Das
COM-Interface
von
CONVAL® 12

Version vom 10. Okt. 2025

Inhalt

Einleitung

CONVAL® 12 verfügt über direkte Schnittstellen, die es ermöglichen CONVAL® als COM-Server unter Windows zu verwenden. Diese Schnittstellenobjekte (Interfaces) können z. B. aus VisualBasic, VBA, VBScript, JScript, Delphi etc. verwendet werden, um CONVAL® zu steuern.

Für die Einbindung in ein Produkt stellt die F.I.R.S.T. eine Typenbibliothek und diese Dokumentation zur Verfügung. Die Dokumentation besteht aus der grafischen Darstellung der Objekthierarchie, beschreibt sämtliche Funktionen der Interfaces und legt die Typenbibliothek in IDL offen.

Die Beispiele wurden in keiner speziellen Programmiersprache verfasst, da die COM-Schnittstelle von jeder Programmiersprache verwendet werden kann, die Zeiger bzw. COM unterstützt. Sie orientieren sich jedoch grob an BASIC. In den Beispielen wird vorausgesetzt, dass ein Objekt vom Interface *IConval9* instanziiert worden ist und einer Variablen *AConval* zugewiesen wurde. Da hier die Syntax stark von der verwendeten Sprache und deren Logik abhängt, lesen Sie bitte in der jeweiligen Dokumentation nach, wie die Variable *AConval* instanziiert wird.

Konventionen

Am linken Seitenrand sind Beispiele blau markiert, Anmerkungen grün und wichtige Hinweise rot.

Die einzelnen Attribute, Operationen und Ereignisse der Interfaces werden wie folgt dargestellt (Bsp.):

Beispiel

NewDialog

```
function NewDialog(CalcKind: long): ICalculationDialog;
```

Erzeugt eine neue Instanz eines Dialogs der angegebenen Berechnung und gibt das *IDialog* Interface zurück. Der Dialog ist standardmäßig nicht sichtbar. Soll nur eine Berechnung gestartet werden, so ist *NewCalculation* zu verwenden.

Beispiel:

```
AConval.NewDialog(AConval.IndexOfCalculationKind("Dimensionierung")).Show
```

In diesem Beispiel wird ein neuer Dialog erzeugt und angezeigt.

In der ersten Zeile steht der Name des Elements. Dabei können auch mehrere Bezeichner mit einem „=“ getrennt auftauchen. In diesem Fall können die Elemente absolut synonym verwendet werden. In der auf die Überschrift folgenden Definition wird der Einfachheit halber aber nur ein Bezeichner verwendet.

In der zweiten Zeile steht die genaue Definition. Die Standardtypen in der Definition entnehmen Sie bitte – wie auch die entsprechende IDL- und VisualBasic-Syntax – der folgenden Tabelle:

Typenbezeichner in diesem Dokument	IDL	VisualBasic	Erklärung
String	BSTR	String	Zeichenfolge
Integer	Long	Long	Ganze Zahl (32-Bit)
Variant	VARIANT	Variant	Varianter Datentyp
Boolean	VARIANT_BOOL	Boolean	Wahrheitswert
Double	double	Double	Fließkommazahl
Char	unsigned char	Char	Einzelnes Zeichen

Handelt es sich um ein Attribut (eine Eigenschaft, *property*), so befindet sich am Ende der Zeile ein Vermerk auf die Verwendungsmöglichkeiten mit folgenden Bedeutungen:

(r/o)	Nur lesen
(r/w)	Lesen und schreiben
(w/o)	Nur schreiben

Steht vor einem Parameter `[inout]`, so handelt es sich um eine Parameterreferenz, d. h. um einen veränderbaren Parameter.

Der erste Parameter eines Events lautet immer dann *ASender*, wenn die Quelle des Ereignisses nicht das Objekt selber ist. Er verweist dann auf die Quelle des Ereignisses.

In der dritten Zeile folgt eine Beschreibung des Elements. Danach folgt optional noch ein Beispiel mit Erklärung.

Historie

Neuerungen in CONVAL 12.3:

- *IConval12* hat jetzt neue Funktionen für das Laden und Speichern von Profilen und erstellen und wiederherstellen von Backups.

Neuerungen in CONVAL 11.3:

- *IConval11* liefert jetzt ein Interface für die Ländereinstellungen. Über *IConval11.LanguageSettings* sind alle Einstellungen des Dialogs „Spracheinstellungen“ verfügbar.
- Über *IConval.Formulas* können die Formeln (neu in Conval 11.2) verwaltet werden. Diese können von benutzerdefinierten Daten benutzt werden. Damit gibt es die neuen Interfaces *IFormulas* und *IFormula*, sowie weitere Funktionen für Parameter von *IOptionalData*.

