Oracle® Fusion Middleware

Oracle WebCenter Forms Recognition Scripting User's Guide

11*g* Release 1 (11.1.1)

E50190-02

November 2015

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

Oracle Fusion Middleware Oracle WebCenter Forms Recognition Scripting User's Guide, 11*g* Release 1 (11.1.1)

E50190-02

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.

As another relevant extension, the former document class level *FocusChanged* event is extended with a new *Reason* called *CdrBeforeFormLoaded*. The event is now also fired right before the desired verification form is about to load but after the *VerifierFormLoad* event described above.

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 Methods and Properties

1.2.2.1 AppendWorkDoc

This event is fired when processing a document separation workflow step in 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.

1.2.2.2 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 showing in this parameter. The developer can set an override value to this parameter, such as every time batch verification completes, always goes to the next invalid batch.

1.2.2.3 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**.

1.2.2.4 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, which was configured within the Runtime Server settings.
`pCancel`	Set this variable to **True** to cancel the export.

1.2.2.5 ForceClassificationReview

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

The following script sample shows how the manual classification process can be forced from custom script event *PostClassify*:

1.2.2.6 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**.

1.2.2.7 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.

1.2.2.8 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.

1.2.2.9 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.

1.2.2.10 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.

1.2.2.11 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.

1.2.2.12 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.
`DesiredOutputStateSucceeded`	The output state of the batch if the workflow step succeeds.
`DesiredOutputStateFailed`	The output state of the batch if the workflow step failed.

1.2.2.13 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.

1.2.2.14 SecurityUpdateAddUserGroup

This method is used to update or add the database security credentials. This script call is used in creating or 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 are updated. The project security is not modified after a script update.

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

Use this method between *SecurityUpdateStart* and *SecurityUpdateCommit*.

Note: If the passed in the *UserName* parameter does not exist in the database then the user is considered as being deleted from the system and marked as **"deleted = true"**, otherwise the user would be recovered and marked as **"deleted = false"**.

The password is updated only during creation or recovery of a user. If an administrator needs to change the password for a script imported user, he would need to first exclude the user from the *SecurityUpdate* call so the user is deleted, and then re-add him with a new password into the next iteration of *SecurityUpdate*.

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

Parameter	Description
`UserName`	The username 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 <*domain*>\<*username*> for logging into Verifier.
`UserPassword`	This password is applied 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 password are required to change their password upon first login. ▪ Auto-imported users with non-empty password are not required to change their password upon first login. ▪ Auto-imported users who already changed their password upon first login will not be required to change their password anymore.
`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**.
`UserRole`	The user role assigned to the Verifier user. The role can be one or a logical combination (separated with a pipe character) of the following text strings: ▪ **SET** Can access settings. ▪ **VER** Verifier user. ▪ **SLV** Verifier supervisor. ▪ **SLM** Learnset manager. ▪ **ADM** Administrator. ▪ **FLT** Filtering.
`UserDomain`	When using Windows authentication, this parameter must be populated with the name of the domain that the Windows user belongs to. This parameter should be left blank if Windows authentication is not used.

For additional information, refer to *SecurityUpdateStart*, *SecurityUpdateCommit*, *UpdateSystemSecurity* and *PostImportBatch*.

1.2.2.15 SecurityUpdateAddUserGroupPwd

This method is used to update or add the database security credentials. This script call is used in creating or 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 are updated. The project security is not modified after a script update.

Use this method between *SecurityUpdateStart* and *SecurityUpdateCommit*.

Note: If the passed in the *UserName* parameter does not exist in the database then the user is considered as being deleted from the system and marked as **"deleted = true"**, otherwise the user would be recovered and marked as **"deleted = false"**.

The password is updated only during creation or recovery of a user. If an administrator needs to change the password for a script imported user, he would need to first exclude the user from the *SecurityUpdate* call so the user is deleted, and then re-add him with a new password into the next iteration of *SecurityUpdate*.

Syntax:

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

Parameter	Description
`UserName`	The username 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 <*domain*>\<*username*> for logging into Verifier.
`UserPassword`	This password is applied 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 password are required to change their password upon first login. ▪ Auto-imported users with non-empty password are not required to change their password upon first login. ▪ Auto-imported users who already changed their password upon first login will not be required to change their password anymore.
`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**.
`UserRole`	The user role assigned to the Verifier user. The role can be one or a logical combination (separated with a pipe character) of the following text strings:

	SET Can access settings.
	VER Verifier user.
	SLV Verifier supervisor.
	SLM Learnset manager.
	ADM Administrator.
	FLT Filtering.
`UserDomain`	When using Windows authentication, this parameter must be populated with the name of the domain that the Windows user belongs to. This parameter should be left blank if Windows authentication is not used.

For additional information, refer to *SecurityUpdateStart*, *SecurityUpdateCommit*, *UpdateSystemSecurity* and *PostImportBatch*.

1.2.2.16 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 are updated. The project security is not modified after a script update.

Syntax: `Project.SecurityUpdateCommit`

For additional information, refer to *SecurityUpdateStart*, *SecurityUpdateAddUserGroup*, *UpdateSystemSecurity* and *PostImportBatch*.

1.2.2.17 SecurityUpdateStart

This method instantiates the security update process. This script call is required in order 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 are updated. The project security is not modified after a script update.

Syntax: `Project.SecurityUpdateStart`

For more information, refer to *SecurityUpdateCommit*, *UpdateSystemSecurity* and *PostImportBatch*.

1.2.2.18 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.

This method needs to be called between *SecurityUpdateStart* and *SecurityUpdateCommit*.

The group settings need to be configured in Web Verifier settings page by an administrator.

Syntax: `SecurityUpdateUserParameter(ByVal UserName As String, ByVal UserDomain As String, ByVal ParameterName As String, ByVal Param1 As Variant, ByVal Param2 As Variant)`

Parameter	Description
`ParameterName`	This parameter value must be set to **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.
`Param1`	This parameter must be set to either **GroupName** or **ExternalGroupID** with the appropriate value passed in *Param2*.
`Param2`	If *Param1* is set to **GroupName** this parameter should contain a string representing the group name as displayed in the Web Verifier group settings. If *Param1* is set to **ExternalGroupID** this parameter should contain an integer that represents the *ExternalGroupID* that was added in *SecurityUpdateAddUserGroup* or *SecurityUpdateAddUserGroupPwd*.

1.2.2.19 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**

1.2.2.20 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.

For additional information, refer to *SecurityUpdateStart*, *SecurityUpdateCommit* and *PostImportBatch*.

1.2.2.21 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.
`ClassName`	The name of the document class to which it is classified manually.

1.2.2.22 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.

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.

1.3 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 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.

1.3.1.2 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.

1.3.1.3 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.

1.3.1.4 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.

1.3.1.5 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.*

1.3.1.6 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.

