* number of the beginning of a table. It must be set after creation of a table, but cannot change afterwards.

Syntax: `TableFirstPage As Long`

2.8.2.64 TableLastPage

This property sets or returns the *DocPage* of the end of a table. It must be set after creation of a table and after assigning the *TableFirstPage*, but cannot change afterwards.

Syntax: `TableLastPage As Long`

2.8.2.65 TableLocation

This property sets or returns the location of a table.

Syntax: `TableLocation(PageNr As Long, Location As CDRLocation) As Long`

Parameter	Description
PageNr	DocPage number.
Location	Location parameter.

2.8.2.66 TableValid

This property sets or returns a validity flag of the table.

Syntax: `TableValid As Boolean`

2.8.2.67 TableValidationErrorDescription

This property sets or returns an error description for the table validation.

Syntax: `TableValidationErrorDescription As String`

2.8.2.67.1 Sample Code

```
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
```

2.8.2.68 Tag

This property sets and returns a tag associated with the table.

Syntax: `Tag As String`

2.8.2.69 TotalSignificance

This property sets and returns the total significance of the table.

Syntax: `TotalSignificance As Double`

2.8.2.70 UMCellColor

This property sets or returns a color of an unmapped table cell.

Syntax: `UMCellColor As OLE_COLOR`

2.8.2.71 UMCellLocation

This property sets or returns the location of an unmapped table cell.

Syntax: `UMCellLocation(UMColumnIndex As Long, RowIndex As Long, Location As CDRLocation) As Long`

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped column.
`RowIndex`	Zero-based index of unmapped row.
`Location`	Location parameter.

2.8.2.72 UMCellText

This property sets or returns the text of an unmapped table cell.

Syntax: `UMCellText (UMColumnIndex As Long, RowIndex As Long) As String`

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped column.

RowIndex	Zero-based index of unmapped row.

2.8.2.73 UMCellVisible

This property sets or returns a Visible flag of an unmapped table cell.

Syntax: `UMCellVisible(UMColumnIndex As Long, RowIndex As Long) As Boolean`

Parameter	Description
UMColumnIndex	Zero-based index of unmapped column.
RowIndex	Zero-based index of unmapped row.

2.8.2.74 UMCellWorktext

This property sets or returns the Worktext object of an unmapped cell.

Syntax: `UMCellWorktext(UMColumnIndex As Long, RowIndex As Long) As ISCBCroWorktext`

Parameter	Description
UMColumnIndex	Zero-based index of unmapped column.
RowIndex	Zero-based index of unmapped row.

2.8.2.75 UMColumnColor

This property sets or returns the color of an unmapped column.

Syntax: `UMColumnColor As OLE_COLOR`

2.8.2.76 UMColumnCount

This read-only property returns the number of unmapped columns.

Syntax: `UMColumnCount As Long`

2.8.2.77 UMColumnLabelLocation

This property sets or returns the location of an unmapped column label.

Syntax: `UMColumnLabelLocation(UMColumnIndex As Long, Location As CDRLocation) As Long`

Parameter	Description
UMColumnIndex	Zero-based index of unmapped column.

`Location`	Location parameter.

2.8.2.78 UMColumnLabelText

This property sets or returns the text of a label of an unmapped column.

Syntax: `UMColumnLabelText(UMColumnIndex As Long) As String`

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped column.

2.8.2.79 UMColumnLocation

This property sets or returns the location of an unmapped column.

Syntax: `UMColumnLocation(UMColumnIndex As Long, PageNr As Long, Location As CDRLocation) As Long`

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped column.
`PageNr`	DocPage number.
`Location`	Location parameter.

2.8.2.80 UMColumnVisible

This property sets or returns a `Visible` flag of an unmapped column. *(Currently not used.)*

Syntax: `UMColumnVisible(UMColumnIndex As Long) As Boolean`

Parameter	Description
`UMColumnIndex`	Zero-based index of unmapped column.

2.8.2.81 UnMapColumn

This method unmaps a column. It transfers content from a specified source column to a new, unmapped column.

Syntax: `UnMapColumn(Column As Variant) As Long`

Parameter	Description
`Column`	Zero-based index or name of source column.

2.8.2.82 WeightingFactor

This property sets or returns a weighting factor for a corresponding evaluation property.

Syntax: `WeightingFactor(EvalPropIndex As Long) As Double`

Parameter	Description
`EvalPropIndex`	Index of evaluation property, as follows: 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).

2.9 SCBCdrTextBlock

2.9.1 Description

This object represents a text block on a document. A text block may contain one or more lines.

2.9.2 Methods and Properties

2.9.2.1 Color

This property sets or returns the color that is used for text block highlighting.

Syntax: `Color As OLE_COLOR`

2.9.2.2 Height

This read-only property returns the height of the text block in pixels.

Syntax: `Height As Long`

2.9.2.3 Left

This property returns the left border of the text block in pixels.

Syntax: `Left As Long`

2.9.2.4 PageNr

This read-only property returns the number of the *DocPage* where the text block is located.

Syntax: `PageNr As Long`

2.9.2.5 Text

This read-only property returns the whole text of the text block.

Syntax: `Text As String`

2.9.2.6 Top

This read-only property returns the top border of the text block in pixels.

Syntax: `Top As Long`

2.9.2.7 Visible

This property controls whether the highlighted rectangle of the text block should be visible if the text block highlighting is enabled.

Syntax: `Visible As Boolean`

2.9.2.8 Weight

This read-only property returns the text block weight.

Syntax: `Weight As Double`

2.9.2.9 Width

This read-only property returns the width of the text block in pixels.

Syntax: `Width As Long`

2.9.2.10 WordCount

This read-only property returns the number of words that belong to the text block.

Syntax: `WordCount As Long`

2.9.2.11 WordID

Use this read-only property as an index for the word array of the workdoc.

Syntax: `WordID(Index As Long) As Long`

Parameter	Description
`Index`	Index of word inside the text block. Must be between **0** and *WordCount* -1

2.10 SCBCdrWord

2.10.1 Description

This object represents a textual word of a document.

2.10.2 Methods and Properties

2.10.2.1 Color

This property sets or returns the color that is used for highlighting checked words.

Syntax: `Color As OLE_COLOR`

2.10.2.2 Height

This read-only property returns the height of the word in pixels.

Syntax: `Height As Long`

2.10.2.3 Left

This read-only property returns the left border of the word in pixels.

Syntax: `Left As Long`

2.10.2.4 PageNr

This read-only property returns the number of the *DocPage* where the word is located.

Syntax: `PageNr As Long`

2.10.2.5 StartPos

This read-only property returns the index of the first character of the word inside the worktext that is attached to the workdoc.

Syntax: `StartPos As Long`

2.10.2.6 Text

This read-only property returns the text of the word.

Syntax: `Text As String`

2.10.2.7 TextLen

This read-only property returns the number of characters of the word.

Syntax: `TextLen As Long`

2.10.2.8 Tooltip

This property sets or returns a tooltip string that displays in the checked words highlight mode.

Syntax: `Tooltip As String`

2.10.2.9 Top

This read-only property returns the top border of the word in pixels.

Syntax: `Top As Long`

2.10.2.10 Visible

If the word highlighting for checked words is enabled, this property sets or returns if the highlighted rectangle of the word should be visible.

Syntax: `Visible As Boolean`

2.10.2.11 Width

This read-only property returns the width of the word in pixels.

Syntax: `Width As Long`

2.10.2.12 Worktext

This read-only property returns the `Worktext` object of the word.

Syntax: `Worktext As ISCBCroWorktext`

2.11 SCBCdrWorkdoc

2.11.1 Description

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

2.11.2 Methods and Properties

2.11.2.1 AddDocFile

This method adds a file into the workdoc. File types include CIDoc, image, and raw text.

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

Parameter	Description
`Path`	Path to the file to be added.
`FileType`	File type of the specified file, such as a CIDoc or Image.
`Assignment`	Specifies how DocPages are assigned to the workdoc.

2.11.2.2 AddField

This method adds a field to the workdoc.

Syntax: `AddField(Name As String)`

Parameter	Description

Name	Contains the name for the new field.

2.11.2.3 AddHighlightRectangle

This method adds a highlight rectangle on the page described by the following parameters. Set HighlightMode to CDRHighlightRectangles 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)`

Parameter	Description
Left	Left of highlight rectangle.
Top	Top of highlight rectangle.
Width	Width of highlight rectangle.
Height	Height of highlight rectangle.
PageNr	DocPage number of highlight rectangle.
Color	Color of highlight rectangle.

2.11.2.3.1 Sample Code

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

2.11.2.4 AnalyzeAlignedBlocks

This method splits the document into blocks that contain only left or right aligned lines. Using this method on a document with centered lines only usually results 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)`

Parameter	Description
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, that specifies how far off an existing block of words may be to belonging to that block. If it's 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 to **0** to ignore its effect.

2.11.2.5 AnalyzeBlocks

This method determines all the *TextBlocks* of text present in a workdoc that are a minimum `XDist` apart from each other on X-axis and a minimum of `YDist` apart from each other on Y-axis.

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

Parameter	Description
`XDist`	Minimum *X* distance between two *TextBlocks*.
`YDist`	Minimum *Y* distance between two *TextBlocks*.

2.11.2.5.1 Sample Code

```
pWorkdoc.AnalyzeBlocks(4,4)
```

2.11.2.6 AnalyzeEdges

This method analyzes a document set of words that are, within a certain tolerance, aligned either right or left. Use Highlight mode *CDRHighlightVerticalEdgesLeft* or C*DRHighlightVerticalEdgesRight* to make the results visible.

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

Parameter	Description
`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.
`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, default value -1] Specifies the page to be analyzed for edges. Set to -1 (default) if analysis is needed for all pages.

2.11.2.7 AnalyzeEdges2

This method is similar to the 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 the AnalyzeEdges method.

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

Parameter	Description
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.
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. 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*.

2.11.2.8 AnalyzeParagraphs

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

Syntax: `AnalyzeParagraphs()`

2.11.2.9 AppendWorkdoc

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

Syntax: `AppendWorkdoc(pWorkdoc As ISCBCdrWorkdoc)`

Parameter	Description

pWorkdoc	The workdoc that is to be appended.

2.11.2.10 AssignDocToPage

Use this method to assign a page of an image or CI Doc to a specific *DocPage* of the workdoc. This method requires that there are already documents inserted to the workdoc using the *AddDocFile* function, and that the *SetPageCount* function is called prior to using this method.

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

Parameter	Description
DocIndex	Zero-based CI Doc or image Index
DocPage	Zero-based *DocPage* inside the image or CI Doc
WorkdocPage	Zero-based *DocPage* inside the workdoc

2.11.2.11 AttractorColor

This property sets or returns the color that is used for attractor highlighting.

Syntax: `AttractorColor As OLE_COLOR`

2.11.2.12 BatchID

A read-only property of the workdoc that allows you to retrieve the ID of the batch in which the current workdoc resides.

Syntax: `strBatchID As String`

2.11.2.12.1 Sample Code

The following sample code shows how to return the Batch ID.

```
Dim strBatchID as String
 strBatchID = pWorkdoc.NamedProperty("BatchID")
```

2.11.2.13 BlockColor

This property sets or returns the color that is used for block highlighting.

Syntax: `BlockColor As OLE_COLOR`

2.11.2.14 BlockCount

This read-only property returns the number of text blocks 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`

2.11.2.15 CandidateColor

This property sets or returns the color that is used for candidate highlighting.

Syntax: `CandidateColor As OLE_COLOR`

2.11.2.15.1 Sample Code

```
pWorkdoc.CandidateColor = vbMagenta
```

2.11.2.16 Clear

Use this method to clear all the memories and to remove all the documents from workdoc. This leaves the workdoc in an initial state.

Syntax: `Clear()`

2.11.2.17 ClearHighlightRectangles

This method removes all highlighted rectangles.

Syntax: `ClearHighlightRectangles()`

2.11.2.18 ClsEngineConfidence

This property sets or returns a confidence level for a classification engine specified by its index in the collection of classification engines.

Syntax: `ClsEngineConfidence (lMethodIndex As Long) As Long`

Parameter	Description
`lMethodIndex`	Zero-based engine index in collection of classification engines.

2.11.2.18.1 Sample Code

The following sample code displays a message box with the confidence value for each classification engine.

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

2.11.2.19 ClsEngineDistance

This property sets or returns the distance value for a classification engine specified by its index in a collection of classification engines.

Syntax: `ClsEngineDistance(lMethodIndex As Long) As Long`

Parameter	Description
`lMethodIndex`	Zero-based engine index in collection of classification engines.

2.11.2.19.1 Sample Code

The following sample code displays a message box for each class, showing the classification engine distance.

```
Dim dblIndividualResult 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
```

2.11.2.20 ClsEngineResult

Use this property to access a classification result matrix. This matrix is used during the classification step to store the results of each used classification method for each document class 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 document class, therefore there is one row for each document class in the classification matrix. The matrix is 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*.

Syntax: `ClsEngineResult(MethodIndex As Long, DocClassIndex As Long) As CDRClassifyResult`

Parameter	Description
`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 WebCenter Forms Recognition 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` or `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 the script as `Project.AllClasses`, which has a type of `SCBCroCollection`. Use the `Count` property to get the number of document classes or use the `ItemIndex` or `ItemName` property to find the index of document class or the name for an index.

2.11.2.20.1 Sample Code

The following sample code sets the classification result of the Brainware Classify Engine to `YES` for a document in doclass `VOID`. If Brainware Classify is the only engine or all other classes would be `CDRClassifyNo`, the document gets classified as `VOID`.

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

2.11.2.21 ClsEngineWeight

This property provides access to the classification weights within the classification result matrix.

Syntax: `ClsEngineWeight(MethodIndex As Long, DocClassIndex As Long) As Double`

Parameter	Description
`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 WebCenter Forms Recognition 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* or *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 the script as `Project.AllClasses`, which has a type of `SCBCroCollection`. Use the *Count* property to get the number of document classes or use the `ItemIndex` or `ItemName` property to find the index of document class or the name for an index.

2.11.2.22 CreationDate

A read-only property of the workdoc that allows 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.

2.11.2.22.1 Sample Code

The following sample code shows how to get the creation date.

```
Dim dtCreationDate as Date
 dtCreationDate = pWorkdoc.NamedProperty("CreationDate")
```

2.11.2.23 CreationDateAsFileTimeUTC

A read-only property of the workdoc that allows the developer to retrieve the creation date of the current workdoc in UTC. When a document is placed in a new exception batch, the attribute updates to a new date/time stamp.

2.11.2.23.1 Sample Code

The following sample code shows how to return the creation date.

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

2.11.2.24 CreateFromWorktext

This method creates a workdoc from the OCR'd text of an image.

Syntax: `CreateFromWorktext(pWorktext As ISCBCroWorktext)`

Parameter	Description
`pWorktext`	Object pointer of the worktext object.

2.11.2.25 CutPage

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

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

Parameter	Description
`PageIndex`	Zero-based index of *DocPage* after which the workdoc has to be cut.
`ppNewWorkdoc`	New workdoc object generated as part of the current workdoc.

2.11.2.26 CurrentBatchState

This read-only property 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 batch component.

Syntax: `pWorkdoc.CurrentBatchState`

2.11.2.27 DeleteFile

This method deletes all **.wdc** files and corresponding images of the workdoc.

Syntax: `DeleteFile(DeleteDocFiles As Boolean)`

Parameter	Description
`DeleteDocFiles`	Boolean flag to inform whether to delete files or not.

2.11.2.28 DisplayPage

This property sets or returns the displayed *DocPage* specified by the zero-based index of the workdoc in the viewer.

Syntax: `DisplayPage As Long`

2.11.2.28.1 Sample Code

If a customer requires Verifier to display a specific page of each document instead of the first one when opening the document, use the DisplayPage property in the script. Index 0 represents first page.

The following sample code displays page 3 if the document has 4 pages or more.

```
Private Sub ScriptModule_VerifierFormLoad(pWorkdoc as
SCBCdrPROJLib.SCBCdrWorkdoc, FormClassName as String, FormName as String)
    If pWorkdoc.PageCount >=4 Then
       pWorkdoc.DisplayPage = 2
    End if
End Sub
```

2.11.2.29 DocClassName

This property sets or returns the name of the document class to which the document was classified.

Syntax: `DocClassName As String`

2.11.2.29.1 Sample Code

```
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
```

2.11.2.30 DocFileCount

This read-only property returns the number of documents from which the workdoc is built.

Syntax: `DocFileCount As Long`

2.11.2.31 *DocFileDatabaseID*

This read-only property returns the database ID of document files attached to a workdoc. It corresponds to the `[File].[Id]` value in the database. The document file index must be passed as a parameter when using the `DocFileDatabaseID` property.

Use this property in custom script as a unique identifier of document files that were processed by WebCenter Forms Recognition.

Syntax: `DocFileDatabaseID(ByVal Index As long)`

Parameter	Description
`Index`	The `Index` parameter has a valid range from **0** to *DocFileCount*-1.

2.11.2.31.1 Sample Code

The following sample code returns the unique ID of the last document file attached to a workdoc.

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

2.11.2.32 DocFileName

This read-only property returns the full path name of a document (image or text file) from which the workdoc is built.

Syntax: `DocFileName(Index As Long) As String`

Parameter	Description
`Index`	The *Index* parameter has a valid range from **0** to *DocFileCount*-1.

2.11.2.32.1 Sample Code

If a workdoc was created from a single document, such as a multi-TIFF file, you can get the name of the document file by accessing the 0 index.

```
Path = pWorkdoc.DocFileName(0)
 The script function below returns the TIF file creation date.
 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
```

2.11.2.33 DocFileType

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

Syntax: `DocFileType(Index As Long) As CDRDocFileType`

Parameter	Description
`Index`	The *Index* parameter has a valid range from **0** to *DocFileCount*-1.

2.11.2.34 DocState

This property sets or returns the current state of the document.

Syntax: `DocState As CDRDocState`

2.11.2.35 EdgeCount

This read-only property returns the number of vertical edges found in a document.

Syntax: `EdgeCount(edgeSide As CDREdgeSide) As Long`

Parameter	Description
`edgeSide`	Flag to distinguish between left and right edges.

2.11.2.36 ErrorDescription

This property sets or returns an error description.

Syntax: `ErrorDescription As String`

2.11.2.36.1 Sample Code

```
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
```

2.11.2.37 ExportDocumentToXML

This method exports the structure and the field data of the current workdoc to an XML file or MSXML object in a predefined format.

Use the named properties `XML_ExportCandidates`, `XML_ExportWords`, and `XML_ExportWordChars` to configure the export to optionally capture the field candidates, OCR word data and the associated character data.

By default, the method exports all fields and table field columns. Use the `XmlExportEnabled` and `ColumnExportEnable` properties to exclude specific fields or table field columns from the XML export.

Whenever possible, the XML element and attribute names correspond to the `SCBCdrWorkdoc` property names.

An `ErrorDescription` attribute is only added to an XML element if the corresponding Valid attribute is set to false.

Syntax: `ExportDocumentToXml(ByVal vTarget As Variant)`

Parameter	Description
`vTarget`	Possible values ▪ A string which specifies the filename, including path. Any existing file will be overwritten. ▪ An MSXML 3.0 or MSXML 6.0 object. It is the equivalent of saving the XML file and reparsing it using this object.

2.11.2.37.1 Sample Code

The following sample code saves the OCR data, candidates, fields and workdoc structure to an XML file.

```
pWorkdoc.NamedProperty("XML_ExportWords") = True
 pWorkdoc.NamedProperty("XML_ExportWordChars") = True
 pWorkdoc.NamedProperty("XML_ExportCandidates") = True
 pWorkdoc.ExportDocumentToXml("C:\ExistingFolder\" & pWorkdoc.Filename & ".xml")
```

2.11.2.37.2 Sample Code

The following sample code saves the XML data to an MSXML2.DOMDocument60 object instead of a file.

```
' Note: Add reference to Microsoft XML, version 6.0 in the script page
 Dim xmlDoc60 As MSXML2.DOMDocument60
```

```
Set xmlDoc60 = New MSXML2.DOMDocument60
pWorkdoc.ExportDocumentToXml(xmlDoc60)
' Change xmlDoc60 here
xmlDoc60.documentElement.appendChild(xmlDoc60.createElement("NewNode"))
' ...
xmlDoc60.Save("xmlDoc60.xml")
Set xmlDoc60 = Nothing
```

2.11.2.37.3 XML Element Definitions

Section	Description
DocClass	Contains document class information, such as class name, parent class and classification results.
DocFiles	Contains the document file structure, such as name and type (CI or Image document.)
DocPages	Contains the document page information, such as size and applied rotation.
Words	Contains information about the single words in the document, such as word text, page number, and position of the word in pixels.
Characters	For a word, contains information about the single characters that compose it, such as character code and position in pixels. Note: For CI documents, the reported position and confidence values are those for the word.
Fields	Contains the workdoc field information, such as name, extracted text, text position and validity.
Candidates	For a field, contains all candidate information, such as text, weight and position.

```
' Note: Add reference to Microsoft XML, version 6.0 in the script page
Dim xmlDoc60 As MSXML2.DOMDocument60
Set xmlDoc60 = New MSXML2.DOMDocument60
pWorkdoc.ExportDocumentToXml(xmlDoc60)
' Change xmlDoc60 here
xmlDoc60.documentElement.appendChild(xmlDoc60.createElement("NewNode"))
' ...
xmlDoc60.Save("xmlDoc60.xml")
Set xmlDoc60 = Nothing
```

2.11.2.37.4 Sample XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Workdoc XML_version="2.0" FileName="01English_US01_STP.wdc">
    <DocClass DocClassName="Invoices">
        <ParentDocClass DocClassName="Invoices"/>
        <ClsDocClass ID="1" ClsDocClassName="Invoices" Res="4" Confidence="0"/>
        <ClsDocClass ID="2" ClsDocClassName="Generic" Res="1" Confidence="1"/>
        ...
    </DocClass>
    <DocFiles DocFileCount="1">
```

```
        <DocFile ID="0" DocFileName="C:\...\01English_US01_STP.tif"
DocFileType="CDRDocFileTypeCroImage"/>
    </DocFiles>
    <DocPages DocPageCount="1">
        <DocPage PageNr="0" DocIndex="0" DocPageIndex="0" Width="2579"
Height="3309" XRes="300" YRes="300" Rotation="359.64413017"/>
    </DocPages>
    <Lines LineCount="54"/>
    <Words WordCount="359">
    ...
        <Word ID="3" Page="0" Line="2" Left="1496" Top="149" Width="67"
Height="22">
            <Text>PAGE</Text>
            <Characters CharCount="4">
                <Char ID="0" Code="P" Confidence="100" Left="1496" Top="150"
Width="14" Height="19"/>
                <Char ID="1" Code="A" Confidence="100" Left="1511" Top="150"
Width="16" Height="19"/>
                ...
            </Characters>
        </Word>
    ...
    </Words>
    <Fields FieldCount="88">
        ...
        <Field ID="2" Name="InvoiceNumber" Valid="false" Page="0" Left="1649"
Top="219" Width="139" Height="31" ErrorDescription="Invalid invoice number">
            <Text>7A6F2</Text>
            <Candidates CandidateCount="61">
                <Candidate ID="0" Weight="1.3563312626" Page="0" Left="1649"
Top="219" Width="139" Height="31">
                    <Text>7A6F2</Text>
                </Candidate>
                <Candidate ID="1" Weight="0.53850805759" Page="0" Left="2337"
Top="219" Width="139" Height="30">
                    <Text>23013</Text>
                </Candidate>
                ...
            </Candidates>
        </Field>
        ...
        <Field ID="15" Name="LineItems" Valid="true" Page="-1" Left="0" Top="0"
Width="0" Height="0">
            <Table Valid="true">
                <Columns ColumnCount="14">
                    ...
                    <Column ID="4" Name="Description"/>
                    <Column ID="5" Name="Quantity"/>
                    ...
                </Columns>
                <Rows RowCount="5">
                    <Row ID="0" Page="0" Valid="true">
                        ...
                        <Cell Column="4" Left="482" Top="1420" Right="1146"
Bottom="1552" Valid="true">
                            <Text>CDROM EDITION</Text>
                        </Cell>
                        <Cell Column="5" Left="1221" Top="1420" Right="1813"
Bottom="1552" Valid="true">
                            <Text>1</Text>
                        </Cell>
                        ...
                    </Row>
```

```
                    ...
                </Rows>
            </Table>
        </Field>
    </Fields>
</Workdoc>
```

2.11.2.38 ExportToXML

This method exports OCR data results of the current workspace into an XML file with a predefined format. The export captures word data and the associated characteristics data.

Syntax: `ExportToXml (ByVal DocumentLanguage As String, ByVal DocumentType As String, ByVal Customer As String, ByVal eExportType As CDRExportType, ByVal XMLFilePath As String)`

Parameter	Description
`DocumentLanguage, DocumentType, Customer`	Customize these parameters to use it later for filtering purposes. The values have no correlation with the Workdoc.
`DocumentType`	Customize this parameters to use it for filtering purposes. The values have no correlation with the workdoc. Note If the string is empty, the value defaults to Default. Null is not allowed.
`Customer`	Customize this parameters to use it for filtering purposes. The values have no correlation with the workdoc. Note If the string is empty, the value defaults to Default. Null is not allowed.
`eExportType`	Defines the type of information from the current document for the export to the XML files.
`XMLFilePath`	Defines the path for the XML file. Leave it as an empty string to save the XML files in the current application start folder. It is recommended to define the file path in script. You can specify an existing folder terminated by a back slash, or define the target name for the XML file explicitly.

2.11.2.38.1 Sample Code

The following sample code exports the OCR data for the current pWorkdoc into an XML file located in the C:\Temp directory.