Neuerungen in CONVAL 11:

- Mit *IConval11.FindValve* / *FindValveEx* können Sie den Ventilauswahldialog für Stellventile verwenden.
- Für *ICalculation* und *IParameter* gibt es weitere Informationen über die Meldungen.
- Mit *ICalculationDialog.HasChanged* kann man nun abfragen, ob sich die Berechnung seite der letzten Speicherung geändert hat.
- In *ICalculation* können Drücke mit einem vorgegeben Referenzdruck gesetzt werden.
- Von *ICalculation* kann auf das entsprechende *ICVFile* Interface zugegriffen werden.
- Die Tabelle der Widerstandsstrukturen von Stellventilberechnungen kann ausgelesen werden.
- Für eine bessere Simulation der Oberfläche von CONVAL können weitere Eigenschaften eines Parameters ausgelesen werden (*IsCalculatedOverwritten*, *IsLookedUpOverwritten*, *DetailLevel*, *DisplayValue*, *VisualSwitchStateNames*)

Neuerungen in CONVAL 10.2:

- Mit dem Interface IConcentrations können nun auch Gas- und Flüssigkeitsgemische erstellt und bearbeitet werden. Dadurch ändert sich allerdings das Handling insofern, als ein Schema, das neu erstellt oder zur Bearbeitung geöffnet wird, auch wieder geschlossen werden muss!

Neuerungen von CONVAL 7.0 zu CONVAL 8.0.3:

- Die Berechnung der Kennlinien kann jetzt besser kontrolliert werden über das Interface ICVCharacteristics. Zu einem vorgegebenen Hub oder Durchfluss können die Parameter der Linealwerte ermittelt werden.
- Erdgaszusammensetzungen können über ICVConcentration verwaltet werden.

Neuerungen von CONVAL 6.0 zu CONVAL 7.0:

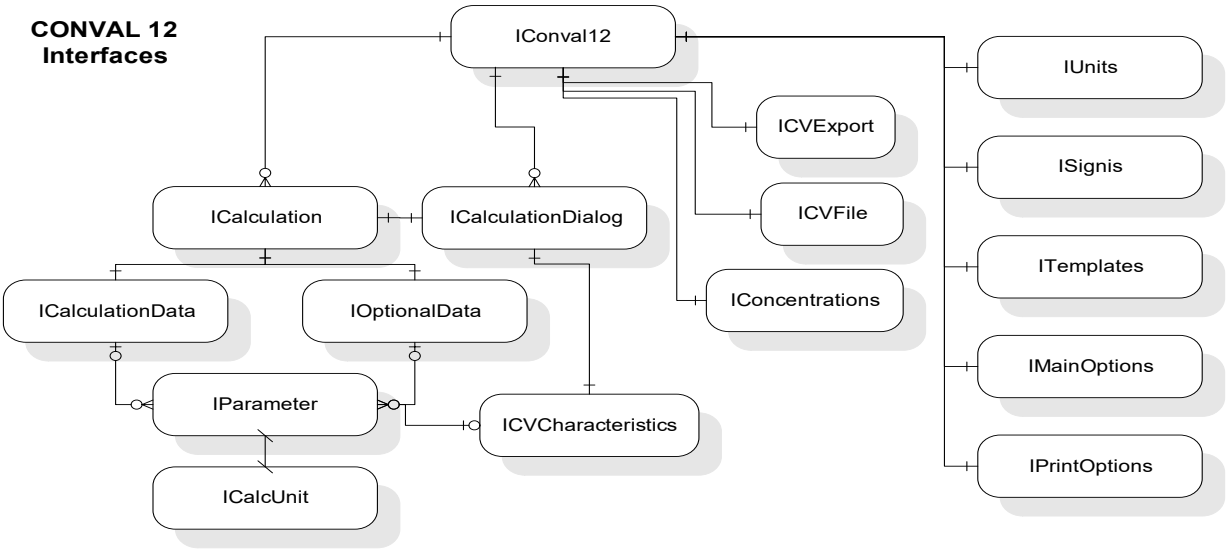
- Alle geladenen Berechnungs-Dateien werden in einer Liste geführt. Die Dateien *ICVFile* können auch geladen werden, ohne eine entsprechende Berechnung zu öffnen. Sie haben dann z. B. einen Lesezugriff auf alle Parameter und deren Werte.
- Der Export einer oder mehrerer Berechnungen kann über eine entsprechende Schnittstelle *ICVExport* generiert werden. Eine Liste der Exporte wird im Hauptinterface *IConval7* geführt.
- Der COM-Server verhält sich nun so, wie es die meisten anderen COM-Server auch tun und er wird geschlossen, wenn er nicht mehr benötigt wird. Bitte beachten Sie diesen Unterschied bei der Portierung Ihrer Anwendungen!