1.3.1.7 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:

```
<Fieldn>_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**.

1.3.2.2 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.
`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.

1.3.2.3 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.

1.3.2.4 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.

1.3.2.5 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.

1.3.2.6 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.

1.3.2.7 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.

1.3.2.8 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.

1.3.2.9 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.
`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**.

1.3.2.10 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.

1.3.2.11 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.

1.3.2.12 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.

1.3.2.13 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.

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 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.

2.1.2.2 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.
`CDRHighlightCheckedWordsAndFields`	Verified words and fields highlighting.
`CDRHighlightFields`	Fields highlighting.

`CDRHighlightNothing`	No highlighting.
`CDRHighlightParagraphs`	Paragraph highlighting.
`CDRHighlightRectangles`	Variable rectangle highlighting.
`CDRHighlightTables`	Table highlighting.
`CDRHighlightTablesAdvanced`	Highlights checked words and selected table cell, also shows 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.

2.1.2.3 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.

2.1.2.4 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.

2.1.2.5 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.

2.1.2.6 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.

2.1.2.7 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.
`CDRDocFileTypeCroCIDoc`	Cairo CIDocument.

2.1.3 Methods and Properties

2.1.3.1 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.

2.1.3.2 AddField

This method adds a field to the workdoc.

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

Parameter	Description
`Name`	Contains the name for the new field.

2.1.3.3 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.

2.1.3.4 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.

2.1.3.5 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*.

2.1.3.6 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 *CDRHighlightVerticalEdgesRight* 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.

2.1.3.7 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*.

2.1.3.8 AnalyzeParagraphs

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

Syntax: `AnalyzeParagraphs()`

2.1.3.9 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.

2.1.3.10 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

2.1.3.11 AttractorColor

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

Syntax: `AttractorColor As OLE_COLOR`

2.1.3.12 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`

2.1.3.13 BlockColor

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

Syntax: `BlockColor As OLE_COLOR`

2.1.3.14 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`

2.1.3.15 CandidateColor

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

Syntax: `CandidateColor As OLE_COLOR`

2.1.3.16 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()`

2.1.3.17 ClearHighlightRectangles

This method removes all highlighted rectangles.

Syntax: `ClearHighlightRectangles()`

2.1.3.18 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.

2.1.3.19 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.

2.1.3.20 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.

2.1.3.21 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.

2.1.3.22 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.

2.1.3.23 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.

2.1.3.24 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.

2.1.3.25 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.

2.1.3.26 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`

2.1.3.27 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.

2.1.3.28 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`

2.1.3.29 DocClassName

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

Syntax: `DocClassName As String`

2.1.3.30 DocFileCount

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

Syntax: `DocFileCount As Long`

2.1.3.31 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 *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.

2.1.3.32 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.

2.1.3.33 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.

2.1.3.34 DocState

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

Syntax: `DocState As CDRDocState`

2.1.3.35 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.

2.1.3.36 ErrorDescription

This property sets or returns an error description.

Syntax: `ErrorDescription As String`

2.1.3.37 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**.

2.1.3.38 Fields

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

Syntax: `Fields As ISCBCdrFields`

2.1.3.39 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`

2.1.3.40 Folder

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

Syntax: `Folder As ISCBCdrFolder`

2.1.3.41 FolderIndex

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

Syntax: `FolderIndex As Long`

2.1.3.42 ForceClassificationReview

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

2.1.3.43 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.

2.1.3.44 GetFileSizeKB

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

Syntax: `GetFileSizeKB(pWorkdoc As SCBCdrWorkdoc)`

Parameter	Description
`pWorkdoc`	The current workdoc object.

2.1.3.45 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.

2.1.3.46 HighlightCandidate

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

Syntax: `HighlightCandidate As Long`

2.1.3.47 HighlightField

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

Syntax: `HighlightField As Long`

2.1.3.48 HighlightMode

This property sets or returns the current mode of highlighting.

Syntax: `HighlightMode As` *`CDRHighlightMode`*

2.1.3.49 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.

2.1.3.50 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.

2.1.3.51 IsPlainText

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

Syntax: `IsPlainText As Boolean`

2.1.3.52 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`

2.1.3.53 LineColor

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

Syntax: `LineColor As OLE_COLOR`

2.1.3.54 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.

2.1.3.55 PageCount

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

Syntax: `PageCount As Long`

2.1.3.56 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.

2.1.3.57 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.

2.1.3.58 ParagraphCount

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

Syntax: `ParagraphCount As Long`

2.1.3.59 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.

2.1.3.60 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.

2.1.3.61 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.

2.1.3.62 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.

2.1.3.63 Refresh

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

Syntax: `Refresh()`

2.1.3.64 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*.

2.1.3.65 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.

2.1.3.66 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.

2.1.3.67 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*.

2.1.3.68 SetDocPageIndex

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

2.1.3.69 ShowTooltips

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

Syntax: `ShowTooltips As Boolean`

2.1.3.70 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.

2.1.3.71 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.

2.1.3.72 TableCount

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

Syntax: `TableCount As Long`

2.1.3.73 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.

2.1.3.74 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.

2.1.3.75 TextlineCount

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

Syntax: `TextlineCount As Long`

2.1.3.76 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.

2.1.3.77 UnloadDocs

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

Syntax: `UnloadDocs()`

2.1.3.78 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.

2.1.3.79 WordColor

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

Syntax: `WordColor As OLE_COLOR`

2.1.3.80 WordCount

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

Syntax: `WordCount As Long`

2.1.3.81 WordSegmentationChars

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

Syntax: `WordSegmentationChars As String`

2.1.3.82 Worktext

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

Syntax: `Worktext As ISCBCroWorktext`

2.2 SCBCdrFields

2.2.1 Description

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