```
pWorkdoc.ExportToXml("", "", "", CDRExportTypeOCRData, "C:\Temp\")
```

XML File Format

Section	Description
`Document`	Contains the general document information, such as page count, line count, and word count.
`Words`	Contains information about the single words in the document, such as word text, word length, page number, and position of the word in pixels.

`Characters`	Contains information about the single characters of the word, such as character text and character position in pixels. For CI documents, this method exports only the word positions, but no individual character positions.

2.11.2.38.2 Sample XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 <Document>
    <Name>00000473</Name>
    <DocumentType>Default</DocumentType>
    <DocumentLanguage>Default</DocumentLanguage>
    <Customer>Default</Customer>
    <PageCount>1</PageCount>
    <Pages>
       <Page id="0" DocFileType="Image"/>
    </Pages>
    <LineCount>39</LineCount>
    <WordCount>334</WordCount>
    <Words>
       <Word id="0">
          <Text>UNICOM</Text>
          <Length>6</Length>
          <StartPos>0</StartPos>
          <Page>0</Page>
          <Line>1</Line>
          <Top>131</Top>
          <Left>303</Left>
          <Height>102</Height>
          <Width>564</Width>
          <Characters>
             <Char id="0">
                <Code>U</Code>
                <Top>133</Top>
                <Left>303</Left>
                <Height>100</Height>
                <Width>80</Width>
             </Char>
             <Char id="1">...</Char>
          </Characters>
       </Word>
       <Word id="1">...</Word>
    </Words>
 </Document>
```

2.11.2.39 FieldColor

This property sets or returns the color that is used to highlight valid and invalid fields.

Syntax: `FieldColor(FieldValid As Boolean) As OLE_COLOR`

Parameter	Description
`FieldValid`	If set to **True** it specifies the color for valid fields, or it specifies the color for invalid fields if **False**.

2.11.2.40 Fields

This read-only property provides access to all fields of a document.

Syntax: `Fields As ISCBCdrFields`

2.11.2.40.1 Sample Code

The following sample code reads the text content of a simple field.

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

2.11.2.41 FileName

This read-only property contains the database ID of the workdoc and returns the database workdoc ID and name.

Note: To retrieve the file name of the image from which the workdoc was created, use the *DocFileName* property found above.

Syntax: `Filename As String`

2.11.2.42 Folder

This read-only property returns the folder to which the workdoc belongs.

Syntax: `Folder As ISCBCdrFolder`

2.11.2.43 FolderIndex

This read-only property provides the index of the folder to which a workdoc belongs.

Syntax: `FolderIndex As Long`

2.11.2.44 ForceClassificationReview

In the application, the *PostClassify* event can force a manual classification review even if the classification succeeded.

2.11.2.44.1 Sample Code

The following sample code shows how to force the manual classification process from the script event *PostClassify*.

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

2.11.2.45 GetEdge

This method returns the coordinates for the 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)`

Parameter	Description

`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.

2.11.2.46 GetFileSizeKB

This method retrieves the file size of an image or document.

Syntax: `GetFileSizeKB(pWorkdoc As SCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	The current workdoc object.

2.11.2.46.1 Sample Code

```
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
```

2.11.2.47 GetWorktextForPageArea

This function returns a worktext object from a specific location on a document. The worktext object contains text and positional information relating to the area specified. You can view this as a temporary zone to read a piece of information through a script and review the returned result for that area.

The area to search starts from *Left* and *Top* coordinates and finishes at *Width* and *Height* coordinates, provided in pixels. These are the same coordinates that you would enter for a reading zone. For more information, refer to *Setting up Zone Analysis* in the *Designer User Guide*).

The project developer may test their page area coordinates using a zone.

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

Parameter	Description
`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 of the area in pixels.
`Height`	Height of the area in pixels.
`includePartial`	Boolean flag. If set to **False** this restricts reading of worktext to specified area, otherwise, if set to **True** this completes words that appear partially in the specified area with outside information.

2.11.2.47.1 Sample Code

The following sample code 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
```

2.11.2.48 HighlightCandidate

This property sets or returns the position of the highlighted candidate.

Syntax: `HighlightCandidate As Long`

2.11.2.49 HighlightField

This property sets or returns the position of the highlighted field.

Syntax: `HighlightField As Long`

2.11.2.50 HighlightMode

This property sets or returns the current mode of highlighting.

Syntax: `HighlightMode As` *`CDRHighlightMode`*

2.11.2.51 IgnoreAnalysisFailures

This is an optional capability to ignore any errors during WebCenter Forms Recognition's extraction analysis phase. Otherwise, the extraction analysis stops in the middle of field extraction and does not apply processing for other fields and does not fire further events.

This capability is optional and is disabled by default to ensure the backwards compatibility is not affected in any way.

If this property is set to **True**, any errors occurring during the extraction analysis phase are ignored. Errors will not cause a sudden termination of the extraction process. Instead, traces are left in the component logs for the CdrProj library at tracing level 1 (i.e. errors).

This functionality can be activated at any time, for example in the *PreExtract* event.

Syntax: `pWorkdoc.NamedProperty(PropertyName As String) As Variant`

Parameter	Description
`PropertyName`	Set this parameter value to **IgnoreAnalysisFailures** to enable this functionality.

2.11.2.51.1 Sample Code

```
' 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
```

2.11.2.52 Image

This read-only property returns an Image object for the specified *DocPage* of the workdoc.

Syntax: `Image(Index As Long) As ISCBCroImage`

Parameter	Description
`Index`	Index of the *DocPage* that is valid from **0** to *PageCount* - 1.

2.11.2.53 IsPlainText

This property sets or returns a Boolean value specifying whether the worktext is plain text or not.

Syntax: `IsPlainText As Boolean`

2.11.2.54 Language

This property sets or 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`

2.11.2.55 LineColor

This property sets or returns the color that is used for line highlighting.

Syntax: `LineColor As OLE_COLOR`

2.11.2.56 Load

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

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

Parameter	Description
`Filename`	Name of the file.
`ImageRootPath`	Relative path of the file.

2.11.2.57 PageCount

This read-only property returns the number of displayable *DocPages* of the workdoc.

Syntax: `PageCount As Long`

2.11.2.58 Pages

This read-only property returns a single *DocPage* of the workdoc.

Syntax: `Pages(PageIndex As Long) As ISCBCdrDocPage`

Parameter	Description
`PageIndex`	Index of the *DocPage* to access, which is valid from **0** to *PageCount*-1.

2.11.2.59 Paragraph

This read-only property provides access to the paragraph array of the Workdoc.

Syntax: `Paragraph(Index As Long) As ISCBCdrTextBlock`

Parameter	Description
`Index`	Specifies the index of the paragraph. Valid indexes are from **0** to *ParagraphCount* – 1.

2.11.2.60 ParagraphCount

This read-only property returns the number of paragraphs in the workdoc.

Syntax: `ParagraphCount As Long`

2.11.2.61 PCAppType

Use this named property to optimize the check amount extraction rates.

Syntax: `pWorkdoc.NamedProperty ("PCAppType")`

Parameter	Description
`"POD" – default setting`	The Check Analysis Engine is tuned to minimize the error rate. You can limit candidate lists to values with the highest confidence levels.

`Proof of deposit`	The engine often fails to recognize values above 1 million US dollars. See the HVOL option below.
`"RMT Remittance`	The Check Analysis Engine is tuned to a maximum read rate without rejection. It expects to use all results and alternative answers. It is not necessary to accept only answers with high confidence values, because users can perform cross-validation using remittance coupons and databases.
`"HVOL"`	This parameter enables the engine to recognize amounts larger than 1 million US dollars.

2.11.2.61.1 Sample Code

```
Private Sub Document_PreExtract(pWorkdoc as SCBCdrPROJLib.ISCBCdrWorkdoc)
    pWorkdoc.NamedProperty("PCAppType") = "HVOL" 'High Value Amount Range
End Sub
```

2.11.2.62 PCCheckType

Use this named property to configure the check type recognition.

Syntax: `pWorkdoc.NamedProperty ("PCCheckType")`

Parameter	Description

`"ALL"`
`- Personal checks`
`- Business checks`
`- Cash tickets`
`- Deposit slips`
`- Money orders`
	This parameter sets the Check Analysis Engine to expect all supported types of documents: personal checks, business checks, cash tickets, deposit slips, and money orders.

`"P"` `- Personal checks`	This parameter sets the Check Analysis Engine to expect the input stream to consist of only personal checks. If your documents consist of 99.5% personal checks, this setting may improve processing speed while not significantly affecting accuracy.

`"PB"`
`– default setting`
`- Personal checks`
`- Business checks`
	This parameter sets the Check Analysis Engine to expect checks and cash tickets. Use this setting if your documents consist mainly of checks.

`"PBD"`
`- Personal checks`
`- Business checks`
`- Cash tickets`
`- Deposit slips`
	This parameter sets the Check Analysis Engine to expect checks, cash tickets, and deposit slips.

2.11.2.62.1 Sample Code

```
Private Sub Document_PreExtract(pWorkdoc as SCBCdrPROJLib.ISCBCdrWorkdoc)
    pWorkdoc.NamedProperty("PCCheckType") = "P" 'Personal Checks only
End Sub
```

2.11.2.63 PCDateHint

Use this named property to configure the reference date for the check date recognition by the Check Analysis Engine. By default, the application uses the system date.

Syntax: `pWorkdoc.NamedProperty ("PCDateHint")`

Parameter	Description
`Format`	Use the following format for the reference date. YYYY.MM.DD Example 2015.06.18 To set the reference date to the system date, add an empty string.

2.11.2.63.1 Sample Code

```
Private Sub Document_PreExtract(pWorkdoc as SCBCdrPROJLib.ISCBCdrWorkdoc)
    pWorkdoc.NamedProperty("PCDateHint") = "2013.04.12" 'April 12th 2012
End Sub
```

The following example sets the reference date to the system date:

```
Private Sub Document_PreExtract(pWorkdoc as SCBCdrPROJLib.ISCBCdrWorkdoc)
    pWorkdoc.NamedProperty("PCDateHint") = " " 'System date
End Sub
```

2.11.2.64 PCReReadAlways

Use this named property to enable re-analyzing the Check Analysis Engine fields. This is helpful if you need to perform Designer or scripting testing, such as for document rotation. It is also helpful if you change one of the Check Analyses Engine settings. By default, re-analysis is switched off.

Syntax: `pWorkdoc.NamedProperty ("PCReReadAlways")`

Parameter	Description
`True`	The Check Analysis Engine repeats the analysis, if • Executed in the Designer's Definition Mode for extraction • The `AnalyzeField` method is used in script. If AnalyzeField is used on a Check Analysis Engine field, all fields with Check Analysis Engine assigned will be re-analyzed. The `<Field> PostAnalysis` event only triggers for the field for which the Analyzed field was triggered. • `AnalyzeDocument` is used in script.

`False – default setting`	The Check Analysis Engine does not execute a secondary analysis on a document when PIC did not unload the document from the memory.

2.11.2.64.1 Sample Code

```
Private Sub Document_PreExtract(pWorkdoc as SCBCdrPROJLib.ISCBCdrWorkdoc)
    pWorkdoc.NamedProperty("PCReReadAlways") = True   'Switch on re-Analysis
End Sub
```

2.11.2.65 PDFExport

This method generates a PDF file from the workdoc based on *CDRPDFExportStyle*.

Syntax: `PDFExport(FileName As String)`

Parameter	Description
`FileName`	Name of the exported PDF file.

2.11.2.66 PDFGetInfoType

This method returns the export type of a given page in a PDF file.

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

Parameter	Description
`PageIdx`	Page number of the PDF file.
`pExportStyle`	Type of export.

2.11.2.67 PDFSetInfoType

This method sets the export type of a PDF file.

Syntax: `PDFSetInfoType(PageIdx As Long, pExportStyle As CDRPDFExportStyle)`

Parameter	Description
`PageIdx`	Zero-based *DocPage* number.
`pExportStyle`	Type of export.

2.11.2.68 ReadZone

This is 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])
```

Parameter	Description
`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`	Optional. Specifies the right border of the OCR region in percent. Use **100** here to read until the right border.
`Left`	Optional. Specifies a left offset for the OCR region in percent. Use **0** here to read from the left border.
`Top`	Optional. Specifies the top offset for the OCR region in percent. Use **0** here to read from the top border.
`Bottom`	Optional. Specifies the bottom line of the OCR region in percent. Use **100** here to read until the bottom border.

2.11.2.69 Refresh

This method refreshes the workdoc's *DocPage* that is currently shown in the Viewer.

Syntax: `Refresh()`

2.11.2.70 RenameDocFile

Use this method to change the name of the CI Doc or image at a given *DocIndex* by the given new name.

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

Parameter	Description
`DocIndex`	Specifies the zero-based CI Doc or image index.
`NewName`	New name given to the document at *DocIndex*.

2.11.2.71 ReplaceFirstImage

This method replaces the first image in a workdoc.

Syntax: `ReplaceFirstImage(Path As String)`

Parameter	Description
`Path`	Image path to replace the existing workdoc's image with.

2.11.2.72 Save

This method saves a workdoc with given file name and its *DocFiles* relatively at the given *ImageRootPath*.

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

Parameter	Description
`Filename`	Filename of workdoc.
`ImageRootPath`	Relative path where all corresponding *DocFiles* are saved. Leave this parameter empty if files are saved in the same directory as the workdoc.

2.11.2.73 SetDocPageIndex

This method allows the script implementation of the page merging workflow step.

2.11.2.73.1 Sample Code

```
For j = 0 To thePreviousWorkdoc.PageCount -1 Step 1
    theNextWorkdoc.InsertPage (thePreviousWorkdoc, j, True,
theNextWorkdoc.PageCount)
    theNextWorkdoc.Pages (theNextWorkdoc.PageCount -1).SetDocPageIndex(0, j + 1)
 Next j
```

2.11.2.74 ShowTooltips

This property sets or returns if tooltips display when moving the mouse pointer over a displayed workdoc.

Syntax: `ShowTooltips As Boolean`

2.11.2.75 SkipDocumentReprocessingAfterMerging

By default, when workdocs are combined in the AppendWorkdoc event, the new combined document is classified and extracted after merging. Set this named property in the AppendWorkdoc event to `True` to skip reprocessing. In this case, the new combined document has the classification and extraction results of `pLastWorkdoc` after merging.

2.11.2.75.1 Sample Code

The following sample code shows how to skip reprocessing.

```
Private Sub ScriptModule_AppendWorkdoc(pLastWorkdoc As
SCBCdrPROJLib.ISCBCdrWorkdoc, pCurrentWorkdoc As SCBCdrPROJLib.ISCBCdrWorkdoc,
pAppendType As SCBCdrPROJLib.CdrMPType)
    ' Merge pLastWorkdoc and pCurrentWorkdoc if they are of same class "CLASS_A"
    If pLastWorkdoc.DocClassName = "CLASS_A" And pCurrentWorkdoc.DocClassName =
"CLASS_A"
       pAppendType = CdrSubseqPage
       ' The new combined document usually does not need to be
classified/extracted
       pLastWorkdoc.NamedProperty("SkipDocumentReprocessingAfterMerging") = True
    End If
End Sub
```

2.11.2.76 SkipTableCellMassValidation

This method allows you to optionally activate special "skip table cell mass validation" mode for validation of table cells. By default, WebCenter Forms Recognition uses "mass validation of

invalid cells". This means that when a Verifier user hits the [Enter] key within an invalid cell, all other invalid cells are automatically re-validated by the system. This behavior may lead to performance problems in WebCenter Forms Recognition projects with a large number of invalid cells that must each be corrected manually. It may be also unacceptable if validation routines are unavailable for some of the processed transactions and manual review by the Verifier user is required for all cells.

You can invoke this feature at any time and is in effect for the next fired cell validation event. You can also re-enable mass validation at any time. One of the possible events in which this script sample can be integrated is *VerifierFormLoad*.

2.11.2.76.1 Sample Code

The following sample code shows how to switch the validation mode individually for each processed document.

```
pWorkdoc.NamedProperty("SkipTableCellMassValidation") = True
```

2.11.2.77 SkipTrainingWithEngine

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

Syntax: `SkipTrainingWithEngine(bstrEngineName As String) As Boolean`

Parameter	Description
`bstrEngineName`	The name of the classification engine.

2.11.2.78 Table

This read-only property returns a table for a given index of the workdoc.

Syntax: `Table(Index As Long) As ISCBCdrTable`

Parameter	Description
`Index`	Specifies the index of the table. Valid indices are from 0 to *TableCount*-1.

2.11.2.79 TableCount

This read-only property returns the number of table objects stored within the workdoc.

Syntax: `TableCount As Long`

2.11.2.80 TextBlock

This read-only property returns a text block by an index of the workdoc.

Syntax: `TextBlock(Index As Long) As ISCBCdrTextBlock`

Parameter	Description

`Index`	Zero-based index of the text block. Valid indices are from **0** to *BlockCount*-1.

2.11.2.81 Textline

This read-only property returns a text line by an index of the workdoc.

Syntax: `Textline(Index As Long) As ISCBCdrTextBlock`

Parameter	Description
`Index`	Zero-based index of the line. Valid indices are from **0** to *TextlineCount*-1.

2.11.2.82 TextlineCount

This read-only property retrieves the number of text lines present in a workdoc.

Syntax: `TextlineCount As Long`

2.11.2.83 TrainedWithEngine

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

Syntax: `TrainedWithEngine(bstrEngineName As String) As Boolean`

Parameter	Description
`bstrEngineName`	Name of the engine.

2.11.2.84 UnloadDocs

This method releases all the images and CI Docs that belong to this workdoc.

Syntax: `UnloadDocs()`

2.11.2.85 Word

This read-only property provides access to the word array of the workdoc.

Syntax: `Word(Index As Long) As ISCBCdrWord`

Parameter	Description
`Index`	Index of the requested word. Valid indices are from **0** to *WordCount*-1.

2.11.2.86 WordColor

This property sets or returns the color that is used for word highlighting.

Syntax: `WordColor As OLE_COLOR`

2.11.2.87 WordCount

This read-only property returns the number of words of the workdoc.

Syntax: `WordCount As Long`

2.11.2.87.1 Sample Code

```
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
```

2.11.2.88 WordSegmentationChars

This property sets or returns a string that contains the characters used for the segmentation of words.

Syntax: `WordSegmentationChars As String`

2.11.2.89 Worktext

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

Syntax: `Worktext As ISCBCroWorktext`

2.11.2.90 XML_ExportCandidates

This named property controls whether the list of candidates for each field in the workdoc is exported to the XML file. The text, weight and position of each candidate is exported.

Set the property to `True` in the *ExportDocument* event to include the candidate list for each field in the exported file.

The default value is `False`.

Syntax: `pWorkdoc.NamedProperty ("XML_ExportCandidates") As Boolean`

2.11.2.91 XML_ExportWordChars

This named property controls whether the characters composing OCR words in the workdoc are exported to the XML file. The character code, confidence and position of each character are exported.

Note: For CI documents, position and confidence values for individual characters are not available. The Top, Left, Height, Width and Confidence values for each letter are those for the word in which the character is found.

Set the property to `True` in the *ExportDocument* event to include the character information in the exported file. This property has no impact if the named property XML_ExportWords is set to `False`.

The default value is `False`.

Syntax: `pWorkdoc.NamedProperty ("XML_ ExportWordChars") As Boolean`

3 Worktext Object Reference (SCBCroWorktextLib)

3.1 SCBCdrWorktextLib

3.1.1 Description

OCR recognition results and CIDocs can be saved in a `Worktext` object. The `Worktext` object contains the raw text including geometric information.

The `SCBCroWorktextLib` provide `Worktext` object.

3.1.2 Type Definitions

3.1.2.1 CroWorktextDirection

Specifies the worktext direction.

Available Types	Description
`CroWorktextDirection_Horizontal`	Worktext direction is horizontal.
`CroWorktextDirection_Vertical`	Worktext direction is vertical.

3.1.2.2 DimensionInfo_t

Specifies the location in millimeters (mm) or pixels.

Available Types	Description
`MM_Height`	Height in mm
`MM_Left`	Left in mm
`MM_Top`	Top in mm
`MM_Width`	Width in mm
`PX_Height`	PX_Height
`PX_Left`	Left in pixels
`PX_Top`	Top in pixels
`PX_Width`	Width in pixels

3.1.2.3 PosInfo_t

Specifies the position of the character.

Available Types	Description
POS_Col	Column index
POS_Img	Image index
POS_Line	Line index
POS_Zone	Zone index

3.1.2.4 WktConversionType

Specifies the worktext conversion type.

Available Types	Description
WorktextConvLowerCase	Worktext is converted to lower case.
WorktextConvUpperCase	Worktext is converted to upper case.

3.2 SCBCroWorktext

3.2.1 Description

The SCBCroWorktext library provides methods and properties to modify the worktext object.

3.2.2 Methods and Properties

3.2.2.1 AppendAlternative

This method appends one or more alternative characters to the active character. The active character is the last one added to the worktext object.

Note: A string longer than one character added to a worktext object using the AppendString method or the Text property makes the last character of the string to the active character.

Syntax:
```
AppendAlternative (Value As String, Confidence As Long, Top As Long,
Left As Long, Height As Long, Width As Long)
```

Parameter	Description
Value	Alternative characters.
Confidence	Confidence of the alternative characters. Optional parameter. The default value is 100.
Top	Top position of the alternative characters. Optional parameter. The default value is 0.
Left	Left position of the alternative characters. Optional parameter. The default value is 0.

Height	Height of the alternative characters. Optional parameter. The default value is 0.
Width	Width of the alternative characters. Optional parameter. The default value is 0.

3.2.2.2 AppendImage

This method adds a new page reference to the worktext.

The page reference contains various zones. To add a new zone to a page reference, use AppendZone.

Zones contain different lines. To add a new line to a zone, use AppendLine.

Lines contain text. To add text to a line, use AppendString or AppendReject.

The added zones, lines and strings always refer to the page added by AppendImage.