Neuerungen von CONVAL 5.0 zu CONVAL 6.0:

- Alle Operationen und Attribute, die mit Schemata / Vorlagen umgehen, sind verändert worden. Dies war notwendig, da es nun „private“ und „öffentliche“ Vorlagen gibt, die in unterschiedlichen Verzeichnissen gespeichert werden. Einige Attribute mussten in diesem Zusammenhang in Operationen umgewandelt werden, da Sie einige Parameter benötigen. Das sind im Einzelnen:
  - In *IUnits* / *ISignis* haben die Operationen *DeleteScheme*, *NewScheme*, *SaveScheme*, *LoadScheme* den neuen Parameter *IsPrivate*.
  - In *IUnits* / *ISignis* wurden die Attribute SchemeCount und SchemeNames durch Operationen ersetzt
  - In *IUnits* / *ISignis* hat sich die Eigenschaft DefaultScheme insofern geändert, als ein Postfix (:private oder :public) angegeben werden muss.
- Das Interface *ITemplates* ist hinzugekommen.

- Die Aufzählungen ENumPrivateMarker und ENumTempScope wurden hinzugefügt (Erläuterung s. u.). Diese werden für die erweiterten Bezeichnungen der Schemata bzw. Vorlagen verwendet, wenn der COM-Server eine solche zurückgibt.
- Der COM-Server wird jetzt nicht mehr als Fenster eingeblendet. Stattdessen ist er als Trayicon (neben der Windows-Uhr) sichtbar. Die Funktionen, die über das Kontextmenü aufgerufen werden konnten sind nun im Kontextmenü des Trayicons zu finden. Möchten Sie das Fenster wie in der Version 5.0 anzeigen, so verwenden Sie die Methode *IConval6.Show* oder den entsprechenden Eintrag aus dem Kontextmenü. Des Weiteren kann das Fenster über das Kontextmenü permanent in den Vordergrund gesetzt werden.
- Der COM-Server kann mit der Methode Exit geschlossen werden. Bitte beachten Sie aber, dass das ohne Rücksicht auf evt. noch geöffnete Referenzen aus der eigenen **oder einer anderen Anwendung** heraus geschieht!
- Verschiedene Optionen lassen sich nun über den COM-Server einstellen: *IMainOptions* und *IPrintOptions*. Zur Verwendung dieser neuen Schnittstellen lesen Sie bitte den entsprechenden Abschnitt weiter unten.

Objekthierarchie



Aufzählungstypen

ENumDetailLevel

Bezeichnet die möglichen Sichtbarkeiten von Parametern.

dlInvisible	Der Parameter ist immer unsichtbar.
dlHigh	Der Parameter ist nur in der „+"–Ansicht sichtbar, also wenn die Details eingeblendet werden.
dlLow	Der Parameter ist auch in der „-"–Ansicht sichtbar, also wenn die Details ausgeblendet werden.

ENumDisplayState

Bezeichnet die möglichen Anzeigemodi eines Parameters:

dsNormal	„normaler“, editierbarer Wert
dsDisabled	Wert ist nicht verfügbar
dsInvisible	Wert wird ausgeblendet
dsReadOnly	Wert kann nur gelesen werden
dsCutOff	Wert ist nicht mit einer Berechnung verbunden

ENumInputState

Bezeichnet die möglichen Eingabemodi eines Parameters:

isUser	Benutzereingabe
isDatabase	Wert stammt aus einer Datenbank (nachgeschlagener Wert)
isCalculated	berechneter Wert

ENumMessageType

Bezeichnet die verschiedenen Arten von Meldungen in CONVAL.

msgtypError	Ein Fehler ist aufgetreten, d.h. der Parameter kann nicht berechnet werden.
msgtypOutOfBounds	Der Wert liegt außerhalb der Grenzen für diese Einheit.
msgtypFatalWarning	Eine schwerwiegende Warnung, die unbedingt beachtet werden sollte.
msgtypMissingParameter	Der Parameter wurde nicht eingegeben, ist aber für die Berechnung relevant.
msgtypWarning	Eine Warnung ist aufgetreten. Das kann z.B. eine Verletzung der Vorschriften in einer Norm sein.
msgtypHint	Ein Hinweis zur Beachtung
msgtypCorrect	Die Werte erfüllen die Vorgaben (z.B. die Norm).