2.2.2 Methods and Properties

2.2.2.1 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.

2.2.2.2 Clear

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

Syntax: `Clear()`

2.2.2.3 Collection

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

Syntax: `Collection As ISCBCroCollection`

2.2.2.4 Count

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

Syntax: `Count As Long`

2.2.2.5 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.

2.2.2.6 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*.

2.2.2.7 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.

2.2.2.8 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.

2.2.2.9 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.

2.2.2.10 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*.

2.2.2.11 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*.

2.2.2.12 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.

2.2.2.13 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*.

2.2.2.14 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.

2.2.2.15 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*.

2.3 SCBCdrField

2.3.1 Description

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

2.3.2 Type Definitions

2.3.2.1 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.

2.3.3 Methods and Properties

2.3.3.1 ActiveTableIndex

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

Syntax: `ActiveTableIndex As Long`

2.3.3.2 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.

2.3.3.3 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.

2.3.3.4 AddTable

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

Syntax: `AddTable()`

2.3.3.5 BoostDigitsOnly

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

Syntax: `BoostDigitsOnly as Boolean`

2.3.3.6 BoostField

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

Syntax: `BoostField as Boolean`

2.3.3.7 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.

2.3.3.8 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.

2.3.3.9 CandidateCount

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

Syntax: `CandidateCount As Long`

2.3.3.10 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`

2.3.3.11 CustomDetailsString

This property sets or returns the *CustomDetailsString*.

Syntax: `CustomDetailsString As String`

2.3.3.12 CustomStatusLong

This property sets or returns the *CustomStatusLong*.

Syntax: `CustomStatusLong As Long`

2.3.3.13 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.

2.3.3.14 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.

2.3.3.15 ErrorDescription

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

Syntax: `ErrorDescription As String`

2.3.3.16 ExternalText

This property sets or returns the extended text.

Syntax: `ExternalText As String`

2.3.3.17 FieldID

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

Syntax: `FieldID As Long`

2.3.3.18 FieldState

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

Syntax: `FieldState As` *`CDRFieldState`*

2.3.3.19 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.

2.3.3.20 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.

2.3.3.21 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.

2.3.3.22 FormattedText

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

2.3.3.23 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

2.3.3.24 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

2.3.3.25 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.

2.3.3.26 Height

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

Syntax: `Height As Long`

2.3.3.27 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.

2.3.3.28 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`

2.3.3.29 LastModificationEndDate

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

Syntax: `LastModificationEndDate As Date`

2.3.3.30 LastModificationEndDateAsFileTimeUtc

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

Syntax: `LastModificationEndDateAsFileTimeUtc As Date`

2.3.3.31 Left

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

Syntax: `Left As Long`

2.3.3.32 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.

2.3.3.33 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.

2.3.3.34 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`

2.3.3.35 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.

2.3.3.36 MultilineText

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

Syntax: `MultilineText As String`

2.3.3.37 Name

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

Syntax: `Name As String`

2.3.3.38 PageNr

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

Syntax: `PageNr As Long`

2.3.3.39 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.

2.3.3.40 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.

2.3.3.41 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.

2.3.3.42 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.

2.3.3.43 TableCount

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

Syntax: `TableCount As Long`

2.3.3.44 Tag

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

Syntax: `Tag As Variant`

2.3.3.45 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`

2.3.3.46 Top

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

Syntax: `Top As Long`

2.3.3.47 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.

2.3.3.48 Valid

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

Syntax: `Valid As Boolean`

2.3.3.49 Width

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

Syntax: `Width As Long`

2.3.3.50 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`

2.4 SCBCdrCandidate

2.4.1 Description

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

2.4.2 Methods and Properties

2.4.2.1 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

2.4.2.2 AttractorCount

This property returns the number of attractors for this candidate.

Syntax: `AttractorCount As Long`

2.4.2.3 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.

2.4.2.4 FilterID

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

Syntax: `FilterID As Long`

2.4.2.5 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`

2.4.2.6 Height

This property returns the height of the candidate in pixels.

Syntax: `Height As Long`

2.4.2.7 KeepSpaces

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

Syntax: `KeepSpaces As Boolean`

2.4.2.8 Left

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

Syntax: `Left As Long`

2.4.2.9 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.

2.4.2.10 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.

2.4.2.11 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`

2.4.2.12 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.

2.4.2.13 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.

2.4.2.14 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.

2.4.2.15 PageNr

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

Syntax: `PageNr As Long`

2.4.2.16 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.

2.4.2.17 Text

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

Syntax: `Text As String`

2.4.2.18 Top

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

Syntax: `Top As Long`

2.4.2.19 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`

2.4.2.20 Width

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

Syntax: `Width As Long`

2.4.2.21 WordCount

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

Syntax: `WordCount As Long`

2.4.2.22 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.

2.4.2.23 Worktext

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

Syntax: `Worktext As ISCBCroWorktext`

2.5 SCBCdrTable

2.5.1 Description

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

2.5.2 Type Definitions

2.5.2.1 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.

2.5.2.2 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.

2.5.3 Methods and Properties

2.5.3.1 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.

2.5.3.2 AddRow

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

Syntax: `AddRow()`

2.5.3.3 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.

2.5.3.4 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.

2.5.3.5 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.

2.5.3.6 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.

2.5.3.7 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.

2.5.3.8 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.

2.5.3.9 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.

2.5.3.10 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.

2.5.3.11 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.

2.5.3.12 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.

2.5.3.13 Clear

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

Syntax: `Clear()`

2.5.3.14 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.

2.5.3.15 ClearRow

This method clears the content of an existing row.

Syntax: `ClearRow(RowIndex As Long)`

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

2.5.3.16 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.

2.5.3.17 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.

2.5.3.18 ColumnCount