Syntax: `AppendImage (ImgNr As Long, XRes As Long, YRes As Long)`

Parameter	Description
ImgNr	Number of the current image.
XRes	Resolution in horizontal direction in dots per inch.
YRes	Resolution in vertical direction in dots per inch.

3.2.2.3 AppendLine

This method adds a new line.

Text added to the worktext object using AppendString or AppendReject always refers to the line generated by AppendLine. To add text to a new line, use the AppendLine method first.

Syntax: `AppendLine (Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
Top	Top position of the line in pixels.Optional parameter. The default value is 0.
Left	Left position of the line in pixels.Optional parameter. The default value is 0.
Height	Height of the line in pixels.Optional parameter. The default value is 0.
Width	Width of the line in pixels.Optional parameter. The default value is 0.

3.2.2.4 AppendReject

This method appends a rejected character to the current worktext.

A reject is a symbol that a recognition engine could not recognize. The worktext cannot contain information about recognized characters only, you can also add unrecognized characters to the worktext.

You need to specify the page, zone and line indices before by using `AppendImage`, `AppendZone` and `AppendLine`.

Syntax: `AppendReject (Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
`Top`	Top position of the reject in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the reject in pixels. Optional parameter. The default value is 0.
`Height`	Height of the reject in pixels. Optional parameter. The default value is 0.
`Width`	Width of the reject in pixels. Optional parameter. The default value is 0.

3.2.2.5 AppendString

This method appends one or more characters to the worktext object and makes the last character of the string to the active character. You need to specify the page, zone and line indices before using `AppendImage, AppendZone` and `AppendLine`. Methods and properties such as `AppendAlternative` refer to the active character.

Syntax: `AppendString (Value As String, Confidence As Long, Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
`Value`	The string to add to the worktext, can consist of one or more characters.
`Confidence`	Current confidence of the value. Optional parameter. The default value is 100. **Possible values** 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.t
`Top`	Top position of the zone in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the zone in pixels. Optional parameter. The default value is 0.
`Height`	Height of the zone in pixels. Optional parameter. The default value is 0.
`Width`	Width of the zone in pixels Optional parameter. The default value is 0.

3.2.2.6 AppendTag

This method appends a tag to the taglist.

A tag contains information associated to a single character.

Syntax: `AppendTag (CharIndex As Long, TagType As Long, TagValue As Variant)`

Parameter	Description
`CharIndex`	Zero-based index of the character being described by the tag.
`TagType`	Describes the meaning of the TagValue value.
`TagValue`	Value that describes the character.

3.2.2.7 AppendTo

This method appends some or all characters including the character information to another worktext object.

Syntax: `AppendTo (pDestination As ISCBCroWorktext, StartPos As Long, CharCount As Long)`

Parameter	Description
`pDestination`	Interface pointer of the destination Worktext object.
`StartPos`	Index of the first character of the source worktext object to append to the destination worktext object.
`CharCount`	Number of characters to copy from the source worktext object to the destination worktext object.

3.2.2.8 AppendZone

This method adds a zone to the current page.

Note: Text added to a worktext object using `AppendString` or `AppendReject` always refers to the line added by `AppendZone`.

Syntax: `AppendZone (Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
`Top`	Top position of the zone in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the zone in pixels. Optional parameter. The default value is 0.
`Height`	Height of the zone in pixels. Optional parameter. The default value is 0.

`Width`	Width of the zone in pixels. Optional parameter. The default value is 0.

3.2.2.9 ApplyForwardLanguageConversion

This method applies forward language conversion to support non-western languages.

Syntax: `ApplyForwardLanguageConversion(Language As Long, UseSingleChar As Boolean)`

Parameter	Description
`Language`	Language value
`UseSingleChar`	True / False
`Top`	Top position of the zone in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the zone in pixels. Optional parameter. The default value is 0.
`Height`	Height of the zone in pixels. Optional parameter. The default value is 0.
`Width`	Width of the zone in pixels Optional parameter. The default value is 0.

3.2.2.10 BulkUpdate

This property sets or returns if the worktext object executes the change events.

If the BulkUpdate property is set to true, the worktext object does not trigger the OnChange events.

Use this property to improve performance when adding or changing multiple worktext characters.

Syntax: `BulkUpdate As Boolean`

3.2.2.11 Clear

This method removes all information contained in the worktext object.

Syntax: `Clear ()`

3.2.2.12 ConvertCharacters

This method converts the worktext to upper or lower case.

Syntax: `ConvertCharacters (ConvType As WktConversionType)`

Parameter	Description

ConvType	Specifies the worktext conversion type.

3.2.2.13 Copy

This method copies characters from a source worktext object to a destination worktext object.

Syntax: `Copy (pDestination As ISCBCroWorktext, StartPos As Long, CharCount As Long)`

Parameter	Description
pDestination	Interface pointer of the destination worktext object.
StartPos	Index of the first character of the source worktext object to copy.
CharCount	Number of characters to copy to the destination worktext object.

3.2.2.14 CorrectPageReferences

This method corrects the page references.

Syntax: `CorrectPageReferences(Start As Long, Offset As Long)`

Parameter	Description
Start	Start position
Offset	Offset

3.2.2.15 CountBase

This property sets or returns the character index offset.

To start all indices at 0, set CountBase to 0. To start all indices at n, set CountBase to n.

Syntax: `CountBase as Long`

3.2.2.16 FindString

This method searches a string or a substring in the current worktext. Returns -1 if no string is found.

Syntax: `FindString (StartPos As Long, Search As String) As Long`

Parameter	Description
StartPos	Position to start searching.
Search	Text to search.

3.2.2.17 GetAlternativeChar

This method returns the alternative item for the item specified by the CharIndex parameter.

Syntax: `GetAlternativeChar (CharIndex As Long, AltNr As Long) As String`

Parameter	Description
`CharIndex`	Index of the character for which the alternative is required.
`AltNr`	Index of the alternative.

3.2.2.18 GetAlternativeCharConfidence

This method returns the confidence level of an alternative item.

Syntax: `GetAlternativeCharConfidence (CharIndex As Long, AltNr As Long) As Long`

Parameter	Description
`CharIndex`	Index of the character to which the alternative character is related to.
`AltNr`	Index of the alternative character.

3.2.2.19 GetAlternativeCount

This method returns the number of alternative characters for a character specified by the CharIndex parameter.

Syntax: `GetAlternativeCount (CharIndex As Long) As Long`

Parameter	Description
`CharIndex`	Index of the character for which the number of alternatives returns.

3.2.2.20 GetBoostReference

This method returns the attributes of the boost character alternative.

Syntax: `GetBoostReference (CharIndex As Long, pCharCode As Long, pConfidence As Long, pAttributes As Long)`

Parameter	Description
`CharIndex`	Index of the character.
`pCharCode`	Character code.

pConfidence	Current confidence of the value. Possible values: 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.
pAttributes	Attributes

3.2.2.21 GetCharAttributes

This method returns the character attributes.

Syntax: `GetCharAttributes (CharIndex As Long, pChar As Long, pAttributes As Long, pConfidence As Long)`

Parameter	Description
CharIndex	Index of the character.
pChar	Character
pAttributes	Attributes
pConfidence	Current confidence of the value. Possible values: 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.

3.2.2.22 GetCharConfidence

This method returns the confidence level of a character.

Possible return values are between 0 and 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.

Syntax: `GetCharConfidence (CharIndex As Long) As Long)`

Parameter	Description
CharIndex	Index of the item.

3.2.2.23 GetCharIndex

This method returns the index of a character.

Syntax: `GetCharIndex (Line As Long, Col As Long) As Long`

Parameter	Description
Line	Line of the character.

Col	Column of the character.

3.2.2.24 GetCharInfo

This method returns the page, zone, line, or column index of a character.

Syntax: `GetCharInfo (posInfo As PosInfo_t, CharIndex As Long) As Long`

Parameter	Description
posInfo	Parameter to choose the type of information returned.
CharIndex	Index of the character.
Top	Top position of the zone in pixels. Optional parameter. The default value is 0.
Left	Left position of the zone in pixels. Optional parameter. The default value is 0.
Height	Height of the zone in pixels. Optional parameter. The default value is 0.
Width	Width of the zone in pixels Optional parameter. The default value is 0.

3.2.2.25 GetTag

This method returns a tag of the taglist.

A tag contains information associated to a single character.

Syntax: `GetTag (TagIndex As Long, CharIndex As Long, TagType As Long, TagValue As Variant)`

Parameter	Description
TagIndex	Index specifying the tags index.
CharIndex	Index of the character being described by the tag.
TagType	Describes or interprets the meaning of the TagValue value.
TagValue	The tag itself.

3.2.2.26 GetTagCount

This read-only property returns the number of available tags.

Syntax: `GetTagCount as Long`

3.2.2.27 ImageCount

This read-only property returns the number of images for which characters are available.

Syntax: `ImageCount As Long`

3.2.2.28 ImageRotation

This property sets or returns the clockwise rotation angle of the image. The rotation angle is the angle around which the image was rotated before recognition. The center point is the left top corner of the image.

Syntax: `ImageRotation (ImgIdx As Long, newVal As Double)`

Parameter	Description
`ImgIdx`	Index of the image.
`newVal`	Rotation angle. **Note:** The parameter value adds to the current rotation, it does not replace it.

3.2.2.29 ImageTranslation

This property sets or returns the image translation.

Syntax: `ImageTranslation (ImgIdx As Long, Direction As CroWorktextDirection) As Long`

Parameter	Description
`ImgIdx`	Index of the image.
`Direction`	Specifies the coordination type.

3.2.2.30 ImageXRes

This read-only property returns the image resolution in horizontal direction.

Syntax: `ImageXRes (ImgIdx As Long) As Long`

Parameter	Description
`ImgIdx`	Index of the image.

3.2.2.31 ImageYRes

This read-only property returns the image resolution in vertical direction.

Syntax: `ImageYRes (ImgIdx As Long) As Long`

Parameter	Description
ImgIdx	Index of the image.

3.2.2.32 GetTagCount

This read-only property returns the number of available tags.

Syntax: `GetTagCount As Long`

3.2.2.33 InsertAfter

This method inserts a text in the worktext object after the specified index.

Syntax: `InsertAfter (CharIndex As Long, value As String, [Confidence As Long = 100], [Top As Long = FALSE], [Left As Long = FALSE], [Height As Long = eFALSE], [Width As Long = FALSE])`

Parameter	Description
CharIndex	Character index to insert.
Value	String or character value to add.
Confidence	Confidence of the added value. **Possible values :** 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident. Optional parameter. The default value is 100.
Top	Top position of the text in pixels. Optional parameter. The default value is 0.
Left	Left position of the text in pixels. Optional parameter. The default value is 0.
Height	Height of the text in pixels. Optional parameter. The default value is 0.
Width	Width of the text in pixels Optional parameter. The default value is 0.

3.2.2.34 InsertBefore

This method inserts a text in the worktext object before the specified index.

Syntax: `InsertBefore (CharIndex As Long, value As String, Confidence As Long, Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
CharIndex	Character index before which the text is inserted.
Value	Added string or character value.

`Confidence`	Confidence of the added value. **Possible values:** 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident. Optional parameter. The default value is 100.
`Top`	Top position of the characters in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the characters in pixels. Optional parameter. The default value is 0.
`Height`	Height of the characters in pixels. Optional parameter. The default value is 0.
`Width`	Width of the characters in pixels. Optional parameter. The default value is 0.

3.2.2.35 InsertRejectAfter

This method inserts a reject character in the worktext after the specified position.

Syntax: `InsertRejectAfter (CharPos As Long, Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
`CharPos`	Reject character to insert.
`Top`	Top position of the reject character in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the reject character in pixels. Optional parameter. The default value is 0.
`Height`	Height of the reject character in pixels. Optional parameter. The default value is 0.
`Width`	Width of the reject character in pixels. Optional parameter. The default value is 0.

3.2.2.36 InsertRejectBefore

This method inserts a reject character in the worktext before the specified character position.

Syntax: `InsertRejectBefore (CharPos As Long, Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
`CharPos`	Position to insert before.
`Top`	Top position of the reject character in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the reject character in pixels. Optional parameter. The default value is 0.

Height	Height of the reject character in pixels. Optional parameter. The default value is 0.
Width	Width of the reject character in pixels. Optional parameter. The default value is 0.

3.2.2.37 IsCharValid

This method returns TRUE if the worktext entry specified by CharIndex is a valid character, or *FALSE* if the entry is a reject.

Syntax: `IsCharValid (CharIndex As Long) As Boolean`

Parameter	Description
CharIndex	Index of requested character.

3.2.2.38 JoinNextLine

This method removes the CRLF of the specified line.

After calling JoinNextLine, the line that had the index (LineIndex + 1) before the call receives the index LineIndex.

A successful call of JoinNextLine reduces the number of lines by 1.

Syntax: `JoinNextLine (LineIndex As Long)`

Parameter	Description
LineIndex	Index of the line.

3.2.2.39 LanguageTranslationMethod

This property sets or returns the current language transcode method identifier.

Syntax: `LanguageTranslationMethod As Long`

3.2.2.40 LineCount

This read-only property returns the number of lines of the worktext.

Syntax: `LineCount As Long`

3.2.2.41 LineDim

This method returns the dimension of the specified line.

Syntax: `LineDim (dInfo As DimensionInfo_t, LineIndex As Long) As Long`

Parameter	Description

dInfo	Parameter to determine the type of information returned.
LineIndex	Index of the line.

3.2.2.42 LineLength

This read-only property returns the length of a line in pixels.

Syntax: `LineLength (LineIndex As Long) As Long`

3.2.2.43 LineStart

This read-only property returns the character index of the first character in the line.

Syntax: `LineStart (LineIndex As Long) As Long`

Parameter	Description
LineIndex	Index of the line.

3.2.2.44 LineText

This method returns the text of a specified line.

Syntax: `LineText (LineIndex As Long) As String`

Parameter	Description
LineIndex	Index of the line.

3.2.2.45 Load

This method loads the worktext from a file.

Syntax: `Load (Filename As String)`

Parameter	Description
Filename	Path and filename of the file that contains the worktext.

3.2.2.46 PackRejects

This method combines a sequence of rejects to a single reject.

Syntax: `PackRejects (PackLimit As Long, Confidence As Long, StartPos As Long, Length As Long)`

Parameter	Description

PackLimit	Minimal length of the reject sequence to combine to a single reject.
Confidence	Confidence limit for a character. Characters with a lower confidence are treated as rejects. **Possible values:** 0 to 100
StartPos	Index of the first character.
Length	Number of characters to process.

3.2.2.47 Read

This method returns characters of a worktext object.

Syntax: `Read (StartPos As Long, Length As Long) As String`

Parameter	Description
StartPos	Position of the first character to return.
Length	Number of characters to return.Optional parameter. The default value is 1.

3.2.2.48 RejectChar

This property sets or returns the character used to symbolize rejects.

Note: If a recognition engine cannot identify a symbol, it returns a reject. The worktext stores the rejects. The default reject character is "?".

Syntax: `RejectChar As String`

3.2.2.49 RejectCount

This read-only property returns the number of rejects.

Syntax: `RejectCount As Long`

3.2.2.50 RemDelimiters

This method removes characters from the worktext object.

Syntax: `RemDelimiters (Delimiters As String)`

Parameter	Description
Delimiters	Characters to remove from the worktext.To combine all lines to a single line, use the symbol for CRLF.

3.2.2.51 Remove

This method removes the specified number of characters from the worktext.

Syntax: `Remove (CharIndex As Long, Length As Long)`

Parameter	Description
`CharIndex`	The first character to remove.
`Length`	Number of characters to remove.Optional parameter. The default value is 1.

3.2.2.52 RemoveLine

This method removes a line from the worktext.

Syntax: `RemoveLine (LineIndex As Long)`

Parameter	Description
`LineIndex`	Index of the line.

3.2.2.53 ReplaceLineLocation

This method replaces the location of the specified line.

Syntax: `ReplaceLineLocation(PositionType As DimensionInfo_t, _LineIndex As Long, _NewPosition As Long)`

Parameter	Description
`PositionType`	Type of the position.
`LineIndex`	Index of the line.
`NewPosition`	New position of the line.

3.2.2.54 ReplaceLineText

This method replaces the text of the specified line.

Syntax: `ReplaceLineText(LineIndex As Long, NewLineText As String)`

Parameter	Description
`LineIndex`	Index of the line.
`NewLineText`	New text of the line.

3.2.2.55 Save

This method saves the worktext object to a file.

Note: The method overwrites existing files.

Syntax: `Save (Filename as String)`

Parameter	Description
`Filename`	Path and name of the file to create or overwrite.

3.2.2.56 SetBoostReference

This method assigns the attributes of the boosted character alternative.

Syntax: `SetBoostReference(CharIndex As Long, CharCode As Long, Confidence As Long, Attributes As Long)`

Parameter	Description
`CharIndex`	Index of requested character.
`CharCode`	Character code.
`Confidence`	Confidence of the character. Optional parameter. The default value is 100. **Possible values:** 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.
`Attributes`	Attributes to assign.

3.2.2.57 SetCharAttributes

This method assigns attibutes to a character.

Syntax: `SetCharAttributes(CharIndex As Long, Char As Long, Attributes As Long, Confidence As Long)`

Parameter	Description
`CharIndex`	Index of requested character.
`Char`	The character
`Attributes`	Attributes to assign.
`Confidence`	Confidence of the character. Optional parameter. The default value is 100. **Possible values:** 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.

3.2.2.58 SetConfidence

This method sets the confidence of a character.

Syntax: `SetConfidence (CharIndex As Long, NewValue As Long)`

Parameter	Description
`CharIndex`	Index of the character which confidence is set.
`NewValue`	Confidence of the character. **Possible values** 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.

3.2.2.59 Substitute

This method substitutes a part of the characters in the worktext object.

Note: The method substitutes only the characters, but not the information associated to the characters, such as position or tag.

Syntax: `Substitute (CharIndex As Long, Length As Long, value As String)`

Parameter	Description
`CharIndex`	Position of the first character to substitute.
`Length`	Number of characters to substitute.
`Value`	String that replaces the worktext characters.

3.2.2.60 Tag

This property sets or returns the content of the tag.

Syntax: `Tag As String`

3.2.2.61 Text

This property sets or returns the characters stored in the worktext object.

Note: Setting new characters initializes the values describing the characters, such as position, with default values.

Syntax: `Text As String`

3.2.2.62 TextLength

This read-only property returns the number of characters stored in the worktext object.

Use this property instead of the WrinWrap method `Len`.

Syntax: `TextLength As String`

3.2.2.63 TransformCoordinates

This method transforms the coordinates of the worktext objects like zones, lines, and characters.

Note: Undoing the coordinate transformation can be performed in a second call of TransformCoordinates using –Dx, -Dy, -RotAngle as parameters if Dx, Dy, RotAngle were the first call of TransformCoordinates parameters.

Syntax: `TransformCoordinates (ImgNr As Long, Dx As Long, Dy As Long, RotAngle As Double)`

Parameter	Description
`ImgNr`	Index of the image.
`Dx`	Specifies the translation in horizontal direction.
`Dy`	Specifies the translation in vertical direction.
`RotAngle`	Specifies the rotation angle. The center point is the top left corner of the document.
`ImgNr`	Index of the image.

3.2.2.64 ZoneDim

This method returns the dimension of the specified zone.

Syntax: `ZoneDim (dInfo As DimensionInfo_t, ZoneIndex As Long) As Long)`

Parameter	Description
`dInfo`	Parameter to choose specific information.
`ZoneIndex`	Index of the zone.

3.2.2.65 AppendString

This method appends one or more characters to the worktext object and makes the last character of the string to the active character. You need to specify the page, zone and line indices before using `AppendImage, AppendZone` and `AppendLine`. Methods and properties such as `AppendAlternative` refer to the active character.

Syntax: `AppendString (Value As String, Confidence As Long, Top As Long, Left As Long, Height As Long, Width As Long)`

Parameter	Description
`Value`	The string to add to the worktext, can consist of one or more characters.
`Confidence`	Current confidence of the value. Optional parameter. The default value is 100. **Possible values** 0 to 100, where 0 means that the recognition is unconfident, and 100 means that the recognition is confident.t
`Top`	Top position of the zone in pixels. Optional parameter. The default value is 0.
`Left`	Left position of the zone in pixels. Optional parameter. The default value is 0.
`Height`	Height of the zone in pixels. Optional parameter. The default value is 0.
`Width`	Width of the zone in pixels Optional parameter. The default value is 0.

3.2.2.66 Value

This property sets or returns a string that contains the complete information contained in the worktext object.

Syntax: `Value As String`

3.2.2.67 ZoneCount

This read-only property returns the number of zones for which characters are available.

Syntax: `ZoneCount As Long`

4 Project Object Reference (SCBCdrPROJLib)

4.1 Description

The *Project* object represents a complete project definition, including all document classes, field definitions, and used classification and extraction methods.

4.2 Type Definitions

4.2.1.1 CDRApplicationName

This type defines the application type.

Available Types	Description
`TANDesigner`	Designer
`TANLearnSetManager`	Learn Set Manager
`TANLocalVerifier`	Verifier used as local project for SLW
`TANRuntimeServer`	Runtime Service Instance
`TANVerifier`	Verifier
`TANWebVerifier`	Web Verifier
`TANUnknown`	Unknown application

4.2.1.2 CDRBatchReleaseAction

This types defines the automatic action when releasing a batch in Verifier.

Available Types	Description
`CdrBatchReleaseActionCancel`	Return to current batch and last document verified.
`CdrBatchReleaseActionReturnToList`	Return to batch list.
`CDRBatchReleaseActionUndefined`	Undefined
`CdrBatchReleaseActionUserDefined`	Default value Let the user decide what to do next selecting the required option in a dialog.
`CdrBatchReleaseActionVerifyNextInvalidBatch`	Open next batch to verify.
`CdrBatchReleaseActionVerifyNextInvalidState`	Verify this batch with next invalid state if batch is still invalid, otherwise get next invalid batch.

CDRBatchReleaseActionUndefined	Undefined
CdrBatchReleaseActionUserDefined	Default value Let the user decide what to do next selecting the required option in a dialog.

4.2.1.3 CDRClassifyMode

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

Available Types	Description
CDRClassifyAverage	Average is computed.
CDRClassifyMax	Maximum is computed.
CDRClassifyWeightedDistance	For each cell of classification matrix difference between maximum of column and classification weight is calculated.

4.2.1.4 CDRDatabaseWorkflowTypes

The workflow type of the batch. These are standard WebCenter Forms Recognition workflow settings for batches.

Available Types	Description
CDRAutoTrainingFailed	Automatic document training failed.
CDRAutoTrainingSucceeded	Automatic document training succeeded.
CDRClassificationFailed	Automatic document classification failed.
CDRClassificationSucceeded	Automatic document classification succeeded.
CDRCleanupFailed	Automatic document cleanup failed.
CDRCleanupSucceeded	Automatic document cleanup succeeded.
CDRDocumentSeparationFailed	Automatic document separation failed.
CDRDocumentSeparationSucceeded	Automatic document separation succeeded.
CDREmailImportFailed	Automatic document import from exchange server failed.
CDREmailImportSucceeded	Automatic document import from exchange server succeeded.
CDRExportFailed	Automatic document export failed.