ENumMixType

Alle Typen von Stoffgemischen (wird in *IConcentration* verwendet)

mixtypAgaGerg	Stoffgemisch nach AGA8 oder GERG
mixtypGas	Gasgemisch
mixtypLiquid	Flüssigkeitsgemisch

ENumOptionalDataParamKind

Bezeichnet die unterschiedlichen Datentypen für die benutzerdefinierten Daten (s. u.):

odpkInteger	ganze Zahl
odpkFloat	Fließkommawert
odpkDate	Datum
odpkBoolean	Wahrheitswert
odpkString	Zeichenfolge
odpkMemo	mehrzeiliger, unformatierter Text
odpkParameter	physikalischer Wert
odpkLine	Trennlinie
odpkHeadLine	Überschrift

ENumParamType

Alle Parameter-Typen (wird in *IParameter* verwendet)

ptParameter	Mathematischer Parameter
ptString	Zeichenfolge
ptSwitch	Ja-Nein-Schalter
ptListSwitch	Schalter für mehrere Zustände; Auswahlliste
ptDataList	Schalter für mehrere Zustände, die aus der Datenbank gelesen werden; Auswahlliste

ENumPrivateMarker

Bezeichnet die Marker, die den Namen der Vorlagen hinzugefügt werden können:

pmInternational	Sprach**un**abhängiger Marker (:private oder :public)
pmLanguage	Sprach**ab**hängiger Marker für die Ausgabe in einem User-Interface. Dieser wird in runde Klammern gesetzt. Z. B.: (privat)
pmNone	Kein Marker, d. h. es kann dann nicht unterschieden werden, ob es sich um eine private oder eine öffentliche Vorlage handelt.

ENumTempScope

Bezeichnet die möglichen Filter bei der Anforderung von Vorlagen-Namen:

tsAll	Alle Vorlagen
tsPrivateOnly	Nur die privaten Vorlagen
tsPublicOnly	Nur die öffentlichen Vorlagen

ENumTemplateType

Alle Schema-Typen (wird in *ITemplates* verwendet)

ttUnit	Einheiten
ttSignis	Signifikante Stellen
ttHeader	Kopfzeilen
ttFooter	Fußzeilen
ttAdditionalData	Benutzerdefinierte Daten
ttFlowElementTable	Wertetabelle im Modul Wirkdruckgeber
ttSimpleParamsList	Parameterliste

ENumValueState

Bezeichnet die unterschiedlichen Zustände eines Zahlenwerts:

vsNormal	gültiger Wert
vsUndefined	undefinierter Wert
vsFaulty	fehlerhafter Wert
vsInfinity	unendlich (na)
vsMinusInfinity	minus unendlich (na)

ENumStartLanguageMode

Bezeichnet die Sprache, die beim Start von CONVAL verwendet werden soll:

slmUseWindowsLanguage	Die aktuelle Windows-Sprache verwenden. Wenn diese nicht unterstützt wird, wird Englisch verwendet.
slmUseLastConvalLanguage	Öffnet CONVAL in der Sprache mit der CONVAL zuletzt geschlossen wurde.
slmUseFixLanguage	Immer dieselbe Sprache beim Öffnen verwenden. Diese wird über *ILanguageOptions.StartLanguageNo* gesetzt.

Interfaces

IConval12

Hauptinterface, über welches die Berechnungen und deren Dialoge erzeugt werden können. Ferner führt es eine Liste der Masken und eine Liste der Berechnungen mit. Über dieses Interface können auch die voreingestellten Einheiten und signifikanten Stellen ausgelesen und manipuliert werden.

Tag

```
property Tag: Variant; (r/w)
```

Dient zur Speicherung beliebiger Informationen.

Show

```
procedure Show;
```

Macht das Hauptfenster des COM-Servers sichtbar. Beim Start des COM-Servers ist wird das Hauptfenster angezeigt.

Hide

```
procedure Hide;
```

Macht das Hauptfenster des COM-Servers unsichtbar. Beim Start des COM-Servers ist wird das Hauptfenster angezeigt.

Visible

```
property Visible: Boolean; (r/o)
```

Über diese Eigenschaft kann ermittelt und eingestellt werden, ob das Hauptfenster des COM-Servers sichtbar ist.

CalculationKindCount

property CalculationKindCount: Integer; (r/o)

Gibt die Anzahl der Berechnungsmodule zurück (Beispiel s. *CalculationKinds*).

CalculationKinds

property CalculationKinds[Index: Integer]: String; (r/o)

Gibt die Bezeichnung des mit *Index* indizierten Berechnungsmoduls zurück. Die Berechnungsmodule sind von 1 bis *CalculationKindCount* durchnummeriert.

Beispiel

```
for i = 1 to CalculationKindCount
  MyStringList.Add(AConval.CalculationKinds[i])
```

In diesem Beispiel werden alle Bezeichnungen der Berechnungsmodule an eine Liste von Zeichenfolgen angehängt.

Anmerkung

Die Bezeichnung eines Berechnungsmoduls ist sprachabhängig! D. h. die Eigenschaft *CalculationKinds* gibt einen anderen Wert zurück, wenn die Sprache umgestellt wurde.