This read-only property returns the number of columns.

Syntax: `ColumnCount As Long`

2.5.3.19 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.

2.5.3.20 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.

2.5.3.21 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.

2.5.3.22 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.

2.5.3.23 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.

2.5.3.24 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.

2.5.3.25 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.

2.5.3.26 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.

2.5.3.27 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.

2.5.3.28 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.

2.5.3.29 DeleteRow

This method deletes a row specified by an index.

Syntax: `DeleteRow(RowIndex As Long)`

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

2.5.3.30 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.

2.5.3.31 FieldName

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

Syntax: `FieldName As String`

2.5.3.32 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.

2.5.3.33 FooterLocation

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

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

Parameter	Description
`Location`	Location parameter.

2.5.3.34 FooterPageNr

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

Syntax: `FooterPageNr As Long`

2.5.3.35 FooterText

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

Syntax: `FooterText As String`

2.5.3.36 HeaderLocation

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

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

Parameter	Description
`Location`	Location parameter.

2.5.3.37 HeaderPageNr

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

Syntax: `HeaderPageNr As Long`

2.5.3.38 HeaderText

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

Syntax: `HeaderText As String`

2.5.3.39 HightlightColumnIndex

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

Syntax: `HighlightColumnIndex As Long`

2.5.3.40 HighlightMode

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

Syntax: `HighlightMode As CDRTableHighlightMode`

2.5.3.41 HighlightRowIndex

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

Syntax: `HighlightRowIndex As Long`

2.5.3.42 HighlightUMColumnIndex

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

Syntax: `HighlightUMColumnIndex As Long`

2.5.3.43 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.

2.5.3.44 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.

2.5.3.45 InsertUMColumn

This method inserts a new, unmapped column.

Syntax: `InsertUMColumn(UMColumnIndex As Long)`

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

2.5.3.46 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`

2.5.3.47 LocationExplicit

This property sets and returns the *LocationExplicit* flag.

Syntax: `LocationExplicit As Boolean`

2.5.3.48 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.

2.5.3.49 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.

2.5.3.50 RemoveAllColumns

This method removes all mapped table columns.

Syntax: `RemoveAllColumns()`

2.5.3.51 RemoveAllRows

This method removes all table rows.

Syntax: `RemoveAllRows()`

2.5.3.52 RemoveAllUMColumns

This method removes all unmapped table columns.

Syntax: `RemoveAllUMColumns()`

2.5.3.53 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.

2.5.3.54 RowCount

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

Syntax: `RowCount As Long`

2.5.3.55 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.

2.5.3.56 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.

2.5.3.57 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.

2.5.3.58 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.

2.5.3.59 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.

2.5.3.60 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).

2.5.3.61 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.

2.5.3.62 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.

2.5.3.63 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`

2.5.3.64 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`

2.5.3.65 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.

2.5.3.66 TableValid

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

Syntax: `TableValid As Boolean`

2.5.3.67 TableValidationErrorDescription

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

Syntax: `TableValidationErrorDescription As String`

2.5.3.68 Tag

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

Syntax: `Tag As String`

2.5.3.69 TotalSignificance

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

Syntax: `TotalSignificance As Double`

2.5.3.70 UMCellColor

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

Syntax: `UMCellColor As OLE_COLOR`

2.5.3.71 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.

2.5.3.72 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.

2.5.3.73 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.

2.5.3.74 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.

2.5.3.75 UMColumnColor

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

Syntax: `UMColumnColor As OLE_COLOR`

2.5.3.76 UMColumnCount

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

Syntax: `UMColumnCount As Long`

2.5.3.77 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.

2.5.3.78 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.

2.5.3.79 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.

2.5.3.80 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.

2.5.3.81 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.

2.5.3.82 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).

2.6 SCBCdrTextBlock

2.6.1 Description

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

2.6.2 Methods and Properties

2.6.2.1 Color

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

Syntax: `Color As OLE_COLOR`

2.6.2.2 Height

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

Syntax: `Height As Long`

2.6.2.3 Left

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

Syntax: `Left As Long`

2.6.2.4 PageNr

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

Syntax: `PageNr As Long`

2.6.2.5 Text

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

Syntax: `Text As String`

2.6.2.6 Top

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

Syntax: `Top As Long`

2.6.2.7 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`

2.6.2.8 Weight

This read-only property returns the text block weight.

Syntax: `Weight As Double`

2.6.2.9 Width

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

Syntax: `Width As Long`

2.6.2.10 WordCount

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

Syntax: `WordCount As Long`

2.6.2.11 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

2.7 SCBCdrWord

2.7.1 Description

This object represents a textual word of a document.

2.7.2 Methods and Properties

2.7.2.1 Color

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

Syntax: `Color As OLE_COLOR`

2.7.2.2 Height

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

Syntax: `Height As Long`

2.7.2.3 Left

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

Syntax: `Left As Long`

2.7.2.4 PageNr

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

Syntax: `PageNr As Long`

2.7.2.5 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`

2.7.2.6 Text

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

Syntax: `Text As String`

2.7.2.7 TextLen

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

Syntax: `TextLen As Long`

2.7.2.8 Tooltip

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

Syntax: `Tooltip As String`

2.7.2.9 Top

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

Syntax: `Top As Long`

2.7.2.10 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`

2.7.2.11 Width

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

Syntax: `Width As Long`

2.7.2.12 Worktext

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

Syntax: `Worktext As ISCBCroWorktext`

2.8 SCBCdrDocPage

2.8.1 Description

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

2.8.2 Type Definitions

2.8.2.1 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.

2.8.2.2 CroLinesDir

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

Available Types	Description
`CroLinesDir_Horizontal`	Horizontal line.
`CroLinesDir_Vertical`	Vertical line.

2.8.2.3 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.

2.8.3 Methods and Properties

2.8.3.1 DisplayImage

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

Syntax: `DisplayImage As Long`

2.8.3.2 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.

2.8.3.3 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.

2.8.3.4 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.

2.8.3.5 Height

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

Syntax: `Height As Double`

2.8.3.6 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.

2.8.3.7 ImageCount

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

Syntax: `ImageCount As Long`

2.8.3.8 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.)

2.8.3.9 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).