CDRExportSucceeded	Automatic document export succeeded.
CDRExtractionFailed	Automatic document extraction failed.
CDRExtractionSucceeded	Automatic document extraction succeeded.
CDRFileSystemExportFailed	Runtime Server based database import from file system batches failed.
CDRFileSystemExportSucceeded	Runtime Server based database import from file system batches succeeded.
CDRImportFailed	Automatic document import failed.
CDRImportSucceeded	Automatic document import succeeded.
CDRManualClassificationIncomplete	Manual document classification was not completed.
CDRManualClassificationSucceeded	Manual document classification succeeded.
CDRManualDocumentSeparationIncomplete	Manual document separation failed.
CDRManualDocumentSeparationSucceeded	Manual document separation succeeded.
CDRManualFinalValidationFullyIncomplete	Manual final document validation was not completed.
CDRManualFinalValidationSucceeded	Manual final document validation succeeded.
CDRManualTrainingFailed	Manual document training failed.
CDRManualTrainingSucceeded	Manual document training succeeded.
CDRModifiedByDesignerApplication	The document was saved via Designer application without changing its workflow status.
CDRModifiedByVerifierApplication	The document was saved via Verifier application without changing its workflow status.
CDROCRFailed	Automatic document OCR failed.
CDROCRSucceeded	Automatic document OCR succeeded.
CDRPartialManualValidationIncomplete	Partial manual document validation was not completed.
CDRPartialManualValidationSucceeded	Partial manual document validation succeeded.

CDRReserved	Reserved for system use.
CDRReset	Initial state of document
CDRScanningFailed	Images scanning failed.
CDRScanningSucceeded	Images scanning succeeded.

4.2.1.5 CdrDocumentBinarizationMode

This type defines the binarization mode.

Available Types	Description
CdrDocumentBinarizationSkipped	Deactivates forced binarization of CIDoc images. Activating this setting may improve OCR results for grayscale and colored images. This setting is compatible with the following preprocessing methods: ▪ Binarisation ▪ Despeckle [IG] ▪ Invert This setting is compatible with the following recognition engines: ▪ FineReader 10 ▪ FineReader 11 ▪ Cairo OMR ▪ QualitySoft Barcode This setting is not compatible with the following preprocessing methods: ▪ Box & Comb Removal ▪ Clean Border [IG] ▪ Lines Manager This setting is not compatible with the following recognition engines: ▪ Kadmos 5 ▪ Cleqs Barcode ▪ Transcripts You can use non-compatible methods and/or engines with this setting active provided that preprocessing includes Binarisation. In case of incompatible preprocessing methods, Binarisation must appear prior to the method.

	Note: This binarization is a simple threshold based method and does not produce results identical to the dynamic binarization which is performed when CdrDocumentBinarizationSkipped is not active.
	
	Sample Code
	
	The following sample code shows how to activate the setting:
	
	`Private Sub ScriptModule_Initialize(ByVal ModuleName As String)`
	`    Settings.DocumentBinarizationMode = CdrDocumentBinarizationSkipped`
	`End Sub`

`CdrDocumentBinarizationAdvanced`	Default setting. Executes advanced document binarization.

4.2.1.6 CDRFieldType

This type defines the type of a field.

Available Types	Description
`CDRFieldTypeTable`	Field type is table.
`CDRFieldTypeText`	Field type is text, which may be single or multiline text.

4.2.1.7 CdrFocusChangeReason

This enumeration defines the reason for the focus change of a Verifier field edit.

Available Types	Description
`CdrBeforeFormClosed`	Focus changed by closing the form.
`CdrBeforeFormLoaded`	Focus is not set yet as the form has just been loaded.
`CdrEnterPressed`	Focus changed by pressing [Enter].
`CdrFcrCandidateCopied`	Focus changed because a candidate and its location were copied to the field.
`CdrFcrRefreshed`	Focus changed because the selection area and its location were copied to the field.
`CdrFcrSelectionCopied`	Focus changed because the selection area and its location were copied to the field.
`CdrFcrWordCopied`	Focus changed because a word and its location were 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.

4.2.1.8 CdrForceValidationMode

This table defines the options for *Force Validation*. Force Validation is when a Verifier user presses the [Enter] key on an invalid field three times to force a known invalid value to be considered valid.

Available Types	Description
CdrForceValDefault	ForceValidationMode inherited.
CdrForceValForbidden	Force Validation not allowed.
CdrForceValPermitted	Force Validation allowed.

4.2.1.9 CdrLocalTrainingReason

This type defines the possible reasons for local training.

Available Types	Description
CdrAddedByUser	A Verifier user manually initiated the training process.
CdrAddedSinceDocumentWasPoorlyExtracted	The document is considered as poorly extracted and therefore needs to be trained.
CdrRejectedByUser	A Verifier user manually rejected the training process.
CdrRejectedSinceDocumentWasWellExtracted	The document is well extracted before the manual verification process, and therefore, is not needed to be trained.

4.2.1.10 CdrMessageSeverity

This type defines the different message severities.

Available Types	Description
CDRSeverityEmailNotification	Stores the message in the log file and forward it to the MMC console / System Monitoring view and send as an email 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.
`CDRSeverityLogFileOnly`	Stores the message to the application log file only.
`CDRSeveritySystemMonitoring`	Stores 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.

4.2.1.11 CdrMessageType

This type defines the different message types:

Available Types	Description
`CDRTypeInfo`	An informational message.
`CDRTypeWarning`	A warning message.
`CDRTypeError`	An error message.

4.2.1.12 CdrMPType

This type defines the possible results of the multi-page classification.

Available Types	Description
`CDRTypeInfo`	An informational message.
`CdrAttachmentPage`	The classified page is an attachment.
`CdrFirstPage`	The classified page is the first page of a new document and does not belong to the previous page.
`CdrLastPage`	The classified page is the last page of the current document. The next page starts a new document.
`CdrPageUndefined`	The page could not be classified.
`CdrSinglePage`	The classified page belongs to a single page document.
`CdrSubseqPage`	The classified page belongs to the previous page. More pages can be appended to the current document.
`CdrUserCorrectedPage`	The page was corrected manually by the Verifier user.
`CdrUserRejectedPage`	The page was rejected manually by the Verifier user.

4.2.1.13 CDRsiModule

This type defines the module in which the smart index definition is used.

Available Types	Description
`CDRsiModuleDistiller`	Use smart indexing in automatic field extraction.
`CDRsiModuleDistVer`	Use smart indexing in automatic field extraction and manual field validation.
`CDRsiModuleVerifier`	Use smart indexing in manual field validation.

4.2.1.14 CdrSLWDifferentResultsAction

When the *Template* and *Associative Search* classify engines determine different results during classification, there are different options how the program should continue the processing.

Available Types	Description
`CdrDoNothing`	Lets the Verifier user decide to skip special processing altogether.
`CdrDoSmartDecision`	Makes a smart decision, for example, the machine makes the decision for the classification. The system determines which one is the right DocClass based on an algorithm that compares the results of the Associative Search and the Template classification. You can select this feature from the **Supervised Learning** tab in the Designer application.
`CdrUseDocumentClassName`	Automatically assigns current document class name to the supplier field content.
`CdrUseSupplierField`	Automatically assigns supplier field content to the document class name.

4.2.1.15 CdrTableFocusChangeReason

This type defines the possible causes for a cell focus change in a verification table.

Available Types	Description
`CdrTfcrBrainwareExtractionApplied`	The focus changed due to applied interactive Brainware table extraction.
`CdrTfcrCellBitmapClicked`	The focus changes because a cell bitmap is clicked.
`CdrTfcrCellDoubleClicked`	The focus changes because a cell is double clicked.
`CdrTfcrCellLocationClicked`	The focus changes because the location of this cell is clicked in the active Cairo Viewer.
`CdrTfcrColumnMapped`	The focus changes because a column is mapped.
`CdrTfcrColumnsSwapped`	The focus changes because columns are swapped.

CdrTfcrColumnUnmapped	The focus changes because a column is unmapped.
CdrTfcrEnterPressed	The focus changes because the validation of a cell is invoked.
CdrTfcrFirstInvalidFieldSelected	The focus changes because the first invalid field is selected.
CdrTfcrFocusRefreshed	The focus changes when the table selected is refreshed.
CdrTfcrFormLoaded	The focus is initialized because a form is loaded.
CdrTfcrMouseClicked	The focus changes because a cell is selected by a mouse click.
CdrTfcrRowsMerged	The focus changes because rows are merged.
CdrTfcrRowsRemoved	The focus changes because one or more table rows are removed.
CdrTfcrSelectionCopied	The focus changes because a user-selected area is copied from the active Cairo Viewer.
CdrTfcrTableCandidateChanged	The focus changes because a new table candidate is selected by the user.
CdrTfcrTabPressed	The focus changes because the user presses the Tab key or any arrow key.
CdrTfcrUnknownReason	The focus changes because of an unknown cause.
CdrTfcrWordCopied	The focus changes because a word is copied from the active Cairo Viewer.

4.2.1.16 CdrTableHeaderClickType

This type defines the possible events which can occur when the user clicks on a table header button.

Available Types	Description
CdrDoNothing	Lets the Verifier user decide to skip special processing altogether.
CdrColumnHeaderClicked	The user clicks a column header button.
CdrColumnHeaderDoubleClicked	The user double-clicks a column header button.
CdrColumnHeaderRightButtonClicked	The user right-clicks a column header button.
CdrRowHeaderClicked	The user clicks a row header button.

CdrRowHeaderDoubleClicked	The user double-clicks a row header button.
CdrRowHeaderRightButtonClicked	The user right-clicks a row header button.
CdrTableHeaderClicked	The user clicks the table header button.
CdrTableHeaderDoubleClicked	The user double-clicks the table header button.
CdrTableHeaderRightButtonClicked	The user right-clicks the table header button.

4.2.1.17 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.

4.2.1.18 CdrVerifierClassifyReason

These are the reasons for classification for 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.

4.2.1.19 CDRVerifierExceptionReason

This type defines the reasons for an exception event in Verifier.

Available Types	Description

CDRExceptionUserActivated	Exception induced by a user clicking on the Exception Handling button in Verifier.
CDRExceptionFieldValidated	Exception induced indirectly after field validation.
CDRExceptionDocumentValidated	Exception induced indirectly after document validation.
CDRExceptionUserAction	Exception induced indirectly by a user when executing an action in Verifier.
CDRExceptionManualClassification	Exception induced indirectly after manual classification.
CDRExceptionUnknown	Exception induced with unknown reason.
CDRExceptionUserActivatedForBatch	Exception induced by a user clicking on the Exception Handling for batches button in Verifier.

4.2.2 SCBCdrProject Methods and Properties

4.2.2.1 ActivateLicensing

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)`

Parameter	Description
ModuleName	A text string that represents the application activating licensing. Any value may be entered here.
LicensePath	A text string that contains the location of the license share file that will be queried. The path must be accessible from the location of the script execution and must point to the **Runtime.lic** file explicitly.

4.2.2.1.1 Sample Code

The following sample code returns licensing utilization information for active licensing counters.

```
'the Project represents the project library object.
 Dim theProject as New SCBCdrPROJLib.SCBCdrProject
 'The location of the shared license file to update.
 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)
```

4.2.2.2 AllClasses

This read-only property returns a collection of all defined document classes of this project.

Syntax: `AllClasses As ISCBCdrDocClasses`

4.2.2.3 BaseClasses

This read-only property returns a collection that contains all defined base document classes.

Syntax: `BaseClasses As ISCBCdrDocClasses`

4.2.2.4 ClassificationMode

This property sets or returns the used classification mode.

Syntax: `ClassificationMode As CDRClassifyMode`

4.2.2.5 CurrentClient

This property sets or returns the "Client" attribute of the batch.

Syntax: `CurrentClient as String`

4.2.2.6 DefaultClassifyResult

This property sets or returns the default document class name to which a document is redirected if no other document class fits.

Syntax: `DefaultClassifyResult As String`

4.2.2.7 DefaultLanguage

This read-only property returns the language used as default.

Syntax: `DefaultLanguage As String`

4.2.2.7.1 Sample Code

```
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
```

4.2.2.8 ExtractClassificationField

In case the ASSA field of the current document class is also the classification field of the project, this method extracts such field.

Note: If you use this method on a workdoc having the fields already extracted, the indexes of the fields might change. However, a new extraction of the whole document restores the original indexes.

Syntax: `ExtractClassificationField (pWorkdoc As ISCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	Current workdoc

4.2.2.9 Filename

This read-only property returns the file name of the project including the directory path.

Syntax: `Filename As String`

4.2.2.10 ForceValidation

If *ForceValidation* is set to *permitted,* then the Verifier user can overrule the validation by pressing three times on the [Enter] 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`

4.2.2.11 GetHostProperties

This method allows the user to get information about the current machine, program, and WebCenter Forms Recognition user.

Syntax:
```
GetHostProperties(appType As CDRApplicationName, appSubtype As Long,
appInstance As String, appUserName As String, appIP As String,
appMachineName As String, appLicensee As String)
```

Parameter	Description
`appType`	Type of the calling application. The parameter can be read from script.
`appSubType`	For internal use only
`appInstance`	Runtime Service instance name, if ApplicationName is TANRuntimeServer. Not used for other applications.
`appUsername`	Login name of the current WebCenter Forms Recognition user. WebCenter Forms Recognition user for Designer, Verifier, LSM, and Web Verifier Windows user for Runtime Server

appIP	IP address of the computer.
appMachineName	Machine name running the script.
appLicensee	Customer name of the used license file.

4.2.2.11.1 Sample Code

The following sample code calls the GetHostProperties in the initialize event. The method returns information into variables as to where the script is executed, who is executing it, and which program 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
```

4.2.2.12 GetVerifierProject

This method returns the Verifier project.

Syntax: `GetVerifierProject(ppVal As Object)`

Parameter	Description
ppVal	Returns the Verifier project object.

4.2.2.13 LastAddressPoolUpdate

This read-only property returns the last time when the address pool was updated.

Syntax: `LastAddressPoolUpdate As Date`

4.2.2.14 Lock

This method locks the project for updating.

Syntax: `Lock()`

4.2.2.15 LogScriptMessageEx

This method enables the developer to utilize the in-built functionality to output messages directly to the core product logs, administration console or System Monitoring notification.

Syntax: `LogScriptMessageEx(ByVal Type As CDRMessageType, ByVal Severity As CDRMessageSeverity, ByVal Message As String)`

Parameter	Description
`Type`	The *CdrMessageType* option to determine whether the message is classified to either an information, warning or error message.
`Severity`	Represents the severity code of the message. This option determines where the message appears, i.e. log, System Monitoring or as an email.
`Message`	The message text to display or send.

4.2.2.15.1 Sample Code

You can place the following sample code in any event. When the event triggers, a message is written to the core product log file (H_, D_, V_ or U_ log).

```
Project.LogScriptMessageEx(CDRTypeInfo, CDRSeverityLogFileOnly, "My message")

[Info] |30| 01:59:33.312 | 3108 | 668184k/1428344k | 514004k/3520792k |
57176k/67252k | 238 | 38/43 | My message)
```

4.2.2.16 MinClassificationDistance

This property sets or returns the minimal distance of classification results.

Syntax: `MinClassificationDistance As Double`

4.2.2.17 MinClassificationWeight

This property sets or returns the minimal classification weight.

Syntax: `MinClassificationWeight As Double`

4.2.2.18 MinParentClsDistance

This property sets or returns the minimal distance between the classification weight of the parent and the derived document classes.

Syntax: `MinParentClsDistance As Double`

4.2.2.19 MinParentClsWeight

This property sets or returns the minimal parent classification weight. This value is used as a threshold during parent classification.

Syntax: `MinParentClsWeight As Double`

4.2.2.20 MoveDocClass

This method moves a document class specified by its name to a new parent document class specified by `NewParentName`.

Syntax: `MoveDocClass(Name As String, NewParentName As String)`

Parameter	Description

Name	The name of the document class to move.
NewParentName	The name of the new parent document class.

4.2.2.21 NoUI

If this property is set to `True`, then no login dialog box displays.

Syntax: `NoUI As Boolean`

4.2.2.22 Page

This read-only property returns the Cairo Page object of the current project.

Syntax: `Page As ISCBCroPage`

4.2.2.23 ParentWindow

This write-only property sets the parent window of the login dialog box. Set the property value to the window handle of the operating system.

Syntax: `ParentWindow As Long`

4.2.2.24 PerformScriptCommandRTS

This method allows the developer to restart or stop the Runtime Server through a custom script.

This 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)
```

Parameter	Description
CommandID	Identifier of the command to execute on the RTS instance. Two commands that are currently supported: 0. Force the RTS instance to stop document processing. 1. Restart the RTS instance.
MessageType	The type of message to log when the command executes: 0. Informational message. 1. Warning message. 2. Error message. Note that error messages are additionally forwarded to the 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 administration console.

4.2.2.24.1 Sample Code

The following code example demonstrate how to stop and restart the RTS instance.

```
' script code stops document processing for the current Runtime Server
 ' instance and logs specified message as error with error code "777"
 Project.PerformScriptCommandRTS 0, 2, 777, "RTS is going to stop from Custom
Script"
 ' specified message as warning with error code "999"
 Project.PerformScriptCommandRTS 1, 1, 999, "RTS is going to be restarted from
Custom Script"
```

4.2.2.25 ReleaseAllAdsPools

This method releases the memory used by all Automatic Document Separation pools loaded in memory by RTS or Verifier.

Use this feature when the project has multiple large Automatic Document Separation pools from different classes that require a lot of memory. If the documents are sorted by class in different batches, only the required pools for a class are loaded in memory when processing the batch. The drawback is a potential decrease in performance due to the fact that the pools have to be reloaded each time a batch is processed.

Syntax: `Project.ReleaseAllAdsPools()`

4.2.2.25.1 Sample Code

The following sample code shows the implementation for RTS processing. It is placed in the Initialize event.

```
Private Sub ScriptModule_Initialize(ByVal ModuleName as String)
    Project.ReleaseAllAdsPools()
End Sub
```

The following sample code shows the implementation for Verifier and Web Verifier process. It is placed in the BatchOpen event.

```
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, ByVal BatchState as Long)
    Project.ReleaseAllAdsPools()
End Sub
```

The scripts above provide an entry in the log similar to the following.

```
[Info] |20| 14:05:26.812 | 7488 | 4820400k/3448008k | 5829788k/6631068k |
195812k/200160k | 543 | 73/57 | Disconnecting ADS Pool for class: Invoices,
field: VendorName
```

4.2.2.26 ReportLicensingStatus

This method retrieves 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.

The information is saved in the H, D, V or U log file.

Syntax: `ReportLicensingStatus(ReportActiveLicensingOnly As Boolean, Severity As SCBCdrPROJLib.CDRMessageSeverity)`

Parameter	Description
`ReportActive LicensingOnly`	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 (i.e. log file, email, or RTS System Monitoring).

4.2.2.26.1 Sample Code

The following code example demonstrates how to get licensing utilization information for all licensing counters.

```
Project represents the project library object.
 Dim theProject as New SCBCdrPROJLib.SCBCdrProject
 'The location of the shared license file to update.
 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)
 'Return all license counters
 theProject.ReportLicensingStatus(False,
SCBCdrPROJLib.CDRMessageSeverity.CDRSeverityEmailNotification)
 'Return only the active license counters
 theProject.ReportLicensingStatus(True,
SCBCdrPROJLib.CDRMessageSeverity.CDRSeverityEmailNotification)
```

4.2.2.26.2 Config File Output Example

```
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.
```

4.2.2.27 SecurityUpdateAddUserGroup and SecurityUpdateAddUserGroupPwd

This method updates or adds the database security credentials. This script call creates or updates the WebCenter Forms Recognition users, roles, and groups.

When updating the security policy of WebCenter Forms Recognition through a custom script, only the database tables update. You cannot modify the project security after a script update.

- Use the *SecurityUpdateAddUserGroupPwd* method to import user accounts with predefined passwords.
- Use this method between *SecurityUpdateStart* and *SecurityUpdateCommit*.

Note: If a user existing in the DB is not presented in SecurityUpdate, then the user is considered as being deleted from the system and marked as `deleted = true`.

The user is recovered and marked as "deleted = false" as soon as the user is present in SecurityUpdate.

The password updates only at creation or recovering of a user. If an administrator needs to change the password for a script imported user, the administrator first needs to exclude the user from the SecurityUpdate call so the user is deleted, and then re-add the user with a new password into the next iteration of the SecurityUpdate

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

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

Parameter	Description
`pContainer`	Container object used to save the validation templates and their settings.
`UserName`	The user name to create or update within the database. These are the user credentials to enter to log into the system. If Domain is populated, the user must enter MyDomain\UserName for logging in to the verification application.
`UserPassword`	This password applies only when creating or recovering a user. For those auto-imported users that were previously imported into WebCenter Forms Recognition, the password remains unchanged. Use case rules ▪ Auto-imported users with empty passwords are required to change their password upon first login. ▪ Auto-imported users with NON-empty passwords are NOT required to change their password upon first login. ▪ Auto-imported users who already changed their password upon first login are not required to change their password again.
`ExternalGroupID`	The external group ID is a security number. A batch and a user are assigned a group ID that enables the user to verify only batches that fall under the same group ID assigned to that user.
`UserRole`	The user role assigned to the Verifier user. The role can be one or a logical combination of the following text strings: ▪ **ADM**: Administrator ▪ **AEB**: Authorization for External Batches ▪ **SET**: Can access settings ▪ **VER**: Verifier user ▪ **SLV**: Verifier supervisor (learnset nomination) ▪ **SLM**: Learnset Manager (global learnset manager) ▪ **FLT**: Filtering The following combinations of roles are possible: Create a user with Verifier and Filter roles, but with no SET role: `Project.SecurityUpdateAddUserGroup "User2", 999, "VER|FLT", "BDomain"`

Create a user with Verifier, Settings and Filter roles:

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

Create a user with Verifier role only, with no SET and FLT role:

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

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

4.2.2.27.1 Sample Code

The example below updates the database user security on a regular basis. The script can be modified to lookup users/roles and update the WebCenter Forms Recognition user table.

```
Private Sub ScriptModule_UpdateSystemSecurity(ByVal InstanceName as String)
    Project.SecurityUpdateStart
    Project.SecurityUpdateAddUserGroup "User1", 777, "VER|SET", "BDomain"
    Project.SecurityUpdateAddUserGroup "User1", 999, "VER|SET", "BDomain"
    Project.SecurityUpdateAddUserGroup "User2", 222, "AEB", "BDomain"
    Project.SecurityUpdateAddUserGroup "User10", 777, "ADM", "BDomain"
    Project.SecurityUpdateAddUserGroupPwd ("User2", "pass", 777, "VER|FLT", "")
    Project.SecurityUpdateCommit
End Sub
```

4.2.2.28 SecurityUpdateCommit

This method completes the security update process. This script call is required to complete updating the WebCenter Forms Recognition users, roles, and groups.

When updating the security policy of WebCenter Forms Recognition through a custom script, only the database tables update. The project security is not modified after a script update.

Syntax: `Project.SecurityUpdateCommit`

4.2.2.28.1 Sample Code

The following sample code updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the WebCenter Forms Recognition 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, "VER|SET", "BDomain"
    Project.SecurityUpdateAddUserGroup "User6", 777, "VER|FLT", "BDomain"
    Project.SecurityUpdateAddUserGroup "User7", 333, "AEB", "BDomain"
    Project.SecurityUpdateAddUserGroup "User10", 777, "ADM", "BDomain"
    Project.SecurityUpdateCommit
End Sub
```

4.2.2.29 SecurityUpdateStart

This method instantiates the security update process. This script call is required to begin updating the WebCenter Forms Recognition users, roles, and groups.

When updating the security policy of WebCenter Forms Recognition through a custom script, only the database tables update. The project security is not modified after a script update.

Syntax: `Project.SecurityUpdateStart`

4.2.2.29.1 Sample Code

The following sample code updates the database user security on a regular basis. The script can be updated to lookup users/roles and update the WebCenter Forms Recognition 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, "VER|SET", "BDomain"
    Project.SecurityUpdateAddUserGroup "User6", 777, "VER|FLT", "BDomain"
    Project.SecurityUpdateAddUserGroup "User7", 333, "AEB", "BDomain"
    Project.SecurityUpdateAddUserGroup "User10", 777, "ADM", "BDomain"
    Project.SecurityUpdateCommit
End Sub
```

4.2.2.30 SecurityUpdateUserParameter

This method establishes default group settings in Web Verifier for script imported users that do not have the SET role so that they are able to load projects and jobs.