IndexOfCalculationKind

function IndexOfCalculationKind(const Index: String): Integer;

Liefert den Index einer Bezeichnung eines Berechnungsmoduls.

NewDialog

function NewDialog(CalcKind: String): ICalculationDialog;

Erzeugt eine neue Instanz eines Dialogs der angegebenen Berechnung und gibt das *IDialog* Interface zurück. Der Dialog ist standardmäßig nicht sichtbar. Soll nur eine Berechnung gestartet werden, so ist *NewCalculation* zu verwenden.

Beispiel

```
AConval.NewDialog(AConval.IndexOfCalculationKind("Dimensionierung")).Show
```

In diesem Beispiel wird ein neuer Dialog erzeugt und angezeigt.

DialogCount

property DialogCount: Integer; (r/o)

Gibt die Anzahl der Dialoge zurück, die erzeugt worden sind (Beispiel s. *Dialogs*).

Dialogs

property Dialogs[Index: Integer]: ICalculationDialog; (r/o)

Gibt den mit *Index* indizierten Dialog als *IDialog* zurück. Die Dialoge sind von 1 bis *DialogCount* durchnummeriert.

Beispiel

```
for i = 1 to AConval.DialogCount
  AConval.Dialogs[i].Hide;
```

In diesem Beispiel werden alle Dialoge versteckt.

NewCalculation

function NewCalculation(CalcKind: Integer): ICalculation;

Erzeugt eine neue Instanz einer Berechnung (ohne Dialog) des angegebenen Typs und gibt diese als *ICalculation* Interface zurück. Dialoge und Berechnungen sind nicht in einander überführbar! D. h. einer Berechnung kann nicht im Nachhinein ein Dialog zugewiesen werden und umgekehrt kann eine Berechnung nicht von ihrem Dialog getrennt werden, wenn ein Dialog erzeugt worden ist.

Beispiel

```
MyCalculation =
    AConval.NewCalculation(AConval.IndexOfCalculationKind("Lochscheiben"))
```

Hier wird der Variablen *MyCalculation* ein neues Berechnungsinterface zugewiesen.

CalculationCount

property CalculationCount: Integer; (r/o)

Gibt die Anzahl der Berechnungen zurück, die über *NewCalculation* erzeugt worden sind (Beispiel s. *Calculations*).

Calculations

property Calculations[Index: Integer]: ICalculation; (r/o)

Gibt die mit *Index* indizierte Berechnung als *ICalculation* Interface zurück. Die Berechnungen sind von 1 bis *CalculationCount* durchnummeriert.

Beispiel

```
for i = 1 to AConval.CalculationCount
  AConval.Calculations[i].Clear
```

In diesem Beispiel werden alle Berechnungen zurückgesetzt.

Units

property Units: IUnits; (r/o)

Gibt ein Interface der Klasse *IUnits* zurück, mit welchem die Einheiten bearbeitet werden können.

Beispiel

```
TemperaturName = AConval.Units.QuantityNames[1]
```

Hier wird der Variablen *TemperaturName* der Name der ersten Größe aus der Liste aller Größen zugewiesen.

Signis

property Signis: ISignis; (r/o)

Gibt ein Interface der Klasse *ISignis* zurück, mit welchem die vorgegebene Anzahl der signifikanten Stellen für Werte einer Größe ausgelesen und eingestellt werden können.

Anmerkung

Im Einzelfall kann ein Parameter einer Berechnung sich über diese Regel hinwegsetzen und mehr oder weniger signifikante Stellen unabhängig von dieser Vorgabe anzeigen.

Templates

property Templates: ITemplates; (r/o)

Gibt ein Interface der Klasse *ITemplates* zurück, mit welchem Schemata und Vorlagen verwaltet werden können.

MainOptions

property MainOptions: IMainOptions; (r/o)

Dieses Attribut liefert ein Interface der Klasse *IMainOptions* zurück, mit dessen Hilfe sich die Programmoptionen einsehen und bearbeiten lassen.

PrintOptions

property PrintOptions: IPrintOptions; (r/o)

Dieses Attribut liefert ein Interface der Klasse *IPrintOptions* zurück, mit dessen Hilfe sich die Programmoptionen einsehen und bearbeiten lassen.

LanguageOptions

property LanguageOptions: ILanguageOptions; (r/o)

With this attribute you can access the settings of the language dialog.

IsCV7File

function IsCV7File(FileName: string): Boolean;

Ermittelt, ob eine Datei auf dem Datenträger mit dem neuen Dateiformat ab CONVAL 7 gespeichert wurde. Nur Dateien, die mit CONVAL 7 oder höher gespeichert wurden, haben die volle Funktionalität des *ICVFile* Interfaces. D. h. nur bei diesen Dateien kann auf die Parameter zugegriffen werden, ohne sie als voll funktionsfähige Berechnung zu laden. Weitere Informationen finden Sie unter *ICVFile*.