2.8.3.10 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`

2.8.3.11 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`*

2.8.3.12 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**.

2.8.3.13 Rotation

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

Syntax: `Rotation As Double`

2.8.3.14 Text

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

Syntax: `Text As String`

2.8.3.15 Width

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

Syntax: `Width As Double`

2.9 SCBCdrFolder

2.9.1 Description

A folder may represent an array of workdocs within a batch. A folder may contain one or more workdocs. During classification and extraction, it is possible to access all workdocs of the same folder from script.

2.9.2 Methods and Properties

2.9.2.1 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.

2.9.2.2 Clear

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

Syntax: `Clear()`

2.9.2.3 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.

2.9.2.4 DocumentCount

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

Syntax: `DocumentCount As Long`

2.9.2.5 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.

2.9.2.6 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.

2.9.2.7 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.

2.9.2.8 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.

3 Project Object Reference (SCBCdrPROJLib)

3.1 Description

The *Project* object represents a complete project definition, including all document classes, field definitions, and used classification and extraction methods.

3.2 Type Definitions

3.2.1.1 CDRBatchReleaseAction

Represents the action taken when the last document of the batch has been verified.

Available Types	Description
`CdrBatchReleaseAction Cancel`	Return to current batch and last document verified.
`CdrBatchReleaseActionReturnToList`	Return to batch list.
`CdrBatchReleaseActionUndefined`	Unknown action.
`CdrBatchReleaseActionUserDefined`	Default. Verifier user makes a selection on next action to take on batch release.
`CdrBatchReleaseActionVerifyNextInvalidBatch`	Open next batch to verify.
`CdrBatchReleaseActionVerifyNextInvalidState`	Open current batch to verify in the next invalid state.

3.2.1.2 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.

3.2.1.3 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.

3.2.1.4 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`	Let the Verifier user decide to skip special processing altogether.
`CdrDoSmartDecision`	Make 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 assign current document class name to the supplier field content.
`CdrUseSupplierField`	Automatically assign supplier field content to the document class name.

3.2.1.5 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.

3.2.1.6 CdrMessageType

This type defines the different message types.

Available Types	Description
`CDRTypeInfo`	An informational message.
`CDRTypeWarning`	A warning message.
`CDRTypeError`	An error message.

3.2.1.7 CdrMessageSeverity

This type defines the different message severities.

Available Types	Description
`CDRSeverityLogFileOnly`	Store the message to the application log file only.
`CDRSeveritySystemMonitoring`	Store the message in the log file and forward it to the host instance's MMC console and to the System Monitoring service of the Runtime Server. This option is applicable when the call is executed from within the Runtime Server application only.
`CDRSeverityEmailNotification`	Store 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.

3.2.2 Methods and Properties

3.2.2.1 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.

3.2.2.2 AllClasses

This read-only property returns a collection of all defined document classes of this project.

Syntax: `AllClasses As` *`ISCBCdrDocClasses`*

3.2.2.3 BaseClasses

This read-only property returns a collection that contains all defined base document classes.

Syntax: `BaseClasses As` *`ISCBCdrDocClasses`*

3.2.2.4 ClassificationMode

This property sets or returns the used classification mode.

Syntax: `ClassificationMode As` *`CDRClassifyMode`*

3.2.2.5 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`

3.2.2.6 DefaultLanguage

This read-only property returns the language used as default.

Syntax: `DefaultLanguage As String`

3.2.2.7 Filename

This read-only property returns the file name of the project including the directory path.

Syntax: `Filename As String`

3.2.2.8 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`*

3.2.2.9 GetVerifierProject

This method returns the Verifier project.

Syntax: `GetVerifierProject(ppVal As Object)`

Parameter	Description
`ppVal`	Returns the Verifier project object.

3.2.2.10 LastAddressPoolUpdate

This read-only property returns the last time when the address pool was updated.

Syntax: `LastAddressPoolUpdate As Date`

3.2.2.11 Lock

This method locks the project for updating.

Syntax: `Lock()`

3.2.2.12 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.

3.2.2.13 MinClassificationDistance

This property sets or returns the minimal distance of classification results.

Syntax: `MinClassificationDistance As Double`

3.2.2.14 MinClassificationWeight

This property sets or returns the minimal classification weight.

Syntax: `MinClassificationWeight As Double`

3.2.2.15 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`

3.2.2.16 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`

3.2.2.17 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.

3.2.2.18 NoUI

If this property is set to **True**, then no login dialog box displays.

Syntax: `NoUI As Boolean`

3.2.2.19 Page

This read-only property returns the Cairo Page object of the current project.

Syntax: `Page As ISCBCroPage`

3.2.2.20 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`

3.2.2.21 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.

3.2.2.22 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()`

3.2.2.23 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
`ReportActiveLicensingOnly`	A Boolean flag to indicate if all licensing counters should be outputted (**False**), or if only the license counters active in the license file should be outputted (**True**).
`Severity`	The location of the utilization output to be sent to. This relates to the defined types shown in CdrMessageSeverity type definition (i.e. log file, email, or RTS System Monitoring).

3.2.2.24 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.

3.2.2.25 SLWDifferentResultsAction

This property sets or returns the action to complete if a template classification and supplier extraction has different results.

Syntax: `SLWDifferentResultsAction As CdrSLWDifferentResultsAction`

3.2.2.26 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`

3.2.2.27 Unlock

This method unlocks the project after updating.

Syntax: `Unlock()`

3.2.2.28 UpdateAddressPool

This method updates the address analysis pool.

Syntax: `UpdateAddressPool()`

3.2.2.29 ValidationSettingsColl

This read-only property returns a collection of all activated validation engines.

Syntax: `ValidationSettingsColl As ISCBCroCollection`

3.2.2.30 ValidationTemplates

This read-only property returns a collection of all available validation templates.

Syntax: `ValidationTemplates As ISCBCroCollection`

3.2.2.31 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.

3.2.2.32 WordSegmentationChars

This property sets or returns a string that contains all characters used for word segmentation.

Syntax: `WordSegmentationChars As String`

3.3 SCBCdrDocClasses

3.3.1 Description

This collection contains all defined *DocClass* objects of the project.

3.3.2 Methods and Properties

3.3.2.1 Collection

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

Syntax: `Collection As ISCBCroCollection`

3.3.2.2 Count

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

Syntax: `Count As Long`

3.3.2.3 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.

3.3.2.4 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*.

3.3.2.5 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.

3.3.2.6 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.

3.3.2.7 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.

3.3.2.8 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*.

3.3.2.9 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*.

3.4 SCBCdrDocClass

3.4.1 Description

A *DocClass* object represents a single document class within a project class hierarchy.

3.4.2 TypeDefinitions

3.4.2.1 CdrFocusChangeReason

This enumeration defines the reason for the focus change of a Verifier field edit.

Available Types	Description
`CdrEnterPressed`	Focus changed by pressing [Enter].
`CdrFcrCandidateCopied`	Focus changed 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.

3.4.2.2 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.

3.4.2.3 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.

3.4.2.4 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.

3.4.3 Methods and Properties

3.4.3.1 ClassificationField

Use this property to read or write the name of the field that is used for the classification.

Syntax: `ClassificationField As String`

3.4.3.2 ClassificationRedirection