With this method implemented, the corresponding group is found and assigned to the user as the PrimaryUserGroup.

If the group or the user cannot be found, a corresponding error message is shown.

This method works with auto-imported users as well as with normal users.

Call this method between SecurityUpdateStart and SecurityUpdateCommit.

An administrator needs to configure the group settings in the Web Verifier settings.

Syntax: `SecurityUpdateUserParameter (BSTR UserName, BSTR UserDomain, BSTR ParameterName, VARIANT Param1, VARIANT Param2)`

Parameter	Description
`ParameterName`	PrimaryGroupID This parameter can have two variants: ▪ Param1: *GroupName* with Param2 as String that represents the group name displayed in the Web Verifier administrator group settings. *ExternalGroupID* with Param2 as Integer that represents the ExternalGroupID that was added in SecurityUpdateAddUserGroup or SecurityUpdateAddUserGroupPwd.

4.2.2.30.1 Sample Code

Add user A to two groups (100 and 101) and set his primary group for settings to be group 100.

Add user Domain\B to one group and set his primary group for settings to be Autoimport_100. This is the displayed name of the group 100 in the Web Verifier administrator group settings

```
Private Sub ScriptModule_UpdateSystemSecurity(ByVal InstanceName as String)
    Project.SecurityUpdateStart
```

```
    Project.SecurityUpdateAddUserGroupPwd("A", "pass", 100, "VER|FLT", "")
    Project.SecurityUpdateAddUserGroupPwd("A", "pass", 101, "VER|FLT", "")
    Project.SecurityUpdateUserParameter("A", "", "PrimaryGroupID",
"ExternalGroupID", 100)
    Project.SecurityUpdateAddUserGroup("B", 100, "VER", "Domain")
    Project.SecurityUpdateUserParameter("B", "Domain", "PrimaryGroupID",
"GroupName", "AutoImport_100")
    Project.SecurityUpdateCommit
End Sub
```

4.2.2.31 ShowValidationTemplates

This method displays the validation templates and their settings in a given container.

Syntax: `ShowValidationTemplates(pContainer As ISCBCdrPPGContainer)`

Parameter	Description
`pContainer`	Container object used to save the validation templates and their settings.

4.2.2.32 SLWDifferentResultsAction

This property sets or returns the action to complete if a template classification and supplier extraction has different results.

Syntax: `SLWDifferentResultsAction As CdrSLWDifferentResultsAction`

4.2.2.33 SLWSupplierInvalidDifferentClsResults

This property sets or returns a Boolean value identifying if a supplier field is made invalid when the template classification and supplier extraction have different results.

Syntax: `SLWSupplierInvalidIfDifferentClsResults As Boolean`

4.2.2.34 Unlock

This method unlocks the project after updating.

Syntax: `Unlock()`

4.2.2.35 UpdateAddressPool

This method updates the address analysis pool.

Syntax: `UpdateAddressPool()`

4.2.2.36 ValidationSettingsColl

This read-only property returns a collection of all activated validation engines.

Syntax: `ValidationSettingsColl As ISCBCroCollection`

4.2.2.37 ValidationTemplates

This read-only property returns a collection of all available validation templates.

Syntax: `ValidationTemplates As ISCBCroCollection`

4.2.2.38 VersionCount

This read-only property returns the number of versions available for a specified file name.

Syntax: `VersionCount(Filename As String) As Long`

Parameter	Description
`Filename`	The name of the file.

4.2.2.39 WordSegmentationChars

This property sets or returns a string that contains all characters used for word segmentation.

Syntax: `WordSegmentationChars As String`

4.3 SCBCdrDocClass

4.3.1 Description

A *DocClass* object represents a single document class within a project class hierarchy.

4.3.2 Methods and Properties

4.3.2.1 ClassificationField

Use this property to read or write the name of the field that is used for the classification.

Syntax: `ClassificationField As String`

4.3.2.2 ClassificationRedirection

This property sets or returns the name of the target *DocClass*.

Syntax: `ClassificationRedirection As String`

4.3.2.3 ClassifySettings

This read-only property returns a collection of chosen classification engines and their settings for this *DocClass*.

Syntax: `ClassifySettings As ISCBCroCollection`

4.3.2.4 DerivedDocClasses

This read-only property returns a collection of all document classes derived directly from this *DocClass*.

Syntax: `DerivedDocClasses As ISCBCdrDocClasses`

4.3.2.5 DisplayName

This property sets or returns the display name of the document class currently in use. If nothing is inserted here, the *DocClass* name is used.

Syntax: `DisplayName As String`

4.3.2.6 Fields

This read-only property provides access to the *FieldDefs* collection of a document class.

Syntax: `Fields As ISCBCdrFieldDefs`

4.3.2.7 ForceSubtreeClassification

This property sets or returns whether the classification to the subtree of this document class is forced.

Syntax: `ForceSubtreeClassification As Boolean`

4.3.2.8 ForceValidation

If this property is set to *permitted,* then the Verifier user can overrule the validation by pressing three times on the [Enter] 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`

4.3.2.9 GetFieldAnalysisSettings

This method returns the analysis settings for the document class.

Syntax:
```
GetFieldAnalysisSettings(FieldName As String, Language As String,
ppAnalysisSettings As ISCBCdrAnalysisSettings)
```

Parameter	Description
`FieldName`	The name of the field for which the analysis settings are retrieved.
`Language`	String.
`ppAnalysisSettings`	The name of the analysis settings object that is used in the code to assign the settings to.

4.3.2.9.1 Sample Code

The following sample code shows how to get 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
```

4.3.2.10 Hidden

This property specifies whether the document class should be visible in the Designer application.

Syntax: `Hidden As Boolean`

4.3.2.11 InitField

This method reinitializes a required field in the workdoc.

Syntax: `InitField(pWorkdoc As ISCBCdrWorkdoc, pField As ISCBCdrField)`

Parameter	Description
`pWorkdoc`	The current workdoc object.
`pField`	The field object for the field to be cleared.

4.3.2.12 ManualTableTrainingMode

This property sets or returns the option for manual table extraction training mode.

Syntax: `ManualTableTrainingMode As Boolean`

4.3.2.13 Name

This property reads or writes the name of the document class.

Syntax: `Name As String`

4.3.2.14 Page

This read-only property returns the *Page* object of this document class, with all defined zones and their OCR settings.

Syntax: `Page As ISCBCroPage`

4.3.2.15 Parent

This property returns the parent document class of the actual document class.

Syntax: `Parent As ISCBCdrDocClass`

4.3.2.16 ShowClassValidationDlg

This method displays the property page validation settings for this document class.

Syntax: `ShowClassValidationDlg(pContainer As ISCBCdrPPGContainer)`

Parameter	Description
pContainer	Container in which the property page displays.

4.3.2.17 ShowFieldValidationDlg

This method displays the property page of the validation settings for the specified field name.

Syntax: `ShowFieldValidationDlg(FieldName As String, pContainer As ISCBCdrPPGContainer)`

Parameter	Description
FieldName	Field for which the dialog is shown.
pContainer	Container in which the property page displays.

4.3.2.18 ShowGeneralFieldPPG

This method starts the field settings property page specifying the active tab.

Syntax: `ShowGeneralFieldPPG(FieldName As String, TabIndexActive As Long, pContainer As ISCBCdrPPGContainer)`

Parameter	Description
FieldName	Field for which the dialog is shown.
TabIndexActive	Zero-based index for the tab that displays.
pContainer	Container in which the property page displays.

4.3.2.19 SubtreeClsMinDist

This property sets or returns the minimal distance to the classification weight of the derived document classes.

Syntax: `SubtreeClsMinDist As Double`

4.3.2.20 SubtreeClsMinWeight

This property sets or returns the minimal classification weight of the derived document classes.

Syntax: `SubtreeClsMinWeight As Double`

4.3.2.21 UseDerivedValidation

This property sets or returns a Boolean value, when derived validation rules are used.

Syntax: `UseDerivedValidation As Boolean`

4.3.2.22 ValidationSettingsColl

This read-only property returns a collection of all activated validation engines.

Syntax: `ValidationSettingsColl As ISCBCroCollection`

4.3.2.23 ValidationTemplateName

This property sets or returns the name of the validation template.

Syntax: `ValidationTemplateName As String`

4.3.2.24 ValidClassificationResult

This property sets or returns if this document class is a valid classification result or if it is omitted for classification.

Syntax: `ValidClassificationResult As Boolean`

4.3.2.25 VisibleInCorrection

This property determines if a project class is available for classification. You can modify this property prior to classification correction for Verifier by setting the property to **True** if the class is available for classification correction, or **False** if the class is unavailable for classification correction.

Dynamic modification of this property is managed through the *ScriptModule_VerifierClassify* event. Dynamic modification of the class visibility overrides the default Designer class property.

Syntax: `VisibleInCorrection As Boolean`

4.3.2.25.1 Sample Code

The following sample code 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
```

4.4 SCBCdrDocClasses

4.4.1 Description

This collection contains all defined *DocClass* objects of the project.

4.4.2 Methods and Properties

4.4.2.1 Collection

This read-only property returns the collection that is internally used to store the *DocClasses*.

Syntax: `Collection As ISCBCroCollection`

4.4.2.2 Count

This read-only property returns the number of items within the collection.

Syntax: `Count As Long`

4.4.2.3 Item

This read-only property returns a specified item from the collection.

Syntax: `Item(Index As Variant) As ISCBCdrDocClass`

Parameter	Description
`Index`	The index can either be a *Long* value specifying the index within the collection or a *String* specifying the item by name.

4.4.2.4 ItemByIndex

This read-only property returns an item from the collection specified by the index.

Syntax: `ItemByIndex(Index As Long) As ISCBCdrDocClass`

Parameter	Description
`Index`	Index of the item to retrieve from the collection. Valid range from **1** to *Count*.

4.4.2.5 ItemByName

This read-only property returns an item from the collection specified by name.

Syntax: `ItemByName(Name As String) As ISCBCdrDocClass`

Parameter	Description
`Name`	The name of the item to retrieve from the collection.

4.4.2.6 ItemExists

This method returns **True** if an item with a specified names exists inside the collection, otherwise **False** is returned.

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

Parameter	Description
`Name`	The name of the item to search for.

4.4.2.7 ItemIndex

This read-only property returns the index of an item specified by name.

Syntax: `ItemIndex(Name As String) As Long`

Parameter	Description
`Name`	Name specifying an item in the collection.

4.4.2.8 ItemName

This read-only property returns the name of an item specified by the index.

Syntax: `ItemName(Index As Long) As String`

Parameter	Description
`Index`	Index specifying an item in the collection. Valid range from **1** to *Count*.

4.4.2.9 Tag

Use this property to store a variant for each item of the collection.

Syntax: `Tag(Index As Long) As Variant`

Parameter	Description
`Index`	Index specifying an item in the collection. Valid range from **1** to *Count*.

4.5 SCBCdrFieldDef

4.5.1 Description

A *FieldDef* object represents the definition of a single field inside a document class.

4.5.2 Methods and Properties

4.5.2.1 AllowDelayedValidation

This property sets or returns if "Allow delayed validation" is enabled.

Syntax: `AllowDelayedValidation As Boolean`

4.5.2.2 AllowInteractiveExtraction

This property sets or returns if table correction is enabled.

`True`: Table correction in Verifier is enabled, but the table is not impacted by learning.

`False`: Table correction in Verifier is disabled.

Syntax: `AllowInteractiveExtraction As Boolean`

4.5.2.2.1 Sample Code

The following sample code looks at a field and determines if the LineItems (Brainware Table Extraction Field) are configured for automatic extraction.

```
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.ItemByName("LineItem
s").AllowInteractiveExtraction = False
    Else

Project.AllClasses.ItemByName(pWorkdoc.DocClassName).Fields.ItemByName("LineItem
s").AllowInteractiveExtraction = True
     End If
End Sub
```

4.5.2.3 AlwaysValid

This property sets or returns if the content of this field is always valid.

Syntax: `AlwaysValid As Boolean`

4.5.2.4 AnalysisTemplate

This read-only property returns the name of the analysis template if used.

Syntax: `AnalysisTemplate(Language As String) As String`

Parameter	Description

`Language`	Language parameter.

4.5.2.5 AppendListItem

This method adds a new list item and returns a new item index for it.

Syntax: `AppendListItem(bstrItem As String) As Long`

Parameter	Description
`bstrItem`	String appended to the list.

4.5.2.6 BoostDigitsOnly

This property sets or returns whether only digits should be boosted.

Syntax: `BoostDigitsOnly as Boolean`

4.5.2.7 BoostField

This property sets or returns whether a field should be boosted.

Syntax: `BoostField as Boolean`

4.5.2.8 ColumnCount

This read-only property returns the number of table columns if *FieldType* is **Table**.

Syntax: `ColumnCount As Long`

4.5.2.9 ColumnName

This read-only property returns the name of the table column specified by an index if *FieldType* is **Table**.

Syntax: `ColumnName(ColumnIndex As Long) As String`

Parameter	Description
`ColumnIndex`	Zero-based index of the table column.

4.5.2.10 DefaultValidationSettings

This read-only property returns the validation settings with the default language.

Syntax: `DefaultValidationSettings As ISCBCdrValidationSettings`

4.5.2.11 Derived

This read-only property returns **True** if the field properties are derived from an upper document class.

Syntax: `Derived As Boolean`

4.5.2.12 DisplayName

The display name can be different from the field name and does not have any restrictions about the used character set, while the field name must be a valid basic name. An application may use the display name instead of the field name to show a more readable name of the field.

Syntax: `DisplayName As String`

4.5.2.13 EvalSetting

This property sets or returns the activated evaluation engine and its settings.

Syntax: `EvalSetting(Language As String) As Object`

Parameter	Description
`Language`	Language parameter.

4.5.2.14 EvalTemplate

This read-only property returns the name of the evaluation template if used.

Syntax: `EvalTemplate(Language As String) As String`

Parameter	Description
`Language`	Language parameter.

4.5.2.15 FieldID

This read-only property returns the internally used field ID.

Syntax: `FieldID As Long`

4.5.2.16 FieldType

This property sets or returns the type of the field.

Syntax: `FieldType As` *`CDRFieldType`*

4.5.2.17 ForceValidation

This property sets or returns the mode for the Force Validation.

Syntax: `ForceValidation As` *`CdrForceValidationMode`*

4.5.2.18 GetAssignedAnalysisEngineName

This method returns the name of the assigned analysis engine or NULL if no engine is assigned.

Syntax: `GetAssignedAnalysisEngineName(Language As String, EngineName As String`

Parameter	Description
`Language`	Language name **Note:** The default language of the project is *German*.
`EngineName`	Name of the analysis engine

4.5.2.18.1 Sample Code

The following code example reads the assigned analysis engine name and displays it in a message box.

```
Private Sub Document_OnAction(pWorkdoc As SCBCdrPROJLib.ISCBCdrWorkdoc, ByVal
ActionName As String)
    Dim strEngineName As String
    If ActionName = "ShowAnalysisEngineName" Then

Project.AllClasses.ItemByName("Invoices").Fields.ItemByName("Date").GetAssignedA
nalysisEngineName(Project.DefaultLanguage, strEngineName)
       MsgBox strEngineName
    End If
End Sub
```

4.5.2.19 ListItem

This property sets or returns a list item string for a given index.

Syntax: `ListItem(lIndex As Long) As String`

Parameter	Description
`lIndex`	Zero-based index.

4.5.2.20 ListItemCount

This read-only property returns the number of strings in the *ListItem* list.

Syntax: `ListItemCount As Long`

4.5.2.20.1 Sample Code

```
Dim lngItem as Long
 For lngItem =
Project.AllClasses.ItemByName("Invoice").Fields("Currency").ListItemCount - 1 To
0 Step -1
```

4.5.2.21 MaxLength

This property sets or returns the maximum number of characters permitted for this field.

Syntax: `MaxLength As Long`

4.5.2.22 MinLength

This property sets or returns the minimal number of characters permitted for this field.

Syntax: `MinLength As Long`

4.5.2.23 Name

This property sets or returns the name of the field.

Syntax: `Name As String`

4.5.2.24 NoRejects

This property sets or returns if rejects are permitted.

Syntax: `NoRejects As Boolean`

4.5.2.25 OCRConfidence

This property sets or returns the confidence level for OCR. The value must be between **0** and **100**.

Syntax: `OCRConfidence As Long`

4.5.2.26 RemoveListItem

This method removes a list item by its index.

Syntax: `RemoveListItem(lIndex As Long)`

Parameter	Description
`lIndex`	The index of the item to be removed from the list.

4.5.2.27 SmartIndex

This property contains all definitions about smart indexing.

Syntax: `SmartIndex As ISCBCdrSmartIndex`

4.5.2.28 UseDerivedOCRSettings

This property sets or returns if OCR settings of the parent document class are used.

Syntax: `UseDerivedOCRSettings As Boolean`

4.5.2.29 UserDerivedValidation

This property sets or returns if the derived validation rules are used for validation of this field.

Syntax: `UseDerivedValidation As Boolean`

4.5.2.30 UseMaxLen

This property sets or returns if the maximum number of characters is limited to the value given by *MaxLength*.

Syntax: `UseMaxLen As Boolean`

4.5.2.31 UseMinLen

This property sets or returns if the minimum number of characters is defined by the value given by *MinLength*.

Syntax: `UseMinLen As Boolean`

4.5.2.32 ValidationSettings

This property sets or returns the chosen validation engine and its settings.

Syntax: `ValidationSettings(Language As String) As ISCBCdrValidationSettings`

Parameter	Description
`Language`	Defines the language for classification, extraction and validation.

4.5.2.33 ValidationTemplate

This read-only property returns the name of the validation template.

Syntax: `ValidationTemplate(Language As String) As String`

Parameter	Description
`Language`	Defines the language for classification, extraction and validation.

4.5.2.34 ValidationType

This read-only property returns the type of validation.

Syntax: `ValidationType As CdrValFieldType`

4.5.2.35 VerifierColumnWidth

This property sets or returns the width of the specified column of the table.

Syntax: `VerifierColumnWidth(ColumnIndex As Long) As Long`

Parameter	Description
`ColumnIndex`	Zero-based Index of the table column.

4.6 SCBCdrFieldDefs

4.6.1 Description

This collection contains all defined *FieldDef* objects of a single document class.

4.6.2 Methods and Properties

4.6.2.1 Collection

This read-only property returns the collection that is internally used to store the *FieldDef* objects.

Syntax: `Collection As ISCBCroCollection`

4.6.2.2 Count

This read-only property returns the number of items within the *FieldDef* collection.

Syntax: `Count As Long`

4.6.2.3 Item

This read-only property returns a specified item from the collection

Syntax: `Item(Index As Variant) As ISCBCdrFieldDef`

Parameter	Description
`Index`	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.

4.6.2.4 ItemByIndex

This read-only property returns an item from the collection specified by index.

Syntax: `ItemByIndex(Index As Long) As ISCBCdrFieldDef`

Parameter	Description
`Index`	Index of the item to retrieve from the collection.

4.6.2.5 ItemByName

This read-only property returns an item from the collection specified by name.

Syntax: `ItemByName(Name As String) As ISCBCdrFieldDef`

Parameter	Description
`Name`	Name of the item to retrieve from the collection.

4.6.2.6 ItemExists

This method returns **True** if an item with specified name exists inside the collection, otherwise **False** is returned.

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

Parameter	Description
`Name`	The name of the item to search for.

4.6.2.7 ItemIndex

This read-only property returns the index of an item specified by name.

Syntax: `ItemIndex(Name As String) As Long`

Parameter	Description
`Name`	Name specifying an item in the collection.

4.6.2.8 ItemName

This read-only property returns the name of an item specified by index.

Syntax: `ItemName(Index As Long) As String`

Parameter	Description
`Index`	Index specifying an item in the collection. Valid range from 1 to *Count*.

4.6.2.9 Tag

Use this property to store a variant for each item of the collection.

Syntax: `Tag(Index As Long) As Variant`

Parameter	Description
`Index`	Index specifying an item in the collection. Valid range from 1 to *Count*.

4.7 SCBCdrLicenseInfoAccess

4.7.1 Description

The Licensing Information Access object allows direct retrieval to the active licensing object. The developer is able to directly query any licensing component in a custom script.

4.7.2 Methods

4.7.2.1 GetLicenseCounterByID

This method returns the license counter information for any given active or inactive license counter.

An active 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)`

Parameter	Description
`CounterID`	The ID of the counter from which the value is retrieved. 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.

4.7.2.1.1 Sample Code

The following sample code returns the OCRed number of documents.

```
Dim theLicensingInterface2 as SCBCdrPROJLib.SCBCdrLicenseInfoAccessDim
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)\[CompanyName]\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")
```

4.7.2.2 GetLicenseCounterByName

This method returns the license counter information for any given active or 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)`

Parameter	Description
CounterName	The name of the counter from which the value is retrieved. 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.

4.7.2.2.1 Sample Code

The following sample code returns the OCRed count of documents in script.

```
Dim theLicensingInterface 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)
```

4.7.2.3 GetLicenseValueById

This method returns the license counter information for any given item in the license file.

Syntax: `GetLicenseValueByID(PropertyID As SCBCdrPROJLib.CDRLicenseFeatureName, Value As Variant)`

Parameter	Description
PropertyID	Depicts the item for which to retrieve the values.
Value	The returned value from the licensing mechanism. The data type varies depending on the item being returned.

4.7.2.3.1 Sample Code

The following sample code returns the Email Importing flag in the license file.

```
Dim theLicensingInterface 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")
```

4.7.2.4 GetLicenseValueByName

This method returns the license counter information for any given item in the license file.

Syntax: `GetLicenseValueByName(PropertyName As String, Value As Variant)`

Parameter	Description
PropertyName	Depicts the name on which to retrieve values. 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.

4.7.2.4.1 Sample Code

The following sample code returns the Email Importing flag in the license file.

```
Dim theLicensingInterface 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")
```

4.8 SCBCdrSettings

4.8.1 Description

The *Settings* object stores arbitrary strings for usage in script.

4.8.2 Methods and Properties

4.8.2.1 ActiveClient

This property sets or returns the name of the currently active client.

Syntax: `ActiveClient As String`

4.8.2.2 AddClient

This method adds a new client with the specified name to the current *Settings* object.

Syntax: `AddClient(newVal As String)`

Parameter	Description
newVal	Name of the new client.

4.8.2.3 AddKey

This method adds a new key specified by its name and its parent. See the *Oracle WebCenter Forms Recognition Designer User Guide* for more information.

Syntax: `AddKey(newVal As String, Parent As String)`

Parameter	Description
`newVal`	New key name.
`Parent`	Name of the parent key. In case of a new base key, use an empty string for *Parent*.

4.8.2.4 Clear

This method clears all clients and keys from the *Settings* object.

Syntax: `Clear()`

4.8.2.5 Client

This read-only property returns the name of the specified client.

Syntax: `Client(Index As Long) As String`

Parameter	Description
`Index`	Zero-based index of the client.

4.8.2.6 ClientCount

This read-only property returns the number of clients

Syntax: `ClientCount As Long`

4.8.2.7 DocumentBinarizationMode

This property sets or returns the binarization mode for document conversion in the recognition engine.

Syntax: `DocumentBinarizationMode as CdrDocumentBinarizationMode`

4.8.2.8 GlobalLearnsetPath

This property sets or returns the global learnset path.

Syntax: `GlobalLearnsetPath As String`

4.8.2.9 Key

This read-only property returns the key name specified by index.

Syntax: `Key(Index As Long) As String`

Parameter	Description
`Index`	Zero-based index of the key.

4.8.2.10 KeyCount

This read-only property returns the number of keys.