LoadCVFile

function LoadCVFile(FileName: string): ICVFile;

Mit dieser Routine laden Sie eine Datei in die Liste *CVFiles*. Weitere Informationen finden Sie unter *ICVFile*.

FileOfName

```
function FileOfName(FileName: string; ForceLoad: Boolean): ICVFile;
```

Mit dieser Funktion erhalten Sie das Interface der Datei *FileName*, sofern es sich schon im Speicher befindet. Ist die Datei noch nicht geladen, so gibt der Parameter *ForceLoad* an, ob die Datei geladen oder ein leerer Wert zurückgegeben werden soll. Weitere Informationen finden Sie unter *ICVFile*.

CVFileCount

```
property CVFileCount: Integer; (r/o)
```

Gibt die Anzahl der geöffneten Dateien zurück.

CVFiles

```
property CVFiles[Index: Integer]: ICVFile; (r/o)
```

Gibt die mit *Index* indizierte Datei als *ICVFile* zurück. Die Dateien sind von 1 bis *FileCount* durchnummeriert.

NewCVExport

```
function NewCVExport: ICVExport;
```

Mit dieser Funktion erzeugen Sie ein neues Export-Interface *ICVExport*. Ändern Sie anschließend die Parameter des Exports und führen ihn aus.

CVExportCount

```
property CVExportCount: Integer; (r/o)
```

Gibt die Anzahl der erzeugten Exporte zurück.

CVExports

```
property CVExports[Index: Integer]: ICVExport; (r/o)
```

Gibt den mit *Index* indizierten Export als *ICVExport* zurück. Die Exporte sind von 1 bis *ExportCount* durchnummeriert.

Concentrations

```
property Concentrations: IConcentrations; (r/o)
```

Gibt ein Interface *IConcentrations* zurück, das Zugriff auf die Erdgaskonzentrationen bzw. Gasgemische bietet. Mit Hilfe dieses Interfaces kann sowohl die Liste der verfügbaren Gemische ausgelesen werden als auch ein einzelnes bearbeitet werden.

LoadProfile

```
function LoadProfile(Ident: String; IsPrivate: Boolean): Boolean;
```

Lädt die Einstellungen aus einem bestehenden Profil. Wie bei allen Schemata mit einem Namen und entweder als privates oder öffentliches Profil. Der Rückabewert gibt an, ob die Einstellungen erfolgreich geladen werden konnten.

SaveProfile

```
function SaveProfile(Ident: String; IsPrivate: Boolean; AllowOverwrite:
Boolean): Boolean;
```

Speichert die aktuellen Einstellungen in einem Profil. Wie bei allen Schemata mit einem Namen und entweder als privates oder öffentliches Profil. Der Rückabewert gibt an, ob die Einstellungen erfolgreich gespeichert werden konnten.

CreateBackup

```
function CreateBackup(FileName: String; WantOverwrite: Boolean): Boolean;
```

Erstellt ein Backup der Profile, Einstellungen, Vorlagen und Schemata. *FileName* ist der vollständige Dateiname mit Pfad. *WantOverwrite* entscheidet, was gemacht werden soll, wenn schon eine Datei mit diesem Namen existiert. Im Erfolgsfall gibt die Funktion True zurück.

RestoreBackup

function RestoreBackup(FileName: String): Boolean;

Mit *RestoreBackup* werden Profile, Einstellungen, Vorlagen und Schemata aus der angegebenen Datei wiederhergestellt. Im Erfolgsfall gibt die Funktion True zurück.

WantEvents

property WantEvents: Boolean; (r/w)

Bestimmt, ob Events an den Client geleitet werden sollen. Werden keine Events benötigt, kann man über diese Eigenschaft in manchen Umgebungen (Early binding) eine bessere Performance erreichen.

MuteExceptions

property MuteExceptions: Boolean; (r/w)

Bestimmt, ob der COM-Server Fehlermeldungen in Dialogboxen ausgeben soll. Der Vorgabewert ist *True*.

ActiveLanguage

property ActiveLanguage: Integer; (r/w)

Gibt den Index der aktuell gesetzten Sprache zurück. Die Liste der Sprachen werden mit *LanguageNames* / *LanguageCount* ermittelt.

Anmerkung: Veraltet! Verwenden Sie *LanguageOptions*.

LanguageCount

property LanguageCount: Integer; (r/o)

Gibt die Anzahl der zur Verfügung stehenden Sprachen zurück.

Anmerkung: Veraltet! Verwenden Sie *LanguageOptions*.

LanguageNames

property LanguageNames[Index: Integer]: String; (r/o)

Gibt den Namen der mit *Index* indizierten Sprache zurück.