This property sets or returns the name of the target *DocClass*.

Syntax: `ClassificationRedirection As String`

3.4.3.3 ClassifySettings

This read-only property returns a collection of chosen classification engines and their settings for this *DocClass*.

Syntax: `ClassifySettings As ISCBCroCollection`

3.4.3.4 DerivedDocClasses

This read-only property returns a collection of all document classes derived directly from this *DocClass*.

Syntax: `DerivedDocClasses As ISCBCdrDocClasses`

3.4.3.5 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`

3.4.3.6 Fields

This read-only property provides access to the *FieldDefs* collection of a document class.

Syntax: `Fields As ISCBCdrFieldDefs`

3.4.3.7 FillRectangle

This method allows the developer to draw a square on the image, which is used to blank out a certain area on the page.

By utilizing the *FillRectangle* method of the *SCBCroImage* object, you can perform image redaction.

3.4.3.8 ForceSubtreeClassification

This property sets or returns whether the classification to the subtree of this document class is forced.

Syntax: `ForceSubtreeClassification As Boolean`

3.4.3.9 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`

3.4.3.10 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.

3.4.3.11 Hidden

This property specifies whether the document class should be visible in the Designer application.

Syntax: `Hidden As Boolean`

3.4.3.12 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.

3.4.3.13 ManualTableTrainingMode

This property sets or returns the option for manual table extraction training mode.

Syntax: `ManualTableTrainingMode As Boolean`

3.4.3.14 Name

This property reads or writes the name of the document class.

Syntax: `Name As String`

3.4.3.15 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`

3.4.3.16 Parent

This property returns the parent document class of the actual document class.

Syntax: `Parent As ISCBCdrDocClass`

3.4.3.17 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.

3.4.3.18 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.

3.4.3.19 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.

3.4.3.20 SubtreeClsMinDist

This property sets or returns the minimal distance to the classification weight of the derived document classes.

Syntax: `SubtreeClsMinDist As Double`

3.4.3.21 SubtreeClsMinWeight

This property sets or returns the minimal classification weight of the derived document classes.

Syntax: `SubtreeClsMinWeight As Double`

3.4.3.22 UseDerivedValidation

This property sets or returns a Boolean value, when derived validation rules are used.

Syntax: `UseDerivedValidation As Boolean`

3.4.3.23 ValidationSettingsColl

This read-only property returns a collection of all activated validation engines.

Syntax: `ValidationSettingsColl As ISCBCroCollection`

3.4.3.24 ValidationTemplateName

This property sets or returns the name of the validation template.

Syntax: `ValidationTemplateName As String`

3.4.3.25 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`

3.4.3.26 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`

3.5 SCBCdrFieldDefs

3.5.1 Description

This collection contains all defined *FieldDef* objects of a single document class.

3.5.2 Methods and Properties

3.5.2.1 Collection

This read-only property returns the collection that is internally used to store the *FieldDef* objects.

Syntax: `Collection As ISCBCroCollection`

3.5.2.2 Count

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

Syntax: `Count As Long`

3.5.2.3 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.

3.5.2.4 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.

3.5.2.5 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.

3.5.2.6 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.

3.5.2.7 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.

3.5.2.8 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*.

3.5.2.9 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*.

3.6 SCBCdrFieldDef

3.6.1 Description

A *FieldDef* object represents the definition of a single field inside a document class.

3.6.2 Type Definitions

3.6.2.1 CdrFieldFormat

This type defines the default format of a certain field.

Available Types	Description
`CdrFieldFormatCurrency`	Currency.
`CdrFieldFormatDate`	Date.
`CdrFieldFormatExtNumber`	Extended number.
`CdrFieldFormatNone`	None.
`CdrFieldFormatNumber`	Number.

3.6.2.2 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.

3.6.2.3 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.

3.6.2.4 CdrValFieldType

This enumeration contains different validation types for fields.

Available Types	Description
`CdrAmountValidation`	Used for amount values or general numeric values.
`CdrChkboxValidation`	Field as used check box.
`CdrCustomValidation`	*TBD*

`CdrDateValidation`	Used for date values.
`CdrListValidation`	Used for lists.
`CdrTableValidation`	Used for tables.
`CdrTextValidation`	Used for text values, strings.

3.6.3 Methods and Properties

3.6.3.1 AlwaysValid

This property sets or returns if the content of this field is always valid.

Syntax: `AlwaysValid As Boolean`

3.6.3.2 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.

3.6.3.3 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.

3.6.3.4 ColumnCount

This read-only property returns the number of table columns if *FieldType* is **Table**.

Syntax: `ColumnCount As Long`

3.6.3.5 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.

3.6.3.6 DefaultValidationSettings

This read-only property returns the validation settings with the default language.

Syntax: `DefaultValidationSettings As ISCBCdrValidationSettings`

3.6.3.7 Derived

This read-only property returns **True** if the field properties are derived from an upper document class.

Syntax: `Derived As Boolean`

3.6.3.8 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`

3.6.3.9 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.

3.6.3.10 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.

3.6.3.11 FieldID

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

Syntax: `FieldID As Long`

3.6.3.12 FieldType

This property sets or returns the type of the field.

Syntax: `FieldType As CDRFieldType`

3.6.3.13 ForceValidation

This property sets or returns the mode for the Force Validation.

Syntax: `ForceValidation As CdrForceValidationMode`

3.6.3.14 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.

3.6.3.15 ListItemCount

This read-only property returns the number of strings in the *ListItem* list.

Syntax: `ListItemCount As Long`

3.6.3.16 MaxLength

This property sets or returns the maximum number of characters permitted for this field.

Syntax: `MaxLength As Long`

3.6.3.17 MinLength

This property sets or returns the minimal number of characters permitted for this field.

Syntax: `MinLength As Long`

3.6.3.18 Name

This property sets or returns the name of the field.

Syntax: `Name As String`

3.6.3.19 NoRejects

This property sets or returns if rejects are permitted.

Syntax: `NoRejects As Boolean`

3.6.3.20 OCRConfidence

This property sets or returns the confidence level for OCR. The value must be between **0** and **100**.

Syntax: `OCRConfidence As Long`

3.6.3.21 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.

3.6.3.22 SmartIndex

This property contains all definitions about smart indexing.

Syntax: `SmartIndex As ISCBCdrSmartIndex`

3.6.3.23 UseDerivedOCRSettings

This property sets or returns if OCR settings of the parent document class are used.

Syntax: `UseDerivedOCRSettings As Boolean`

3.6.3.24 UserDerivedValidation

This property sets or returns if the derived validation rules are used for validation of this field.

Syntax: `UseDerivedValidation As Boolean`

3.6.3.25 UseMaxLen

This property sets or returns if the maximum number of characters is limited to the value given by *MaxLength*.

Syntax: `UseMaxLen As Boolean`

3.6.3.26 UseMinLen

This property sets or returns if the minimum number of characters is defined by the value given by *MinLength*.

Syntax: `UseMinLen As Boolean`

3.6.3.27 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.

3.6.3.28 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.

3.6.3.29 ValidationType

This read-only property returns the type of validation.

Syntax: `ValidationType As CdrValFieldType`

3.6.3.30 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.

3.7 SCBCdrSettings

3.7.1 Description

The *Settings* object stores arbitrary strings for usage in script.

3.7.2 Methods and Properties

3.7.2.1 ActiveClient

This property sets or returns the name of the currently active client.

Syntax: `ActiveClient As String`

3.7.2.2 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.

3.7.2.3 AddKey

This method adds a new key specified by its name and its parent. Refer to 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*.

3.7.2.4 Clear

This method clears all clients and keys from the *Settings* object.

Syntax: `Clear()`

3.7.2.5 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.

3.7.2.6 ClientCount

This read-only property returns the number of clients

Syntax: `ClientCount As Long`

3.7.2.7 GlobalLearnsetPath

This property sets or returns the global learnset path.