Syntax: `KeyCount As Long`

4.8.2.11 KeyIcon

This property sets the new value for the specified key or returns the key's value.

Syntax: `KeyIcon(Key As String)`

Parameter	Description
`Key`	The name of the key.

4.8.2.12 KeyParent

This read-only property returns the parent name of the specified key index.

Syntax: `KeyParent(Index As Long) As String`

Parameter	Description
`Index`	Zero-based index of the key.

4.8.2.13 MoveKey

This method moves a key specified by its name to the *NewParent* specified by its name.

Syntax: `MoveKey(Key As String, NewParent As String)`

Parameter	Description
`Key`	Name of the key that should be moved.
`NewParent`	Name of the new parent. In the case of moving it as a base key, this parameter should be left empty.

4.8.2.14 ProjectFileName

This property sets or returns the file name of the project.

Syntax: `ProjectFileName As String`

4.8.2.15 RemoveClient

This method removes a client specified by its name.

Syntax: `RemoveClient(ClientName As String)`

Parameter	Description
ClientName	Name of the client that should be removed.

4.8.2.16 RemoveKey

This method removes a key specified by its name.

Syntax: `RemoveKey(KeyName As String)`

Parameter	Description
KeyName	The name of the key that should be removed.

4.8.2.17 SupervisedLearningDisabled

This property sets or returns the state of supervised learning in Designer and Verifier workstations.

Syntax: `SupervisedLearningDisabled As Boolean`

4.8.2.18 TopDownEventSequence

This property sets or returns the value of a top-down event sequence.

Syntax: `TopDownEventSequence As Boolean`

4.8.2.19 Value

This property returns the value of the specified key.

Syntax: `Value(Key As String, Parent As String, Client As String) As String`

Parameter	Description
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 is used.

4.8.2.19.1 Sample Code

```
MyDBPath = Settings.Value(" DatabaseName ", " ", " ")
 'now we can open the database
 DB.Open(MyDBPath, …)
```

4.9 SCBCdrScriptModule

4.9.1 Description

This is a global object at the project level. All script module events occurring at project level belong to this object.

4.9.2 Methods and Properties

4.9.2.1 ModuleName

This read-only property 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: **Server**
- Web Verifier Client: **Verifier**
- Verifier Application: **Verifier**
- Local Verifier Project: **LocalVerifier**
- Learnset Manager: **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`

4.9.2.1.1 Sample Code

The following sample code 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
```

The following function 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
```

4.9.2.2 ReadZone

Use this method to read a zone on a *CroImage* object.

Syntax: `ReadZone(Image As ISCBCroImage, ZoneName As String) As String`

Parameter	Description
`Image`	*SCBCroImage* object.
`ZoneName`	Name of read zone.

4.9.2.3 ReadZoneEx

Use this method to read a zone on a *CroImage* object.

Syntax: `ReadZoneEx(Image As ISCBCroImage, ZoneName As String, Result As ISCBCroWorktext)`

Parameter	Description
`Image`	*SCBCroImage* object.
`ZoneName`	Name of read zone.
`Result`	Result of reading returned as *SCBCroWorktext* object.

4.10 SCBCdrScriptAccess

4.10.1 Description

WebCenter Forms Recognition provides a public interface *SCBCdrScriptAccess* for external access to the project and class level custom script pages. The interface can be queried from the main *SCBCdrProject* interface available in WebCenter Forms Recognition custom script. Using this interface it is possible to retrieve, modify and dump project and class level scripts.

4.10.2 Methods and Properties

4.10.2.1 DumpAllPages

This method dumps all script pages available in the project as a Unicode text file.

Syntax: `DumpAllPages(FileName As String)`

Parameter	Description
`FileName`	Name of the dump file.

4.10.2.2 GetPageCode

This method retrieves the project or specified class level script code.

Syntax: `GetPageCode(ClassName As String, ScriptCode As String)`

Parameter	Description
`ClassName`	Name of the class.
`ScriptCode`	Class script code.

4.10.2.3 SetPageCode

This method assigns the project or specified class level script code.

Syntax: `SetPageCode(ClassName As String, ScriptCode As String)`

Parameter	Description
`ClassName`	Name of the class.
`ScriptCode`	Class script code.

4.10.2.3.1 Sample Code

The following sample code sets the script code to blank.

```
theScriptAccess.SetPageCode(strClassName, "")
```

5 CDRADSLib

5.1 SCBCdrSupExSettings

5.1.1 Description

This collection contains the functions for the Associative Search engine.

5.1.2 Methods and Properties

5.1.2.1 AddFilterAttribute

This method adds new filters for the 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(Key As String, Value As String)`

Parameter	Description
`Key`	Name of the attribute to be filtered.
`Value`	Value of the attribute that is searched for in the datasource.

5.1.2.1.1 Sample Code

The following sample code configures the multi-column attribute filtering to be used with the Vendor Search button in Verifier. The Vendor Search 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
    theDocClass.GetFieldAnalysisSettings "VendorName","German",
theAnalysisSettings
```

```
    Set theObject = theAnalysisSettings
    Set theSupplierSettings = theObject
    theSupplierSettings.ClearFilterAttributes()
    theSupplierSettings.AddFilterAttribute "SupplierName", "VAN"
    theSupplierSettings.AddFilterAttribute "SupplierName", "VAN3"
End Sub
```

5.1.2.2 ClearFilterAttributes

This method clears all existing filters of the multi-column attribute Search.

Syntax: `ClearFilterAttributes()`

5.1.2.2.1 Sample Code

```
Dim theSupplierSettings as Object
 Set theSupplierSettings = FieldAnalysissettings
 Dim theAdsSettings as CDRADSLib.SCBCdrSupExSettings
 Set theAdsSettings = theSupplierSettings
 theAdsSettings.ClearFilterAttributes
```

5.1.2.3 PoolVersion

This property sets or returns the pool version.

Possible Values:

- **2:** Brainware V2
- **3:** Brainware V3
- **4:** RMS

Syntax: `PoolVersion As Long`

6 Analysis Engines Object Reference

6.1 SCBCdrAssociativeDbExtractionSettings

6.1.1 Description

This interface covers all methods and properties that are required for controlling and accessing the universal format of the ASSA engine's pool.

6.1.2 Type Definitions

6.1.2.1 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.

6.1.3 Methods and Properties

6.1.3.1 AddColumn

This method adds a new column field to the pool.

Syntax:
```
AddColumn ColumnName As String, IsSearchField As Boolean,
NewColumnIndex As Long)
```

Parameter	Description
`ColumnName`	Name of the column field.
`IsSearchField`	Boolean value that has to be set to true when the inserted column field is a search field.
`NewColumnIndex`	Returns the index of the newly created entry in the pool.

6.1.3.2 AddPhrase

This method appends a new phrase to the list of phrases to use for the address.

Syntax:
```
AddPhrase(Phrase As String, IsIncludePhrase As Boolean)
```

Parameter	Description
`Phrase`	This string variable contains the phrase that is added to the list.

`IsIncludePhrase`	If the value of the Boolean variable is **True** and the phrase is found, then the resulting address is accepted. If the value of the Boolean variable is **False** and the phrase is found, then the address is not accepted.

6.1.3.3 ChangeEntry

This method updates or inserts the content of the entry data to the specified column.

Syntax: `ChangeEntry(ColumnName As String, EntryData As String)`

Parameter	Description
`ColumnName`	Name of the column that is changed.
`EntryData`	The content of the specified column is updated with this data.

6.1.3.4 ClassNameFormat

This property sets or returns the format definition for a document class name.

Syntax: `ClassNameFormat As String`

6.1.3.5 ColumnCount

This read-only property returns the number of columns of a currently opened pool.

Syntax: `ColumnCount As Long`

6.1.3.6 ColumnName

This property returns or sets the name of the column by its index.

Syntax: `ColumnName(ColumnIndex As Long) As String`

Parameter	Description
`ColumnIndex`	Zero-based index of the column to retrieve.

6.1.3.7 CommitAddEntry

This method takes effect after execution of *StartAddEntry* and *ChangeEntry*.

Use this method only in context with the *StartUpdate*, *StartAddEntry*, *ChangeEntry* and *CommitUpdate* methods.

Syntax: `CommitAddEntry(NewIndex As Long)`

Parameter	Description
`NewIndex`	Returns the index of the new entry.

6.1.3.8 CommitUpdate

This method closes and saves the currently opened pool.

Syntax: `CommitUpdate()`

6.1.3.9 EnableCandidateEvaluation

This property sets or returns if a candidate evaluation (so-called *Second Pass*) is permitted. The following three options are available:

- Above configured search areas (*EvalOverSearchAreas* is set to **True**).
- First page only (*EvalFirstPageOnly* is set to **True**).
- All pages of the document. Evaluation is performed using the entire text of the document, which is performed if neither of the above restrictions is true.

The *EvalOverSearchAreas* and *EvalFirstPageOnly* restrictions are mutually exclusive. Therefore, when setting one to **True**, the other one automatically becomes **False**.

Note: If candidate evaluation is disabled, then candidates returned after the first pass typically have very low confidence.

Syntax: `EnableCandidateEvaluation As Boolean`

6.1.3.10 EntryCount

This read-only property returns the number of entries of the pool.

Syntax: `EntryCount As Long`

6.1.3.11 EvalFirstPageOnly

This property sets or returns if candidate evaluation is performed using the text from first page only.

When *EvalFirstPageOnly* is set to **True**, the *EvalOverSearchAreas* property becomes **False** automatically.

Syntax: `EvalFirstPageOnly As Boolean`

6.1.3.12 EvalOverSearchAreas

This property sets or returns if the candidate evaluation is processed using only the text from configured search areas.

When *EvalOverSearchAreas* is set to **True**, the *EvalFirstPageOnly* property becomes **False** automatically.

Syntax: `EvalOverSearchAreas As Boolean`

6.1.3.13 FieldContentsFormat

This property sets or returns the format definition for the representation of the engine.

Syntax: `FieldContentsFormat As String`

6.1.3.14 FindLocation

This property sets or returns if address analysis is enabled. If **True**, the position of the address is found.

Syntax: `FindLocation As Boolean`

6.1.3.15 GeneratePool

This method imports the pool from the source specified in the *AutomaticImportMethod* property.

Syntax: `GeneratePool()`

6.1.3.16 GeneratePoolFromCsvFile

This method removes the previous pool and generates a new one using the CSV file, designed in the new format.

Syntax: `GeneratePoolFromCsvFile()`

6.1.3.17 GeneratePoolFromODBC

This method removes the previous pool and generates a new one using the ODBC source with the parameters set on the property page.

Syntax: `GeneratePoolFromODBC()`

6.1.3.18 GetClassNameByID

This method 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)`

Parameter	Description
`IdHigh`	The upper part of the 64-bit unique ID.
`IdLow`	The lower part of the 64-bit unique ID.
`ClassName`	Returns the formatted document class name for the specified entry.

6.1.3.19 GetEntry

This method 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`

Parameter	Description

`Index`	Index of the entry to be retrieved.
`FieldName`	Name of the column to be retrieved.

6.1.3.20 GetFormattedValueByID

This method 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)`

Parameter	Description
`IdHigh`	The upper part of the 64-bit unique ID.
`IdLow`	The lower part of the 64-bit unique ID.
`FormattedValue`	Returns the formatted entry representation for the specified entry.

6.1.3.21 GetIDByIndex

This method returns the unique ID of an entry by index.

Syntax: `GetIDByIndex(Index As Long, IdHigh As Long, IdLow As Long)`

Parameter	Description
`Index`	Zero-based index of the entry.
`IdHigh`	Returns the upper part of the 64-bit unique ID.
`IdLow`	Returns the lower part of the 64-bit unique ID.

6.1.3.22 GetIndexById

This method returns the index of an entry by its unique ID.

Syntax: `GetIndexByID(IdHigh As Long, IdLow As Long, Index As Long)`

Parameter	Description
`IdHigh`	The upper part of the 64-bit unique ID.
`IdLow`	The lower part of the 64-bit unique ID.
`Index`	Returns the index of the entry.

6.1.3.23 GetSearchArea

This method returns an area on the document in which to search.

Syntax: `GetSearchArea(SearchAreaIndex As Long, Left As Long, Top As Long, Width As Long, Height As Long)`

Parameter	Description
`SearchAreaIndex`	Index of the area. Accepts values from **0** to **5** with following meaning: 0. First page header. 1. First page footer. 2. Subsequent pages header. 3. Subsequent pages footer. 4. Last page header. 5. Last page footer.
`Left`	Distance as a percentage from the left border of the document.
`Top`	Distance as a percentage from the top of the document.
`Width`	Width as a percentage of the search area.
`Height`	Height as a percentage of the search area.

6.1.3.24 IdentityColumn

This property sets or returns the unique ID of a column name.

Syntax: `IdentityColumn As String`

6.1.3.25 ImportFieldNames

This property sets or returns if the column names are taken from the first line of a CSV file.

Syntax: `ImportFieldNames As Boolean`

6.1.3.26 ImportFileName

This property sets or returns the filename of the CSV file.

Syntax: `ImportFileName As String`

6.1.3.27 ImportFileNameRelative

This property sets or returns if the name of a CSV file is stored relative to the path of the project file.

Syntax: `ImportFileNameRelative As Boolean`

6.1.3.28 IsPhraseIncluded

This property sets or returns if a phrase to find the address is sufficient.

Syntax: `IsPhraseIncluded(PhraseIndex As Long) As Boolean`

Parameter	Description
`PhraseIndex`	Zero-based index of phrase.

6.1.3.29 IsSearchField

This property sets or returns if a field is used for an associative search.

Syntax: `IsSearchField(ColumnIndex As Long) As Boolean`

Parameter	Description
`ColumnIndex`	Zero-based index of the column.

6.1.3.30 LastImportTimeStamp

This read-only property returns the timestamp for the last import.

Syntax: `LastImportTimeStamp As Date`

6.1.3.31 MaxCandidates

This property sets or returns the maximum number of results of the associative search engine.

Syntax: `MaxCandidates As Long`

6.1.3.32 MinDistance

This property sets or returns the required minimum distance to the next best candidate for a valid result.

Syntax: `MinDistance As Double`

6.1.3.33 MinRelevance

This property sets or returns the minimum relevance for search results. The default value is **0.0**.

Syntax: `MinRelevance As Double`

6.1.3.34 MinThreshold

This property sets or returns the required minimum value for a valid engine result.

Syntax: `MinThreshold As Double`

6.1.3.35 NumberKeepLocalCopies

Use this property to allow additional local copies of the ASE pool for later reuse.

The default value is 1. If the value is not modified, the system deletes any additional local pool copies when the application that created the copy is closed.

Note: The property is saved within the project once the script is executed. To change the property, modify and rerun the script.

Syntax: `NumberKeepLocalCopies As Long`

6.1.3.35.1 Sample Code

The following sample code sets the NumberKeepLocalCopies property to 10.

```
Dim theDocClass as SCBCdrDocClass
 Dim theSupplierSettings as CDRADSLib.SCBCdrSupExSettings
 Dim theAnalysisSettings as ISCBCdrAnalysisSettings
 Dim theObject as Object
 Set theDocClass=Project.AllClasses.ItemByName("Invoices")
 theDocClass.GetFieldAnalysisSettings
"VendorASSA",Project.DefaultLanguage,theAnalysisSettings
 Set theObject = theAnalysisSettings
 Set theSupplierSettings = theObject
 theSupplierSettings.NumberKeepLocalCopies = 10
```

6.1.3.36 ODBCName

This property sets or returns the name of the ODBC source.

Syntax: `ODBCName As String`

6.1.3.37 Password

This property sets or returns the password of the ODBC source.

Syntax: `Password As String`

6.1.3.38 Phrase

This property sets or returns the phrase by its index.

Syntax: `Phrase(PhraseIndex As Long) As String`

Parameter	Description
`PhraseIndex`	Zero-based index of the phrase.

6.1.3.39 PhrasesCount

This read-only property returns the number of phrases used for address analysis.

Syntax: `PhrasesCount As Long`

6.1.3.40 PoolName

This property sets or returns the name of the associative search pool.

Syntax: `PoolName As String`

6.1.3.41 PoolPath

This property sets or returns the file path of the associative search pool.

Syntax: `PoolPath As String`

6.1.3.42 PoolPathRelative

This property sets or returns if the pool should be saved relative to the path of the project.

Syntax: `PoolPathRelative As Boolean`

6.1.3.43 ProjectPath

This read-only property returns the path of the project file.

Syntax: `ProjectPath As String`

6.1.3.44 RemovePhrase

This method removes a phrase from a list of phrases for address analysis, specified by its index.

Syntax: `RemovePhrase(PhraseIndex As Long)`

Parameter	Description
`PhraseIndex`	Zero-based index of the phrase to be removed.

6.1.3.45 SavePoolInternal

This property sets or returns if a pool should be saved within the project file or as separate files.

Syntax: `SavePoolInternal As Boolean`

6.1.3.46 SearchAreaActive

This property sets or returns if the corresponding search area is active or not.

Syntax: `SearchAreaActive(SearchAreaIndex As Long) As Boolean`

Parameter	Description
`SearchAreaIndex`	Index of the area. Accepts values from **0** to **5** with following meaning: 0. First page header. 1. First page footer. 2. Subsequent pages header. 3. Subsequent pages footer.

4. Last page header.
5. Last page footer.

6.1.3.47 Separator

This property sets or returns a separator, either a semicolon or comma, that is used for CSV files.

Syntax: `Separator As String`

6.1.3.48 SetSearchArea

This method sets the area on the document in which to search.

Syntax: `SetSearchArea(SearchAreaIndex As Long, Left As Long, Top As Long, Width As Long, Height As Long)`

Parameter	Description
`SearchAreaIndex`	Index of the area. Accepts values from **0** to **5** with following meaning: 0. First page header. 1. First page footer. 2. Subsequent pages header. 3. Subsequent pages footer. 4. Last page header. 5. Last page footer.
`Left`	Distance as a percentage from the left border of the document.
`Top`	Distance as a percentage from the top of the document.
`Width`	Width as a percentage of the search area.
`Height`	Height as a percentage of the search area.

6.1.3.49 SQLQuery

This property sets or returns an SQL statement used to import ODBC source.

Syntax: `SQLQuery As String`

6.1.3.50 StartAddEntry

This method prepares the insertion of a new entry to the associative search pool.

Syntax: `StartAddEntry()`

6.1.3.51 StartUpdate

This method generates and opens a new empty pool, or opens an existing pool for the update.

Syntax: `StartUpdate(RemoveExistingPool As Boolean)`

Parameter	Description
`RemoveExistingPool`	When this Boolean variable is set to **True**, the old pool is removed, otherwise the existing pool is updated by further *AddPhrase* calls. In this case, it should not be required to call the *AddColumn* method, because the former column information has to be taken. Moreover, in case this parameter is **True**, and the ***AddColumn*** method is invoked, that method will report an error because it must be prohibited to modify the existing column.

6.1.3.52 UserName

This property sets or returns the user name required to login in to the ODBC source.

Syntax: `Username As String`

6.1.3.53 VendorTypeColumn

This property sets or returns the column that defines the vendor type. The vendor type column must contain a value from **0** to **2** as follows:

0. No class is created for this vendor through Supervised Learning Workflow.
1. Allows one document for that vendor to be trained.
2. Allows unlimited training.

Syntax: `VendorTypeColumn As String`

7 SCBCdrParaCheckLib

7.1 SCBCdrParaCheckAnalysisSettings

7.1.1 Description

This class contains the functions for the Check Analysis engine.

7.1.2 Type Definitions

7.1.2.1 CROParaCheckPayeeRecognitionMode

This data type determines the recognition mode.

Available Types	Description
`CdrAUTFile`	Automatic import from file for associative search field.
`CROParaCheckPayeeRecognitionModeAcc`	This is the default mode. In this mode, the names in the vocabulary entries reflect the entire payee name information.
`CROParaCheckPayeeRecognitionModeKeyWordSearch`	This mode restricts the search to keywords. This enhances the search performance. In this mode, the name entries in the vocabulary entries list contain one single word or a part of the Payee Name.

7.1.3 Methods and Properties

7.1.3.1 AddItemToVocabulary

This method adds a vocabulary entry to the list of possible items.

Syntax: `AddItemToVocabulary (Name As string, Weight As long, caption As CroParaCheckVocabularyEntriesAddOptions, IsVisible As Boolean)`

Parameter	Description
`Name`	Payee Name - Provide the name of the possible payee candidates. Keyword - Provide a keyword for possible payee candidates. This can be a single word or part of the payee name.
`Weight`	Set a value between 0 and 15. Usually, most words have a weight of zero. A weight value of 15 corresponds to a vocabulary entry that appears in approximately 50 % of the images.
`caOption`	CroParaCheckVocabularyEntriesAddStatic: add entry to the static vocabulary CroParaCheckVocabularyEntriesAddDynamic: add entry to the dynamic vocabulary
`IsVisible`	Defines if the entry is visible in the GUI table in Designer application.

7.1.3.1.1 Sample Code

```
Dim theFieldCheckAnalysisSettings as Object
 Set theFieldCheckAnalysisSettings = FieldAnalysissettings
 Dim ParaCheckSettings as SCBCdrParaCheckAnalysisSettings
 Set ParaCheckSettings = theFieldCheckAnalysisSettings
 ParaCheckSettings. AddItemToVocabulary "PayeeName1", 0,
CroParaCheckVocabularyEntriesAddDynamic,True
 ParaCheckSettings. AddItemToVocabulary "PayeeName2", 15,
CroParaCheckVocabularyEntriesAddDynamic, True
 ParaCheckSettings. AddItemToVocabulary "PayeeName3", 0,
CroParaCheckVocabularyEntriesAddDynamic, False
 ParaCheckSettings. AddItemToVocabulary "PayeeName4", 5,
CroParaCheckVocabularyEntriesAddDynamic, True
```

7.1.3.2 ClearVocabulary

Use this method to remove vocabulary entries.

Syntax: `ClearVocabulary (clOption as CroParaCheckVocabularyEntriesClearOptions)`

Parameter	Description
`clOption`	- CroParaCheckVocabularyEntriesClearStaticOnly: Remove all static entries - CroParaCheckVocabularyEntriesClearDynamicOnly: Remove all dynamic entries - CroParaCheckVocabularyClearEntriesAll: Remove all dynamic and static entries

7.1.3.2.1 Sample Code

```
Dim theFieldCheckAnalysisSettings as Object
 Set theFieldCheckAnalysisSettings = FieldAnalysissettings
 Dim ParaCheckSettings as SCBCdrParaCheckAnalysisSettings
 Set ParaCheckSettings = theFieldCheckAnalysisSettings
 ParaCheckSettings.ClearVocabulary CroParaCheckVocabularyEntriesClearDynamicOnly
```

7.1.3.3 FieldType

This property sets or returns the field type.

Syntax: `FieldType As Long`

7.1.3.4 MinDistance

This property sets or returns the minimum distance.

Syntax: `MinDistance As double`

7.1.3.5 MinWeight

This property sets or returns the minimum weight.

Syntax: `MinWeight As double`

7.1.3.6 PayeeLineRecMode

This property sets or returns the engine recognition mode.

Syntax: `PayeeLineRecMode As CROParaCheckPayeeRecognitionMode`

Mode	Description
`CROParaCheckPayeeRecognitionModeAcc`	This is the default mode. In this mode the names in the vocabulary entries reflect the entire Payee Name information
`CROParaCheckPayeeRecognitionModeKeyWordSearch`	This mode restricts the search to keywords. This enhances the search performance. In this mode, the name entries in the vocabulary entries list contain one single word or a part of the Payee Name

7.1.3.7 PayeeVocCoverage

Use this property to define the vocabulary coverage parameter for the payee line. This value expresses the occurrence likelihood of the predefined names on the checks.

Syntax: `PayeeVocCoverage As long`

Parameter	Description
`Long`	Define the vocabulary coverage as a LONG value in the range between 1 and 100. The default value is 35

7.1.3.7.1 Sample Code

```
Private Sub Document_PreExtract(pWorkdoc as SCBCdrPROJLib.ISCBCdrWorkdoc)
    Set ParaCheckSettings =
Project.AllClasses(pWorkdoc.DocClassName).Fields("Payee").AnalysisSetting(Projec
t.DefaultLanguage)
    ParaCheckSettings.PayeeVocCoverage = 85
End Sub
```

7.1.3.8 PayeeVocEntries

This property returns the collection of vocabulary entries objects.

Syntax: `PayeeVocEntries As ISCBCroPayeeVocEntries`

7.2 PayeeVocEntries

7.2.1 Description

This is a collection of all vocabulary entry objects contained in the current ParaCheckSettings object.

7.2.2 Methods and Properties

7.2.2.1 Count

This property returns the number of items within the vocabulary entry collection.

Syntax: `Count As Long`

7.2.2.2 Item

This read-only property returns a specified item from the collection.

Syntax: `Item (Index As Variant) As ISCBCroPayeeVocEntry`

Parameter	Description
`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.

7.2.2.3 ItemByIndex

This property returns an item from the collection specified by the index.

Syntax: `ItemByIndex (Index As Long) As ISCBCroPayeeVocEntry`

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

7.2.2.4 ItemByName

This property returns the specified item from the collection.

Syntax: `ItemByName (Name As String) As ISCBCroPayeeVocEntry`

Parameter	Description
`Name`	Name of the item to retrieve from the collection

7.2.2.5 ItemExists

This method returns TRUE if an item with the specified name exists inside the collection, or FALSE if not.

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

Parameter	Description
`Name`	Name of item for which to search

7.2.2.6 ItemIndex

Returns the index of an item specified by name.

Syntax: `ItemIndex (Name As String) As Long`

Parameter	Description
`Name`	Name specifying an item in the collection

7.2.2.7 ItemName

Returns the name of an item specified by index.

Syntax: `ItemName (Index As Long) As String`

Parameter	Description
`Index`	Index specifying an item in the collection, valid range from 1 to Count.

7.2.2.8 MoveItem

This method moves an item specified by OldIndex from OldIndex to NewIndex.

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

Parameter	Description
`OldIndex`	Index of item to move valid range from 1 to Count.
`NewIndex`	New index of the item after the move has occurred, valid range from 1 to Count.

7.2.2.9 Remove

This method removes the specified item from the collection and releases the reference count to this item.

Syntax: `Remove (ItemName As String)`

Parameter	Description
`ItemName`	Name of item to remove.

7.2.2.10 Tag

This property stores a variant for each item of the collection.

Syntax: `Tag (Index As Long) As Variant`

Parameter	Description

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

8 SCBCdrFormatEngine

This class provides methods and properties for the Format Analysis engine.

8.1 Sample Code

The following sample code searches for a 1 to 10 digits long number in the workdoc by using the Simple Expression algorithm. If the word is found, the variable `pValid` is set to true.