Anmerkung: Veraltet! Verwenden Sie *LanguageOptions*.

DecimalSeparator

property DecimalSeparator: Char; (r/w)

Dieses Attribut repräsentiert das aktuell zur Trennung der Vor- von den Nachkommastellen verwendete Zeichen. Im deutschsprachigen Raum ist ein Komma üblich.

Anmerkung: Veraltet! Verwenden Sie *LanguageOptions*.

Anmerkung Diesem Attribut kann nur der Wert , oder . zugewiesen werden. Alle anderen Zuweisungen werden ignoriert.

ThousandSeparator

property ThousandSeparator: Char; (r/w)

Dieses Attribut repräsentiert das Zeichen, welches zur besseren Lesbarkeit großer Zahlen die „Tausenderblöcke“ von einander trennt.

Anmerkung: Veraltet! Verwenden Sie *LanguageOptions*.

Anmerkung Diesem Attribut kann nur der Wert , oder . zugewiesen werden. Alle anderen Zuweisungen werden ignoriert.

ShortDateFormat

property ShortDateFormat: String; (r/w)

Dieses Attribut enthält das kurze Datumsformat.

Anmerkung: Veraltet! Verwenden Sie *LanguageOptions*.

LongDateFormat

property LongDateFormat: String; (r/w)

Dieses Attribut enthält das lange (ausführlichere) Datumsformat.

Anmerkung: Veraltet! Verwenden Sie *LanguageOptions*.

FindValve

```
function FindValve([inout] VManufacturer: string; [inout] VSeries: string;
    VSize, VSeat, VStroke: sring; VCv: double; VChar: Integer; [out]
    Valve: string; WantDialog, WantANSI: Boolean): Boolean;
```

Mit dieser Funktion rufen Sie den Stellventil-Auswahldialog auf. Dies funktioniert auch, wenn kein Stellventilmodul geöffnet ist. Die Funktion gibt True zurück, wenn ein Ventil ausgewählt wurde. Die Parameter haben folgende Bedeutungen:

- *VManufacturer* – Hersteller. Der Hersteller kann für einen Filter übergeben werden. Wird die Funktion erfolgreich beendet, so gibt dieser Parameter den ausgewählten Hersteller zurück.
- *VSeries* – Baureihe. Die Baureihe kann für einen Filter übergeben werden. Dabei kann das Zeichen „%“ als Platzhalter für kein, ein oder mehrere weitere Zeichen verwendet werden. Wird die Funktion erfolgreich beendet, so gibt dieser Parameter die ausgewählte Baureihe zurück.
- *VSize* – Die Ventilnennweite im Ausgang kann als Filter übergeben werden.
- *VSeat* – Der Sitzdurchmesser kann als Filter übergeben werden.
- *VStroke* – Der Hub kann als Filter übergeben werden.
- *VCv* – Der Nenndurchflusskoeffizient kann als Filter übergeben werden. Dabei steht die Filterung auf „exakt“, d.h. es werden nur Ventil gfunden, die genau diesen Durchflusskoeffizient haben.
- *VChar* – Es kann nach der Ventilkennlinie gefiltert werden:
    - *0* – Filter nicht gesetzt
    - *1* – gleichprozentig
    - *2* – linear
    - *3* – auf / zu
- *Valve* – Über diesen Parameter wird die Ventilbezeichnung in CONVAL zurückgegeben. Bitte beachten Sie, dass diese für dasselbe Ventil unterschiedlich sein kann, wenn sich die Sprache, die Ländereinstellung (Dezimalpunkt / -komma) oder die Wahl zwischen ANSI und DIN ändert. Auch kann sich bei einem Datenbank-Update der Name des Ventils ändern.
- *WantDialog* – Wenn die Suche mehr als einen Treffer hat, kann über diesen Parameter bestimmt werden, ob sich der Dialog öffnet, damit der Anwender dort ein Ventil auswählen kann. Die Funktion wird erst beendet, wenn der Dialog geschlossen wird.
- *WantANSI* – Mit diesem Parameter bestimmen Sie, ob ANSI (cv-Werte) oder DIN (kv-Werte) verwendet wird, falls beide in der Datenbank hinterlegt sind. Ist nur ein Wert für ein ventil hinterlegt, so hat der Parameter keinen Einfluss.

FindValveEx

```
function FindValveEx(VManufacturer, VSeries, VSize, VSeat, VStroke: sring;
    VCv: double; VChar: Integer; [out] Valve: string; [inout] Ref1, Ref2:
    string; WantDialog, WantANSI, ForceDialog: Boolean): Boolean;
```

Diese Funktion ist eine Erweiterung der Funktion FindValve. Alle Parameter haben dieselbe Bedeutung. Die zusätzlichen Parameter sind:

- *Ref1, Ref2* – Referenz-IDs für Ventile des Herstellers Fisher.
- *ForceDialog* – Der Dialog wird auch angezeigt, wenn es nur einen Treffer gibt.