Syntax: `GlobalLearnsetPath As String`

3.7.2.8 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.

3.7.2.9 KeyCount

This read-only property returns the number of keys.

Syntax: `KeyCount As Long`

3.7.2.10 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.

3.7.2.11 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.

3.7.2.12 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.

3.7.2.13 ProjectFileName

This property sets or returns the file name of the project.

Syntax: `ProjectFileName As String`

3.7.2.14 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.

3.7.2.15 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.

3.7.2.16 SupervisedLearningDisabled

This property sets or returns the state of supervised learning in Designer and Verifier workstations.

Syntax: `SupervisedLearningDisabled As Boolean`

3.7.2.17 TopDownEventSequence

This property sets or returns the value of a top-down event sequence.

Syntax: `TopDownEventSequence As Boolean`

3.7.2.18 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.

3.8 SCBCdrScriptModule

3.8.1 Description

This is a global object at the project level. All script module events occurring at project level belong to this object.

3.8.2 Methods and Properties

3.8.2.1 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`

3.8.2.2 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.

3.8.2.3 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.

3.9 SCBCdrScriptProject

3.9.1 Type Definitions

3.9.1.1 CDRApplicationName

Available Types	Description
`TANDesigner`	WebCenter Forms Recognition Designer.
`TANLearnSetManager`	WebCenter Forms Recognition Learn Set Manager.
`TANLocalVerifier`	WebCenter Forms Recognition Verifier used as a local project for Supervised Learning Workflow.
`TANRuntimeServer`	WebCenter Forms Recognition Runtime Server instance.
`TANUnknown`	Unknown application.
`TANVerifier`	WebCenter Forms Recognition Verifier.
`TANWebVerifier`	WebCenter Forms Recognition Web Verifier.

3.9.2 Methods and Properties

3.9.2.1 CurrentClient

This property retrieves and sets the client attribute of the batch.

Syntax:
```
CurrentClient As String
```

3.9.2.2 GetHostProperties

This method allows the user to retrieve information about the current machine, application, 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`	Represents the calling application by a *CDRApplicationName* type. The parameter can be read from script.
`appSubType`	*Only used for internal purposes.*
`appInstance`	The name of the WebCenter Forms Recognition Runtime Service instance, if *appType* is **TANRuntimeServer**. Not used for other applications.
`appUsername`	Login Name of the current WebCenter Forms Recognition user for Designer, Verifier, Learn Set Manager or Web Verifier. Windows user for Runtime Server.
`appIP`	IP address of the computer.
`appMachineName`	Machine name that is running the script.
`appLicense`	Customer name of the used license file.

3.10 SCBCdrScriptAccess

3.10.1 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.

3.10.2 Methods and Properties

3.10.2.1 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.

3.10.2.2 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.

3.10.2.3 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 CDRADSLib

4.1 SCBCdrSupExSettings

4.1.1 Description

This collection contains the functions for the Associative Search engine.

4.1.2 Methods and Properties

4.1.2.1 ClearFilterAttributes

This method clears all existing filters of the multi-column attribute Search.

Syntax: `ClearFilterAttributes()`

4.1.2.2 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 Analysis Engines Object Reference

5.1 SCBCdrAssociativeDbExtractionSettings

5.1.1 Description

This interface covers all methods and properties that are required for controlling and accessing the universal format of the ASSA engine's pool.

5.1.2 Type Definitions

5.1.2.1 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.

5.1.3 Methods and Properties

5.1.3.1 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.

5.1.3.2 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.

5.1.3.3 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.

5.1.3.4 ClassNameFormat

This property sets or returns the format definition for a document class name.

Syntax: `ClassNameFormat As String`

5.1.3.5 ColumnCount

This read-only property returns the number of columns of a currently opened pool.

Syntax: `ColumnCount As Long`

5.1.3.6 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.

5.1.3.7 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.

5.1.3.8 CommitUpdate

This method closes and saves the currently opened pool.

Syntax: `CommitUpdate()`

5.1.3.9 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`

5.1.3.10 EntryCount

This read-only property returns the number of entries of the pool.

Syntax: `EntryCount As Long`

5.1.3.11 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`

5.1.3.12 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`

5.1.3.13 FieldContentsFormat

This property sets or returns the format definition for the representation of the engine.

Syntax: `FieldContentsFormat As String`

5.1.3.14 FindLocation

This property sets or returns if address analysis is enabled. If **True**, the position of the address is found.

Syntax: `FindLocation As Boolean`

5.1.3.15 GeneratePool

This method imports the pool from the source specified in the *AutomaticImportMethod* property.

Syntax: `GeneratePool()`

5.1.3.16 GeneratePoolFromCsvFile

This method removes the previous pool and generates a new one using the CSV file, designed in the new format.

Syntax: `GeneratePoolFromCsvFile()`

5.1.3.17 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()`

5.1.3.18 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.

5.1.3.19 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.

5.1.3.20 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.

5.1.3.21 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.

5.1.3.22 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.

5.1.3.23 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.

5.1.3.24 IdentityColumn

This property sets or returns the unique ID of a column name.

Syntax: `IdentityColumn As String`

5.1.3.25 ImportFieldNames

This property sets or returns if the column names are taken from the first line of a CSV file.

Syntax: `ImportFieldNames As Boolean`

5.1.3.26 ImportFileName

This property sets or returns the filename of the CSV file.

Syntax: `ImportFileName As String`

5.1.3.27 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`

5.1.3.28 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.

5.1.3.29 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.

5.1.3.30 LastImportTimeStamp

This read-only property returns the timestamp for the last import.

Syntax: `LastImportTimeStamp As Date`

5.1.3.31 MaxCandidates

This property sets or returns the maximum number of results of the associative search engine.

Syntax: `MaxCandidates As Long`

5.1.3.32 MinDistance

This property sets or returns the required minimum distance to the next best candidate for a valid result.

Syntax: `MinDistance As Double`

5.1.3.33 MinRelevance

This property sets or returns the minimum relevance for search results. The default value is **0.0**.

Syntax: `MinRelevance As Double`

5.1.3.34 MinThreshold

This property sets or returns the required minimum value for a valid engine result.

Syntax: `MinThreshold As Double`

5.1.3.35 ODBCName

This property sets or returns the name of the ODBC source.

Syntax: `ODBCName As String`

5.1.3.36 Password

This property sets or returns the password of the ODBC source.

Syntax: `Password As String`

5.1.3.37 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.

5.1.3.38 PhrasesCount

This read-only property returns the number of phrases used for address analysis.

Syntax: `PhrasesCount As Long`

5.1.3.39 PoolName

This property sets or returns the name of the associative search pool.

Syntax: `PoolName As String`

5.1.3.40 PoolPath

This property sets or returns the file path of the associative search pool.

Syntax: `PoolPath As String`

5.1.3.41 PoolPathRelative

This property sets or returns if the pool should be saved relative to the path of the project.

Syntax: `PoolPathRelative As Boolean`

5.1.3.42 ProjectPath

This read-only property returns the path of the project file.

Syntax: `ProjectPath As String`

5.1.3.43 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.

5.1.3.44 SavePoolInternal

This property sets or returns if a pool should be saved within the project file or as separate files.

Syntax: `SavePoolInternal As Boolean`

5.1.3.45 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.

5.1.3.46 Separator

This property sets or returns a separator, either a semicolon or comma, that is used for CSV files.

Syntax: `Separator As String`

5.1.3.47 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: 6. First page header. 7. First page footer. 8. Subsequent pages header. 9. Subsequent pages footer. 10. Last page header. 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.

5.1.3.48 SQLQuery

This property sets or returns an SQL statement used to import ODBC source.