```
Private Sub PoNo_Validate(pField as SCBCdrPROJLib.ISCBCdrField, pWorkdoc as
SCBCdrPROJLib.ISCBCdrWorkdoc, pValid as Boolean)
    Dim FormatEngine as SCBCdrFormatEngine
    Dim oPOTemplate as SCBCdrFormatSettings
    If FormatEngine Is Nothing Then Set FormatEngine = New SCBCdrFormatEngine
    If oPOTemplate Is Nothing Then Set oPOTemplate = New SCBCdrFormatSettings
    oPOTemplate.DeleteAll
    oPOTemplate.AddFormat("#[1-10]")
    oPOTemplate.MaxDistance = 0.1
    oPOTemplate.MaxWordCount = 5
    oPOTemplate.MaxWordGap = 4
    oPOTemplate.MaxWordLen = 100
    oPOTemplate.KeepSpaces = False
    oPOTemplate.CaseSensitive = False
    oPOTemplate.IgnoreCharacters(0) = ",.-/()[]"
    oPOTemplate.CompareType(0) = CdrTypeSimpleExpression
    Dim strFoundString as String
    If FormatEngine.FindStringFirst(pWorkdoc, oPOTemplate, , strFoundString)
Then
       strFoundString = "Found"
       pField.Text = strFoundString
       pValid = True
    Else
       pValid = False
    End If
End Sub
```

8.2 Sample Code

The following sample code writes the confidence of the test string **1234567890** against the Format String #[3-5], which is the index 3 of the field *MyField*, into the log file.

```
Private Sub MyField_PreExtract(pField as SCBCdrPROJLib.ISCBCdrField, pWorkdoc as
SCBCdrPROJLib.ISCBCdrWorkdoc)
    Dim DocClass as SCBCdrDocClass
    Dim theAnalysisSettings as SCBCdrPROJLib.ISCBCdrAnalysisSettings
    Dim theSettings as Object
    Dim theFormatSettings as SCBCdrFormLib.SCBCdrFormatSettings
    Dim AE as Object
    Dim theFormatEngine as SCBCdrFormLib.SCBCdrFormatEngine
    Dim TestStr as String
    Dim fDist as Single
    'Get current DocClass
    Set DocClass=Project.AllClasses.ItemByName(pWorkdoc.DocClassName)
    'Get the settings for the field 'MyField'
    DocClass.GetFieldAnalysisSettings("MyField","German", theAnalysisSettings)
    'Convert them to the SCBCdrFormatSettings
    Set theSettings = theAnalysisSettings
    Set theFormatSettings = theSettings
    'Get SCBCdrFormatEngine from the project
    Set AE=Project.GetAnalysisEngineByName("Format Analysis Engine")
```

```
    Set theFormatEngine = AE
    'Test a string against Format String index 2 of field MyField, using the
current options for that specific search string
    TestStr = "1234567890"
    'The Format String index 2 is #[3-5] (Simple Expression)
    fDist = theFormatEngine.TestString(TestStr, 2, theFormatSettings)
    Project.LogScriptMessageEx(CDRTypeInfo, CDRSeverityLogFileOnly, "Distance
from the String 1234567890 using #[3-5] is: " & CStr(fDist)) 'Expected value is
0.5
End Sub
```

8.3 Sample Code

The following sample code shows how to set **Compare case sensitive** and **Keep spaces between connected words** options for a specific format string of the field *MyField,* and how to check if an option is active.

```
Private Sub MyField_PreExtract(pField as SCBCdrPROJLib.ISCBCdrField, pWorkdoc as
SCBCdrPROJLib.ISCBCdrWorkdoc)
    Dim DocClass as SCBCdrDocClass
    Dim theAnalysisSettings as SCBCdrPROJLib.ISCBCdrAnalysisSettings
    Dim theSettings as Object
    Dim theFormatSettings as SCBCdrFormLib.SCBCdrFormatSettings
    Dim nFlag3 as Long
    'Get current DocClass
    Set DocClass=Project.AllClasses.ItemByName(pWorkdoc.DocClassName)
    'Get the settings for the field 'MyField'
    DocClass.GetFieldAnalysisSettings ("MyField","German", theAnalysisSettings)
    'Convert them to the SCBCdrFormatSettings
    Set theSettings = theAnalysisSettings
    Set theFormatSettings = theSettings
    'Set the "Case Sensitive" option (bit 1) for Format string 3
    'Get the current settings for the Format String
    nFlag3 = theFormatSettings.SrchFlag(3)
    'Set bit 1. To clear the option, use "nFlag3 And (Not 1)"
    theFormatSettings.SrchFlag(3) = nFlag3 Or 1
    'Set the "Keep spaces..." option (bit 2) for Format string 3
    'Get the current settings for the Format String
    nFlag3 = theFormatSettings.SrchFlag(3)
    'Set bit 2. To clear the option, use "nFlag3 And (Not 2)"
    theFormatSettings.SrchFlag(3) = nFlag3 Or 2
    'If "Keep spaces..." (bit 2) is enabled for Format String 3
    'write a message in the Log
    If (theFormatSettings.SrchFlag(3) And 2) Then
       Project.LogScriptMessageEx (CDRTypeInfo, CDRSeverityLogFileOnly, "Keep
Spaces option is active for Format String 3")
    End If
End Sub
```

8.4 Methods and Properties

8.4.1 FindStringFirst

This method finds the first word in the workdoc according to the `pSettings` parameter. The search starts from the first word in the workdoc and the first FormatSetting search pattern.

Note: The method does not take the region settings into account.

Syntax: `FindStringFirst(pWorkdoc As ISCBCdrWorkdoc, pSettings As SCBCdrFormatSettings, pFormatIndex As Long, pFoundString As String, pWordID As Long) As Boolean`

Parameter	Description
`pWorkdoc`	The current workdoc.
`pSettings`	The settings of the specific format string.
`pFormatIndex`	Output parameter that contains the index of the matched format setting if ***pMatched*** is true. contains the index of the format string that produced a successful match Optional parameter. The default value is 0.
`pFoundString`	Output parameter that contains the found string. Optional parameter. The default value is "0".
`pWordID`	Output parameter that contains the WordID of the best matching word in the workdoc. Optional parameter. The default value is 0.

8.4.2 FindStringNext

This method finds the next word in the workdoc according to the pSettings parameter.

The method continues to search from the last word or format setting index found by the previous *FindStringFirst* or *FindStringNext* call.

Note: The method does not take the region settings into account.

Syntax: `FindStringNext(pWorkdoc As ISCBCdrWorkdoc, pSettings As SCBCdrFormatSettings, pFormatIndex As Long, pFoundString As String, pWordID As Long) As Boolean`

Parameter	Description
`pWorkdoc`	The current workdoc.
`pSettings`	The settings of the specific format string.
`pFormatIndex`	Output parameter that contains the index of the matched format setting if pMatched is true. Optional parameter. The default value is 0.
`pFoundString`	Output parameter that contains the found string. Optional parameter. The default value is "0".
`pWordID`	Output parameter that contains the index of the best matching word in the workdoc. Optional parameter. The default value is 0.

8.4.3 GetBlockID

This method retrieves the workdoc block index for the current matched word. This method is called typically after executing *FindStringFirst* or *FindStringNext* for search operations where the entire block is needed, such as in an address block search algorithm.

Syntax: `GetBlockID(plBlockID As Long)`

Parameter	Description
`plBlockID`	Output parameter. ▪ Contains a valuable result if the **`AnalysisMethod`** property is set to **`CdrAnalysisString`**. ▪ Contains 0 (zero) if the *AnalysisMethod* property is set to `CdrAnalysisDesignator`.

8.4.4 SrchFlag

Bit flag property to set or return the string construction rules for a single Format String.

- Bit 1 contains the "Compare case sensitive" option. Use this option to make the candidate search case-sensitive.
- Bit 2 contains the "Keep spaces between connected words" option. Use this option to keep the spaces in the format string.
- Bit 3 contains the trigram method. If set to *CdrTypeTrigram*, the new trigram method added in version 12.2.1.3.0 is used, otherwise the trigram method of the version 11.1.1.9.0 and before is used.

Use the bit-wise operators **Or** and And **Not** to set or clear the options.

Note: The options "Compare case sensitive" and "Keep spaces between connected words" on the General tab of the "Format Analysis Engine" settings in Designer refer to all Format Strings.

See *Rules for string construction from words* in the *WebCenter Forms Recognition Designer Guide*.

Syntax: `SrchFlag (index As Long) As Long`

Parameter	Description
`Index`	The index parameter has a valid range from 0 to FormatCount – 1.

8.4.5 TestString

Use this method to test a particular search string (Format String) against an arbitrary text, using the settings assigned to that specific Format String.

The method returns the distance value. The distance calculates as follows: (StrLength-MatchedSubstrLength)/StrLength.

- 0.0 means that the exact search string was found.
- 1.0 means that not even a partial match was found.

Note: The considered MatchedSubstrLength is the one with the maximum length in the Format String expression.

Syntax:
```
TestString(ByVal bstrText As String, ByVal nFormatIndex As Long,
pFormatSettings As SCBCdrFormatSettings) As Single
```

Parameter	Description
`bstrText`	The string to be tested against the Format String.
`nFormatIndex`	Zero-based index of the Format String list.
`pFormatSettings`	The settings of the specific Format String.

9 SCBCdrFormatSettings

This class provides methods and properties to specify the format settings.

9.1 Sample Code

The following sample code searches for the word *Invoice* or a very similar/misspelled word in the workdoc by using the Levenshtein algorithm. If the word is found, the variable `InvoiceFound` is set to `true`.

```
Dim FormatSettings As SCBCdrFormatSettings
 Dim FormatEngine as SCBCdrFormatEngine
 Dim strStringFound as string
 Dim lngWordID as long
 Dim InvoiceFound as Boolean

 Set FormatSettings = New SCBCdrFormatSettings
 FormatSettings.AddFormat("Invoice")
 FormatSettings.CompareType(0) = CdrTypeLevenShtein
 FormatSettings.AnalysisMethod(0) = SCBCdrFormLib.CdrAnalysisString
 FormatSettings.MaxWordCount = 4
 FormatSettings.MaxWordGap= 5.00
 FormatSettings.MaxDistance = 0.35
 FormatSettings.CaseSensitive = False
 FormatSettings.MaxWordLen = 150
 If FormatEngine.FindStringFirst(pWorkdoc,
FormatSettings,,strStringFound,lngWordID) Then
    InvoiceFound = true  // The word "Invoice" was found on the document
 else
    InvoiceFound = false // The word "Invoice" was not found on the document
 End if
```

9.2 Type Definitions

9.2.1 CdrAnalysisMethod

This type defines the analysis method.

Available Types	Description
`CdrAnalysisString`	The result contains the matched string.
`CdrAnalysisDesignator`	The result contains the information positioned to the matched string, where the position is defined by DesignatorType.

9.2.2 CdrDesignatorType

This type defines the designator type.

Available Types	Description
`MaxWordGap`	Corresponds to the “Max. Compare Distance” property

`CdrDesignatorEndOfLine`	The compare operation returns all words between the search string and the end of the line.
`CdrDesignatorNextBlock`	The compare operation returns the block below the search string.
`CdrDesignatorNextLine`	The compare operation returns the line below the search string.
`CdrDesignatorNextParagraph`	The compare operation returns the paragraph below the search string.
`CdrDesignatorNextWord`	The compare operation returns the word right to the search string.
`CdrDesignatorPrevWord`	The compare operation returns the word left to the search string.
`CdrDesignatorThisBlock`	The compare operation returns the block that contains the search string.
`CdrDesignatorWordAbove`	The compare operation returns the word above the search string.
`CdrDesignatorWordBelow`	The compare operation returns the word below the search string.

9.3 Methods and Properties

9.3.1 AddFormat

This method adds the format string to the list of format strings.

Syntax: `AddFormat(newVal As String)`

Parameter	Description
`newVal`	Format string to add to the list of format strings.

9.3.2 AnalysisMethod

This property sets or returns the analysis method for the specified format string from the list of format strings.

Syntax: `AnalysisMethod(Index As Long) As CdrAnalysisMethod`

Parameter	Description
`Index`	Index of the format string.

9.3.2.1.1 Sample Code

```
FormatSettings.AnalysisMethod(0) = SCBCdrFormLib.CdrAnalysisString
```

9.3.3 BottomFirst

This property sets or returns the search ending position from the bottom of the documents's first page as a percentage value between 0 and 100.

Note: This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `BottomFirst As Long`

9.3.4 BottomLast

This property sets or returns the search ending position from the bottom of the documents's last page as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `BottomLast As Long`

9.3.5 BottomSubseq

This property sets or returns the search ending position from the bottom of the documents's subsequent pages as a percentage value between 0 and 100.

Note: This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `BottomSubseq As Long`

9.3.6 CaseSensitive

This property sets or returns if the search operation works case-sensitive.

Syntax: `CaseSensitive As Boolean`

9.3.7 CompareType

This property sets or returns the string compare algorithm which is used for the specified format string from the list of format strings.

Syntax: `CompareType(Index As Long) As CdrCompareType`

Parameter	Description
`Index`	Index of the format string.

9.3.8 DesignatorType

This property sets or returns the DesignatorType for the specified format string from the list of format strings.

Syntax: `DesignatorType(Index As Long) As CdrDesignatorType`

9.3.9 DeleteAll

This method removes all format strings from the list of format strings.

Syntax: `DeleteAll()`

9.3.10 DeleteFormat

This method deletes the specified format string from the list of format strings.

Syntax: `DeleteFormat(Index As Long)`

Parameter	Description
`Index`	Index of the format string to delete.

9.3.11 FormatCount

This read-only property returns the number of format strings in the list of format strings.

Syntax: `FormatCount As Long`

9.3.12 FormatString

This property sets or returns the specified format string in/from the list of format strings.

Syntax: `FormatString (index As Long) As String`

Parameter	Description
`Index`	Index of the format string.

9.3.13 FormatValid

This read-only property returns the result of the validation check for the specified format string from the list of format strings.

Syntax: `FormatValid(Index As Long)as Boolean`

Parameter	Description
`Index`	Index of the format string.

9.3.14 IgnoreCharacters

This property sets or returns the ignore character string for the specified format string from the list of format strings. The ignore character string contains the characters or symbols which are ignored in the search algorithm.

Syntax: `IgnoreCharacters(Index As Long) As String`

Parameter	Description
`Index`	Index of the format string.

9.3.15 KeepSpaces

This property sets or returns if the engine keeps the spaces between the connected words.

Syntax: `KeepSpaces As Boolean`

9.3.16 LeftFirst

This property sets or returns the search starting position from the left of the documents's first page as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `LeftFirst As Long`

9.3.17 LeftLast

This property sets or returns the search starting position from the left of the documents's last page as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `LeftLast As Long`

9.3.18 LeftSubseq

This property sets or returns the search starting position from the left of the documents's subsequent pages as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `LeftSubseq As Long`

9.3.19 MaxDistance

This property sets or returns the maximal compare distance. A match requires that the actual compare distance is less or equal to the maximum compare distance.

Syntax: `MaxDistance As Double`

9.3.20 MaxWordCount

This property sets or returns the maximum number of words combined as input for the search operation. Words which are combined must be in the same line.

Syntax: `MaxWordCount As Long`

9.3.21 MaxWordGap

This property sets or returns the maximum distance in millimeters that permits word concatenation during the search. The requirements strongly depend on the font size.

Syntax: `MaxWordGap As Double`

9.3.22 MaxWordLen

This property sets or returns the maximum overall word length in millimeters on the document of the combined input string for the search engine.

Syntax: `MaxWordLen As Double`

9.3.23 MoveFormat

This method moves a format string to a new position in the list of format strings.

Syntax: `MoveFormat(OldIndex As Long, NewIndex As Long)`

Parameter	Description
`OldIndex`	Current index of the format string.
`NewIndex`	New index of the format string.

9.3.24 Prefix

This property sets or returns the prefix string for the specified format string from the list of format strings. The prefix string contains the characters or symbols which are ignored when found at the beginning of the word.

Syntax: `Prefix(Index As Long) As String`

Parameter	Description
`Index`	Index of the format string.

9.3.25 ResetTranslationLanguage

This method sets the transliteration approach to CDRTranslationLanguageDefault, which means that no transliteration executes for the format string, ignore char string, prefix and suffix string. This is the default setting and should be used in case the workdoc was not transliterated. This method has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `ResetTranslationLanguage()`

9.3.26 RightFirst

This property sets or returns the search starting position from the right of the documents's first page as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `RightFirst As Long`

9.3.27 RightLast

This property sets or returns the search starting position from the right of the documents's last page as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `RightLast As Long`

9.3.28 RightSubseq

This property sets or returns the search starting position from the right of the documents's subsequent pages as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `RightSubseq As Long`

9.3.29 SettingsCheckSum

This property sets or returns the checksum string created during extraction for the format settings object for the field. The checksum is not stored in the storage object. Checksum is used in WebCenter Forms Recognition internally. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `SettingsCheckSum As String`

9.3.30 SetTranslationLanguage

This method specifies the transliteration approach for the format analysis engine. In general, use the same approach as used for the workdoc's OCR text. This method has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `SetTranslationLanguage(Language As CDRTranslationLanguage)`

Parameter	Description
`Language`	Transliteration approach

9.3.31 Suffix

This property sets or returns the suffix string for the specified format string from the list of format strings. The suffix string contains the characters or symbols which are ignored when found at the end of the word.

Syntax: `Suffix(Index As Long) As String`

Parameter	Description
`Index`	Index of the format string.

9.3.32 UseFirstPage

This property sets or returns if the engine searches on first page of the document. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `UseFirstPage As Boolean`

9.3.32.1 UseLastPage

This property sets or returns if the engine searches on the last page of the document. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `UseLastPage As Boolean`

9.3.33 UseSubseqPage

This property sets or returns if the engine searches on subsequent pages of the document. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `UseSubseqPage As Boolean`

9.3.34 TopFirstPage

This property sets or returns the search starting position from the top of the documents's first page as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `TopFirstPage As Boolean`

9.3.35 TopLastPage

This property sets or returns the search starting position from the top of the documents's last page as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `TopLastPage As Boolean`

9.3.36 TopSubseqPage

This property sets or returns the search starting position from the top of the documents's subsequent pages as a percentage value between 0 and 100. This property has no impact to the methods *FindStringFirst* and *FindStringNext*.

Syntax: `TopSubseqPage As Boolean`

10 SCBCdrBATCHLib Object Reference

10.1 SCBCdrBATCHLib

10.1.1 Description

This library provides classes and types to work with batches.

10.1.2 SCBCdrBatchRoot Methods and Properties

10.1.2.1 ActivateSupport

This property sets or returns if database support for the batch connection is enabled.

Syntax: `ActivateSupport As Boolean`

10.1.2.1.1 Sample Code

```
Dim pBatchRoot as New SCBCdrBATCHLib.SCBCdrBatchRoot
 pBatchRoot.ActivateSupport = True
```

10.1.2.2 BatchCount

This read-only property returns the number of batches according to the filter conditions applied by the `SetFilter` method.

Syntax: `BatchCount As Long`

10.1.2.2.1 Sample Code

```
Dim pBatchRoot as New SCBCdrBATCHLib.SCBCdrBatchRoot
 …
 pBatchRoot.Connect("Job name", "", "LOGIN_AS_CURRENT", "", "Module type name")
```

10.1.2.3 BatchId

This read-only property returns the batch id as a string. The index of the batch is influenced by the filter conditions.

Syntax: `BatchID(BatchIndex As Long)`

Parameter	Description
`BatchIndex`	Index of the batch.

10.1.2.3.1 Sample Code

```
Dim pBatchRoot as New SCBCdrBATCHLib.SCBCdrBatchRoot
 …
 pBatchRoot.Connect("Job name", "", "LOGIN_AS_CURRENT", "", "Module type name")
```

10.1.2.4 Connect

This method connects to the batch root.

Syntax: `Connect(BatchRootPath As String, ImageRootPath As String, UserName As String Password As String, ModuleType As String)`

Parameter	Description
`BatchRootPath`	Batch Root Path. If `ActivateSupport` is `True`, the parameter can be an empty string.
`ImageRootPath`	Image Root Path. If `ActivateSupport` is `True`, the parameter can be an empty string.
`UserName`	The user that can connect to the database job. In most cases it is applicable to use the special name `LOGIN_AS_CURRENT` in order to reuse the current connection to the database.
`Password`	Password. If `UserName = LOGIN_AS_CURRENT`, the password can be an empty string.
`ModuleType`	Type of the module connecting

10.1.2.4.1 Sample Code