Syntax: `SQLQuery As String`

5.1.3.49 StartAddEntry

This method prepares the insertion of a new entry to the associative search pool.

Syntax: `StartAddEntry()`

5.1.3.50 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.

5.1.3.51 UserName

This property sets or returns the user name required to login in to the ODBC source.

Syntax: `Username As String`

5.1.3.52 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`

6 StringComp Object Reference (SCBCdrSTRCOMPLib)

6.1 SCBCdrStringComp

6.1.1 Description

This component provides several implementations of string compare algorithms.

6.1.2 Type Definitions

6.1.2.1 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.

6.1.3 Methods and Properties

6.1.3.1 CaseSensitive

This property controls if the compare algorithm should work case-sensitive.

Syntax: `CaseSensitive As Boolean`

6.1.3.2 CompTypes

This property selects the compare algorithm used for the next call of *Distance*.

Syntax: `CompType As` *`CdrCompareType`*

6.1.3.3 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**.

6.1.3.4 LevDeletions

This read-only property returns the count of deletions calculated by the last *Distance* function.

Syntax: `LevDeletions As Single`

6.1.3.5 LevInsertions

This read-only property returns the count of insertions calculated by the last *Distance* function.

Syntax: `LevInsertions As Single`

6.1.3.6 LevRejects

This read-only property returns the count of rejects calculated by the last *Distance* function.

Syntax: `LevRejects As Single`

6.1.3.7 LevReplacements

This read-only property returns the count of replacements calculated by the last *Distance* function.

Syntax: `LevReplacements As Single`

6.1.3.8 LevSame

This read-only property returns the count of equal characters calculated by the last *Distance* function.

Syntax: `LevSame As Single`

6.1.3.9 LevTraceMatrix

This read-only property returns the Levenshtein trace matrix calculated by the last *Distance* function.

Syntax: `LevTraceMatrix As String`

6.1.3.10 LevTraceResult

This read-only property returns the Levenshtein trace result calculated by the last *Distance* function.

Syntax: `LevTraceResult As String`

6.1.3.11 MatchEndPosition

This read-only property returns the matching end position calculated by the last *Distance* function.

Syntax: `MatchEndPosition As Single`

6.1.3.12 MatchStartPosition

This read-only property returns the matching start position calculated by the last *Distance* function.

Syntax: `MatchStartPosition As Single`

6.1.3.13 SearchExpression

This property contains the search expression that should be used for the next compare operation.

Syntax: `SearchExpression As String`

6.1.3.14 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.

6.2 SCBCdrEmailProperties

6.2.1 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.

6.2.2 Type Definitions

6.2.2.1 CdrMessageSeverity

This type defines the different message severities.

Available Types	Description
`CDRSeverityLogFileOnly`	Store the message to the application log file only.

`CDRSeveritySystemMonitoring`	Store the message in the log file and forward it to the host instance's MMC console and to the System Monitoring service of the Runtime Server. This option is applicable when the call is executed from within the Runtime Server application only.
`CDRSeverityEmailNotification`	Store 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.

6.3 SCBCdrLicenseInfoAccess

6.3.1 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.

6.3.2 Methods

6.3.2.1 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.

6.3.2.2 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.

6.3.2.3 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.

6.3.2.4 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.

7 Verifier Component Library

7.1 SCBCdrVerificationForm

7.1.1 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.

7.1.2 Properties

7.1.2.1 DefaultLabelFont

This property sets or returns the default font for all label elements available on this verification form.

Syntax: `DefaultLabelFont As StdFont`

7.1.2.2 DefaultLabelFontColor

This property sets or returns the default color for all label elements available on this verification form.

Syntax: `DefaultLabelFontColor As OLE_COLOR`

7.1.2.3 DefaultLabelBackgroundColor

This property sets or returns the default background color for all label elements available on this verification form.

Syntax: `DefaultLabelBackgroundColor As OLE_COLOR`

7.1.2.4 DefaultFieldFont

This property sets or returns the default font for all verification field elements available on this verification form.

Syntax: `DefaultFieldFont As StdFont`

7.1.2.5 DefaultFieldFontColor

This property sets or returns the default color for all verification field elements available on this verification form.

Syntax: `DefaultFieldFontColor As OLE_COLOR`

7.1.2.6 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`

7.1.2.7 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`

7.1.2.8 FormBackgroundColor

This property sets or returns the background color for the form.

Syntax: `FormBackgroundColor As OLE_COLOR`

7.1.2.9 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`

7.2 SCBCdrVerificationField

7.2.1 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.

7.2.2 Type Definitions

7.2.2.1 CdrVerifierFieldType

This is the Verifier field type. This type interface is a member of the Verifier project library.

Available Types	Description
`CDRVerifierFieldTypeCheckbox`	Checkbox 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.

7.2.3 Properties

7.2.3.1 AutoCompletionEnabled

This property enables or disables automatic completion for a verification field.

Syntax: `AutoCompletionEnabled As Boolean`

7.2.3.2 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`

7.2.3.3 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`

7.2.3.4 Font

This property sets the font for the content of the verification field.

Syntax: `Font As StdFont`

7.2.3.5 FontColor

This property sets the font color for the content of the verification field.

Syntax: `FontColor As OLE_COLOR`

7.2.3.6 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`

7.2.3.7 Left

This property provides the left position of the field on the Verifier form.

Syntax: `Left As Long`

7.2.3.8 Name

This property provides the name of the field on the Verifier form.

Syntax: `Name As String`

7.2.3.9 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`

7.2.3.10 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`

7.2.3.11 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`

7.2.3.12 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`*

7.2.3.13 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`

7.3 SCBCdrVerificationTable

7.3.1 Description

This interface is used to identify verification properties specific for table validation elements.

7.3.2 Methods and Properties

7.3.2.1 Font

This property sets or returns the font settings for the individual table field element.

Syntax: `Font As StdFont`

7.3.2.2 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`

7.3.2.3 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`

7.3.2.4 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`

7.3.2.5 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`

7.3.2.6 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`

7.4 SCBCdrVerificationButton

7.4.1 Description

Use this interface to set verification properties specific for all custom buttons defined on a verification form.

7.4.2 Properties

7.4.2.1 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`

7.4.2.2 FontColor

This property sets or returns the font color for the individual custom button control.

Syntax: `FontColor As OLE_COLOR`

7.4.2.3 BackgroundColor

This property sets or returns background color for the individual custom button control.

Syntax: `BackgroundColor As OLE_COLOR`

7.5 SCBCdrVerificationLabel

7.5.1 Description

This object is part of the 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.

7.5.2 Properties

7.5.2.1 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`

7.5.2.2 Font

This property sets the font for the content of the verification field label.

Syntax: `Font As StdFont`

7.5.2.3 FontColor

This property sets the font color for the content of the verification field label.

Syntax: `FontColor As OLE_COLOR`

7.5.2.4 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`

7.5.2.5 Left

This property provides the left position of the field on the Verifier form.

Syntax: `Left As Long`

7.5.2.6 Name

This property provides the name of the field label on the Verifier form.

Syntax: `Name As String`

7.5.2.7 Text

This property allows the project developer to set the text of the verification field label on the Verifier form.

Syntax: `Text As String`

7.5.2.8 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`

7.5.2.9 Width

This property provides the width size information of the field label on the Verifier form.

Syntax: `Width As Long`

8 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.

8.1 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
```

8.2 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.