```
Dim pBatchRoot as New SCBCdrBATCHLib.SCBCdrBatchRoot
 …
 pBatchRoot.Connect("Job name", "", "LOGIN_AS_CURRENT", "", "Module type name")
```

10.1.2.5 Disconnect

This method disconnects the batch from the batch root.

Syntax: `Disconnect()`

10.1.2.6 SetConnectionProperties

Use this method to specify the used job and the instance name.

Syntax: `SetConnectionProperties(SelectedJobName As String, InstanceName As String, CreateJobIfNotExist As Boolean)`

Parameter	Description
`SelectedJobName`	Job name from the JOBS table.
`InstanceName`	RTS instance used to connect.
`CreateJobIfNotExist`	True: Create the job if not present. False: Use an existing job only.

10.1.2.6.1 Sample Code

```
Dim pBatchRoot as New SCBCdrBATCHLib.SCBCdrBatchRoot
 …
 theBatchRoot.SetConnectionProperties "My DB Job", "MyRTS", False
```

10.1.2.7 SetFilter

This method sets the filter for the batch root. By default, all batches from the job are available at the batch root level. This method allows you to select only batches of a specific state.

Syntax: `SetFilter(State As Long)`

Parameter	Description
`State`	State of the batch you want to access in the batch root level.

10.1.2.8 SetLoginProperties

This method sets the login credentials for the project database user.

Syntax: `SetLoginProperties(UserName As String, Password As String, ModuleType As String, InstanceName As String)`

Parameter	Description
`UserName`	User name
`Password`	Password
`ModuleType`	Type of the module connecting
`InstanceName`	RTS instance used to connect

10.1.2.8.1 Sample Code

```
Dim pBatchRoot as New SCBCdrBATCHLib.SCBCdrBatchRoot
…theBatchRoot.SetLoginProperties "username", "password", "Server", "MyRTS"
```

11 StringComp Object Reference (SCBCdrSTRCOMPLib)

11.1 SCBCdrStringComp

11.1.1 Description

This component provides several implementations of string compare algorithms.

11.1.2 Type Definitions

11.1.2.1 CdrCompareType

This table contains a list of string compare algorithms.

Available Types	Description
`CdrTypeLevenShtein`	Levenshtein algorithm.
`CdrTypeRegularExpression`	Regular expression.
`CdrTypeSimpleExpression`	Simple expression.
`CdrTypeStringComp`	Exact string compare.
`CdrTypeTrigram`	Trigram algorithm.

11.1.3 Methods and Properties

11.1.3.1 CaseSensitive

This property controls if the compare algorithm should work case-sensitive.

Syntax: `CaseSensitive As Boolean`

11.1.3.2 CompTypes

This property selects the compare algorithm used for the next call of *Distance*.

Syntax: `CompType As CdrCompareType`

11.1.3.3 Distance

This method performs the selected string compare algorithm. You must first initialize the search expression and the compare method. 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)`

Parameter	Description
`String`	Specifies the string that should be compared with the search expression.
`Distance`	Returns the distance of the compare operation that is between **0.0** and **1.0**.

11.1.3.4 LevDeletions

This read-only property returns the count of deletions calculated by the last *Distance* function.

Syntax: `LevDeletions As Single`

11.1.3.5 LevInsertions

This read-only property returns the count of insertions calculated by the last *Distance* function.

Syntax: `LevInsertions As Single`

11.1.3.6 LevRejects

This read-only property returns the count of rejects calculated by the last *Distance* function.

Syntax: `LevRejects As Single`

11.1.3.7 LevReplacements

This read-only property returns the count of replacements calculated by the last *Distance* function.

Syntax: `LevReplacements As Single`

11.1.3.8 LevSame

This read-only property returns the count of equal characters calculated by the last *Distance* function.

Syntax: `LevSame As Single`

11.1.3.9 LevTraceMatrix

This read-only property returns the Levenshtein trace matrix calculated by the last *Distance* function.

Syntax: `LevTraceMatrix As String`

11.1.3.10 LevTraceResult

This read-only property returns the Levenshtein trace result calculated by the last *Distance* function.

Syntax: `LevTraceResult As String`

11.1.3.11 MatchEndPosition

This read-only property returns the matching end position calculated by the last *Distance* function.

Syntax: `MatchEndPosition As Single`

11.1.3.12 MatchStartPosition

This read-only property returns the matching start position calculated by the last *Distance* function.

Syntax: `MatchStartPosition As Single`

11.1.3.13 SearchExpression

This property contains the search expression that should be used for the next compare operation.

Syntax: `SearchExpression As String`

11.1.3.14 ValidateSearchExpression

This method performs a syntax check for the specified compare method and search expression.

Syntax: `ValidateSearchExpression(Type As CdrCompareType, SearchExpression As String) As Boolean`

Parameter	Description
`Type`	Compare method to use for validation.
`SearchExpression`	Search expression to validate.

12 DISTILLERVERIFIERCOMPLib Object Reference

12.1 DISTILLERVERIFIERCOMPLib

12.1.1 Description

The Verifier Component library *DISTILLERVERIFIERCOMPLib* provides methods and properties to work with verification forms and verification form elements.

12.1.2 Type Definitions

12.1.2.1 CdrVerifierFieldType

This type lists the Verifier field types. This type interface is a member of the Cedar Verifier Project library.

Available Types	Description
`CDRVerifierFieldTypeCheckbox`	Check box field type
`CDRVerifierFieldTypeCombobox`	Combo box field type
`CDRVerifierFieldTypeTableCheckBoxCell`	Table check box cell field type
`CDRVerifierFieldTypeTextMultiline`	Multiline Text field type
`CDRVerifierFieldTypeTextSingleline`	Single Line Text field type

12.2 SCBCdrVerificationForm

12.2.1 Description

This interface is used to set properties specific for the verification form object, as well as to set default properties for embedded elements, such as verification fields, labels, tables and buttons.

For the Web Verifier, use this method in the *VerifierFormLoad* event.

12.2.2 Properties

12.2.2.1 DefaultElementBackgroundColorInvalid

This property sets or returns the default color for all invalid (in terms of validation status) field elements available on this verification form.

Syntax: `DefaultElementBackgroundColorInvalid As OLE_COLOR`

12.2.2.2 DefaultElementBackgroundColorValid

This property sets or returns the default color for all valid (in terms of validation status) field elements available on this verification form.

Syntax: `DefaultElementBackgroundColorValid As OLE_COLOR`

12.2.2.3 DefaultFieldFont

This property sets or returns the default font for all verification field elements available on this verification form.

Syntax: `DefaultFieldFont As StdFont`

12.2.2.4 DefaultLabelFont

This property sets or returns the default font for all label elements available on this verification form.

Syntax: `DefaultLabelFont as OLE_COLOR`

12.2.2.5 DefaultLabelFontColor

This property sets or returns the default color for all label elements available on this verification form.

Syntax: `DefaultLabelFontColor As OLE_COLOR`

12.2.2.5.1 Sample Code

```
Dim clrDefaultColor as OLE_COLOR
 clrDefaultColor = -1 the
Form.VerificationLabels.ItemByIndex(lNextLabelIndex).FontColor = clrDefaultColor
```

12.2.2.6 DefaultLabelBackgroundColor

This property sets or returns the default background color for all label elements available on this verification form.

Syntax: `DefaultLabelBackgroundColor As OLE_COLOR`

12.2.2.7 FormBackgroundColor

This property sets or returns the background color for the form.

Syntax: `FormBackgroundColor As OLE_COLOR`

12.2.2.8 FormBackgroundColorDI

This property sets or returns the 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`

12.2.2.9 SetFieldFocus

This method sets the focus to the specified field or table cell and updates the `HighlightField`, `HighlightColumnIndex`, and `HighlightRowIndex` settings.

The method returns an error message if the specified field or table cell is hidden or does not exists on the verification form.

Do not use this method in either VerifierFormLoad or in any Validate field or table events. These events often execute in sequence and may affect the focus following each event, independent of SetFieldFocus.

Carefully use this method within the FocusChanged or CellFocusChanged events, As an endless loop may result.

Syntax: `SetFieldFocus(BSTR FormName, ISCBCdrWorkdoc pWorkdoc, BSTR bstrFieldName, BSTR bstrTableColumnName, long lTableRowIndex)`

Parameter	Description
`String`	Specifies the string that should be compared with the search expression.
`FormName`	Verification form name
`pWorkdoc`	The current workdoc
`bstrFieldName`	Field name
`bstrTableColumnName`	Column name
`lTableRowIndex`	Row index, the *SetFieldFocus* action is cancelled when the RowIndex value is "-1".

12.2.2.9.1 Sample Code

The following sample code sets the focus to the field *Table* in the first row of the column *Quantity*.

```
Private Sub Document_OnAction(pWorkdoc as SCBCdrPROJLib.ISCBCdrWorkdoc, ByVal
ActionName as String)
    If ActionName = "GoToField" Then
       Project.SetFieldFocus("Form_Invoices_1", pWorkdoc, "Table", "Quantity",
0)
    End If
End Sub
```

12.3 SCBCdrVerificationField

12.3.1 Description

This interface is used to identify verification properties specific for header fields' validation elements, like dropdown lists, checkboxes, and normal edit fields.

Note: To get the *OLE_COLOR* or *StdFont* object for the properties below, add **OLE Automation** as a reference.

12.3.2 Properties

12.3.2.1 AutoCompletionEnabled

This property enables or disables automatic completion for a verification field.

Syntax: `AutoCompletionEnabled As Boolean`

12.3.2.2 BackgroundColorInvalid

This property sets the color for the verification field to display to the user when the field requires manual verification.

When the field is invalid in Verifier, the color that is set displays to the user. By default, the invalid background color of the field is red.

Syntax: `BackgroundColorInvalid As OLE_COLOR`

12.3.2.3 BackgroundColorValid

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 displays to the user. By default, the valid background color of the field is green.

Syntax: `BackgroundColorValid As OLE_COLOR`

12.3.2.4 Font

This property sets the font for the content of the verification field.

Syntax: `Font As StdFont`

12.3.2.5 FontColor

This property sets the font color for the content of the verification field.

Syntax: `FontColor As OLE_COLOR`

12.3.2.6 Invisible

This property determines if the field is visible or hidden from the Verifier or Web Verifier form. The developer uses script options to hide or display the field from the verifier user. For the Web Verifier, this property can only be used in the *VerifierFormload* event.

Syntax: `Invisible As Boolean`

12.3.2.7 Left

This property provides the left position of the field on the Verifier form.

Syntax: `Left As Long`

12.3.2.8 Name

This property provides the name of the field on the Verifier form.

Syntax: `Name As String`

12.3.2.9 ReadOnly

This property determines if the verification field on the Verifier or Web Verifier form is editable or read-only. For the Web Verifier, use this method in the *VerifiedFormatLoad* event.

Set the property to **True** to make the field non-editable.

Syntax: `ReadOnly As Boolean`

12.3.2.10 TabIndex

This property allows the project developer 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 developer to change the sequence number to reorder the tab sequence of fields.

Syntax: `TabIndex As Long`

12.3.2.11 Top

This property provides the top position coordinates of the field on the Verifier form.

The project developer may choose to reorder positional information of the field if another element is being hidden. Using the *RepaintControls* method, the form UI is updated with the changes made.

Syntax: `Top As Long`

12.3.2.12 Type

This property provides the field type information of the field on the Verifier form. The developer may choose to review information based on the field type.

Syntax: `Type As CdrVerifierFieldType`

12.3.2.13 Width

This property provides the width size information of the field on the Verifier form.

The developer may choose to reorder or resize positional information of the field if another element is being hidden. Using the *RepaintControls* method, the form UI is updated with the changes made.

Syntax: `Width As Long`

12.4 SCBCdrVerificationTable

12.4.1 Description

This interface is used to identify verification properties specific for table validation elements.

12.4.2 Methods and Properties

12.4.2.1 FontFont

This property sets or returns the font settings for the individual table field element.

Syntax: `FontFont As StdFont`

12.4.2.2 BackgroundColorInvalid

This property sets or returns the 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`

12.4.2.3 BackgroundColorValid

This property sets or returns the 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`

12.4.2.4 HeaderBackgroundColor

This property sets or 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`

12.4.2.5 HeaderFont

This property sets or returns the font settings for all header buttons of the table field element, including row header buttons, column header buttons and the table header button.

Syntax: `HeaderFont As StdFont`

12.4.2.6 HeaderFontColor

This property sets or returns the font color for the header buttons of the table field element, including row header buttons and column header buttons.

Syntax: `HeaderFontColor As OLE_COLOR`

12.5 SCBCdrVerificationButton

12.5.1 Description

Use this interface to set verification properties specific for all custom buttons defined on a verification form.

12.5.2 Properties

12.5.2.1 Font

This property sets or returns the font settings, such as name, type and style, for the individual custom button control.

Syntax: `Font As StdFont`

12.5.2.2 FontColor

This property sets or returns the font color for the individual custom button control.

Syntax: `FontColor As OLE_COLOR`

12.5.2.3 BackgroundColor

This property sets or returns background color for the individual custom button control.

Syntax: `BackgroundColor As OLE_COLOR`

12.6 SCBCdrVerificationLabel

12.6.1 Description

This object is part of the Verifier Component Library. It enables the project developer to manipulate the verifier form labels.

The Verifier Component Library is not enabled by default. This component can be added to the script references for any project class.

12.6.2 Properties

12.6.2.1 BackgroundColor

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`

12.6.2.2 Font

This property sets the font for the content of the verification field label.

Syntax: `Font As StdFont`

12.6.2.3 FontColor

This property sets the font color for the content of the verification field label.

Syntax: `FontColor As OLE_COLOR`

12.6.2.4 Invisible

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.

Syntax: `Invisible As Boolean`

12.6.2.5 Left

This property provides the left position of the field on the Verifier form.

Syntax: `Left As Long`

12.6.2.6 Name

This property provides the name of the field label on the Verifier form.

Syntax: `Name As String`

12.6.2.7 Text

This property allows the project developer to set the text of the verification field label on the Verifier form.

Syntax: `Text As String`

12.6.2.8 Top

This property provides the top position coordinates of the field label on the Verifier form. The developer may choose to reorder positional information of the field label if another element is being hidden. Using the *RepaintControls* method, the form UI is updated with the changes made.

Syntax: `Top As Long`

12.6.2.9 Width

This property provides the width size information of the field label on the Verifier form.

Syntax: `Width As Long`

13 AP Packaged Project INI File Encryption

WebCenter Forms Recognition allows the user to encrypt a password (or any other value) within the AP Packaged Project's INI file. RSA encryption is used, which contains a public key and a private key.

The public key can be distributed to anybody that needs to encrypt strings and store them in the project INI file, for example, a WebCenter Forms Recognition administrator. Refer to *Section 8.2: Project INI File Encryption for the Integrator* below for information about how to encrypt a password using the public key.

Public key example:

```
<RSAKeyValue><Modulus>vJ+W7SuXuvOrWVoy4tPrbfLCuoHElo750cpTuEzLPk6iz6bHAodPVgLFaOEK+XMMS2G5
z+6961vuQsDGUt+O1Ag1PiTXCa6rrAaeCaaDO4HI8Mmpw0OkUZEfCZpTTYCYQPfZlgokwomF6VDSB9dlUS430IT0gc
tQY1b5iM4MqT0=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>
```

Only the project developer should know the private key. It will be coded into the project script to enable the decryption of the encrypted INI settings at runtime. Refer to *Section 8.1: Project INI File Encryption for the Project Developer* below for information about how to use the private key in project script to decrypt and use an encrypted password from the project INI file.

Private key example:

```
<RSAKeyValue><Modulus>vJ+W7SuXuvOrWVoy4tPrbfLCuoHElo750cpTuEzLPk6iz6bHAodPVgLFaOEK+XMMS2G5
z+6961vuQsDGUt+O1Ag1PiTXCa6rrAaeCaaDO4HI8Mmpw0OkUZEfCZpTTYCYQPfZlgokwomF6VDSB9dlUS430IT0gc
tQY1b5iM4MqT0=</Modulus><Exponent>AQAB</Exponent><P>8SRHEvT5Bn2paRHSDR9yCQb7WGYE9PbeHzuqwH
6iWa0LNYJrSrhhUeCEpwlPLQWQq10KmMZgG0+Br4nuBMmMHQ==</P><Q>yD7l9fjB/MJWYaV3LcEzY286Q+Xvo74i6
THvHkKqB1NKYGcN9xF9d8XbiUQNgBZ/4F02T6mFeYDO32KFVRXHoQ==</Q><DP>nRDTFn7nwRmSgfRwi8minkyk5DQ
3IFO35EIZ+x3Ao4Z52ZWkStwDz6/c12vR3XJVg7irkU0NBlzoDK1bklSw5Q==</DP><DQ>B3xieGmORva05/2ZkPpS
A3ubAALOjJ6FC5a0S7tOQ+vXMfdoTD45JIsfA+ipYIp2yVpyt1OtC7fHBA7Y0S95QQ==</DQ><InverseQ>4S1xqlX
K9f1rawGCbFWOVp6lz1fCoQ8RfyDE87/G/pUilHRJV2acBAcngY3c/MRMKrXQb8lx99k7dENUYc8ywQ==</Inverse
Q><D>KAL6cwkCQKgbuvKFRNSLZmFOqV2JpB5kI/p1U+0GWAs6Qi4wnPqy+53O3naOa2faPctXLSKJqvlvSz21VDMUC
syphvOSxBtc1cZHJp4ueQPA7u+qrIJaDY1RhlAVoqNfCJFX6+McVJ+I/X+mZOCtdUaCuAoNn014UYOaMujYDQE=</D
></RSAKeyValue>
```

Note: WebCenter Forms Recognition does not include tools for generating RSA encryption keys. The examples provided above can be used for demonstrating/testing, but it is the responsibility of the implementer to generate and provide the appropriate public and private keys in the required XML format.

13.1 Project INI File Encryption for the Project Developer

Where a database password is encrypted in the project INI file it is necessary to decrypt it at runtime, and then use the decrypted password in the database connection string to make the connection to the database. An example of how this would typically appear in the project INI file is:

```
SQL_VL_01_ConnectionString=Provider=OraOLEDB.Oracle.1; Persist Security Info=True;User
ID=WFR;Data Source=ORCL

SQL_VL_01_ConnectionPassword=puejB5SQNCFGgwe6MRoWc1G1y7qX8xSAhgUZjhN6JolhYdKIxla7vLMU4bYmG
9V3Ayxualp/ObgXRqnSAmGsGF1FPZXktRmf58SXbnCDXmYrYgp8eS3IaqiLUPrhTiRCvfr8ZsMtK+3usmahfxpESUO
Q7MZf36suWV4V3sBf9Xw=
```

Note: The **ConnectionString** setting does not contain the password. Instead, the database password is stored in its encrypted form in the **ConnectionPassword** setting.

The following script example shows how the encrypted password is retrieved from the INI file, decrypted, and then added to the connection string, resulting in a fully formed connection string that can be used to make a connection to the database through ADO.

Note: You must add a reference to the **CdrCrypt** COM object in the project script page.

```
Dim theCedarCryptographyHelper As New CdrCrypt.RSACodecInt

Dim strEncryptedPassword As String

Dim strOpenPassword As String

Dim strPrivateKey As String

strPrivateKey =
"<RSAKeyValue><Modulus>vJ+W7SuXuvOrWVoy4tPrbfLCuoHElo750cpTuEzLPk6iz6bHAodPVgLFaOEK+XMMS2G
5z+6961vuQsDGUt+O1Ag1PiTXCa6rrAaeCaaDO4HI8Mmpw0OkUZEfCZpTTYCYQPfZlgokwomF6VDSB9dlUS430IT0g
ctQY1b5iM4MqT0=</Modulus><Exponent>AQAB</Exponent><P>8SRHEvT5Bn2paRHSDR9yCQb7WGYE9PbeHzuqw
H6iWa0LNYJrSrhhUeCEpwlPLQWQq10KmMZgG0+Br4nuBMmMHQ==</P><Q>yD7l9fjB/MJWYaV3LcEzY286Q+Xvo74i
6THvHkKqB1NKYGcN9xF9d8XbiUQNgBZ/4F02T6mFeYDO32KFVRXHoQ==</Q><DP>nRDTFn7nwRmSgfRwi8minkyk5D
Q3IFO35EIZ+x3Ao4Z52ZWkStwDz6/c12vR3XJVg7irkU0NBlzoDK1bklSw5Q==</DP><DQ>B3xieGmORva05/2ZkPp
SA3ubAALOjJ6FC5a0S7tOQ+vXMfdoTD45JIsfA+ipYIp2yVpyt1OtC7fHBA7Y0S95QQ==</DQ><InverseQ>4S1xql
XK9f1rawGCbFWOVp6lz1fCoQ8RfyDE87/G/pUilHRJV2acBAcngY3c/MRMKrXQb8lx99k7dENUYc8ywQ==</Invers
eQ><D>KAL6cwkCQKgbuvKFRNSLZmFOqV2JpB5kI/p1U+0GWAs6Qi4wnPqy+53O3naOa2faPctXLSKJqvlvSz21VDMU
CsyphvOSxBtc1cZHJp4ueQPA7u+qrIJaDY1RhlAVoqNfCJFX6+McVJ+I/X+mZOCtdUaCuAoNn014UYOaMujYDQE=</
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
```

13.2 Project INI File Encryption for the Integrator

As an implementer or WebCenter Forms Recognition administrator, you simply need encrypt (for example) the database password and add the encrypted value to the project INI file. To encrypt a value, such as the database password:

1. Open the Windows Command Line
2. Navigate to the <*Installation Folder*>**\Bin\bin** directory in the Command Line
3. Execute the following command, replacing the <*myPassword*> and <*publicKey*> placeholders with the actual values for your environment:

   `DstCrypt.exe /text` <*myPassword*> `/key "`<*publicKey*>`"  >>` <*output text file name*>

   For example:

   ```
   DstCrypt.exe /text MyPassword /key
   "<RSAKeyValue><Modulus>vJ+W7SuXuvOrWVoy4tPrbfLCuoHElo750cpTuEzLPk6iz6bHAodPVgLFaOE
   K+XMMS2G5z+6961vuQsDGUt+O1Ag1PiTXCa6rrAaeCaaDO4HI8Mmpw0OkUZEfCZpTTYCYQPfZlgokwomF6
   VDSB9dlUS430IT0gctQY1b5iM4MqT0=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>"
   >> my_encrypted_password.txt
   ```

   Note: Notice that the public key value is contained in quotes but the password is not.

The text file specified in the command (e.g. my_encrypted_password.txt) will now contain the encrypted text string for the password.

4. Add the encrypted password to the **ConnectionPassword** setting in the project INI file. For example:

```
SQL_VL_01_ConnectionPassword=puejB5SQNCFGgwe6MRoWc1G1y7qX8xSAhgUZjhN6JolhYdKIxla7v
LMU4bYmG9V3Ayxualp/ObgXRqnSAmGsGF1FPZXktRmf58SXbnCDXmYrYgp8eS3IaqiLUPrhTiRCvfr8ZsM
tK+3usmahfxpESUOQ7MZf36suWV4V3sBf9Xw=
```

Note: Remember to remove the **`Password=<`*database password*`>;`** component from the setting in the corresponding **ConnectionString** setting.

---

14 Troubleshoot Scripting Issues

To provide script dumps for script issues, such as compilation problems that occur in Web Verifier, complete the following steps. This feature requires advanced product knowledge.

Note: If you enable this feature, encrypted script pages are not exported out of the project.

1. In Windows registry, complete one of the following substeps:
   - For a 32-bit machine, navigate to `[HKEY_LOCAL_MACHINE\SOFTWARE\Oracle\Cedar].`
   - For a 64-bit machine, navigate to `[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Oracle\Cedar].`
2. In the right pane, right-click and then click **New > DWORD (32-bit) Value**.
3. In the **Name** field, type **`DumpProjectScriptCode`** and then click **OK.**
4. Right-click the **DumpProjectScriptCode** key and click **Modify**.
5. In the **Edit DWORD (32-bit) Value** dialog box, in the **Value data** field, complete one of the following steps and then click **OK.**
   - To disable, type **0** and then click **OK**.
   - To enable, type **1** and then click **OK**.