Version

```
property Version: String; (r/o)
```

Dieses Attribut gibt die Hauptversionsnummer des COM-Servers zurück (z. B. „12.0“).

Build

```
property Build: String; (r/o)
```

Dieses Attribut gibt die Buildversionsnummer des COM-Servers zurück (z. B. „12.0.1.6“).

Edition

property Edition: String (r/o)

Dieses Attribut gibt die CONVAL-Edition zurück (z. B. „Gesamtedition“).

AlwaysInFront

property AlwaysInFront: Boolean (r/w)

Dieses Attribut gibt an, ob das Hauptfenster, sofern es eingeblendet ist, immer im Vordergrund dargestellt werden soll oder nicht.

Linebreak

property Linebreak: string (r/w)

Mit diesem Attribut steuern Sie, in welcher Form Zeilenumbrüche in Zeichenfolgen codiert werden. Insbes. können Sie auf diese Weise mehrzeilige Zeichenfolgen in Excel in eine einzelne Zelle schreiben. Gültige Werte sind:

„*10*“, „*13*“, „*1013*“, „*1310*“, „*Excel*“

Dabei stehen „*10*“ und „*13*“ für die entsprechenden Steuerzeichen. „*Excel*“ und „*10*“ sind synonym.

Instances

property Instances: Integer (r/o)

Gibt die Anzahl der COM-Server-Instanzen an, die geladen sind. Diese Eigenschaft ist nur für interne Zwecke vorgesehen.

StayAlive

property StayAlive: Boolean; (r/w)

Ein Grundkonzept von COM-Servern ist es, dass sie sich automatisch schließen, wenn sie nicht mehr verwendet werden. Wenn das Programm *COMConval12.exe* also automatisch gestartet wird, indem Sie ein Objekt der Klasse *IConval12* erzeugen, so wird das Programm auch automatisch wieder geschlossen, wenn Sie das Objekt wieder freigeben. Da dieses Verhalten nicht immer erwünscht ist, können Sie verhindern, dass sich der COM-Server schließt, indem Sie *StayAlive* = *True* setzen.

Anmerkung: Beachten Sie, dass ein COM-Server, der „von Hand“ (also nicht durch die Erzeugung eines Objektes) gestartet wurde, sich nicht selbständig schließt, auch wenn *StayAlive* = *False* ist.

Anmerkung: Diese Eigenschaft bezieht sich auf den COM-Server insgesamt und nicht nur auf die jeweilige Instanz. D. h. die Eigenschaft kann in jeder Instanz von *IConval12* ausgelesen und verändert werden.

RestartServerCalls

property RestartServerCalls: Integer; (r/w)

Diese Eigenschaft ist eine Erweiterung von *StayAlive* und steuert diese. Ist der Wert von *RestartServerCalls* = 0, so lässt sich das Verhalten über die Eigenschaft *StayAlive* steuern, hat *RestartServerCalls* einen positiven Wert, wird *StayAlive* auf *False* gesetzt, wenn der COM-Server mindestens so viele Instanzen erzeugt hat und alle wieder freigegeben wurden. Diese Eigenschaft wurde hinzugefügt, um nach langfristigen Belastungen des COM-Servers einen unproblematischen Neustart „im Hintergrund“ zu ermöglichen.

Exit

procedure Exit;

Diese Routine schließt den COM-Server.

Achtung!

Bitte beachten Sie, dass der COM-Server ohne Rücksicht auf evt. noch geöffnete Referenzen aus der eigenen **oder einer anderen Anwendung** heraus geschlossen wird! Sollten noch andere Anwendungen den COM-Server verwenden, so sind deren Referenzen anschließend ungültig. In ungünstigen Fällen kann dies zu einem Absturz der entsprechenden Anwendung führen.

Events

Exception

```
event Exception(Message: String);
```

Dieses Ereignis tritt ein, wenn ein nicht vorhergesehender Fehler im COM-Server entsteht. Der Parameter *Message* gibt dann Aufschluss über die Art des Fehlers.

BeforeCloseDialog

```
event BeforeCloseDialog(ASender: ICalculationDialog;
                        [inout] CanClose: Boolean);
```

Dieses Ereignis tritt ein, bevor ein Dialog geschlossen und aus dem Speicher entfernt wird. Über den Parameter *CanClose* kann das Schließen verhindert werden.

BeforeCloseCalculation

```
event BeforeCloseCalculation(ASender: ICalculation);
```

Dieses Ereignis tritt ein, bevor eine Berechnung geschlossen und aus dem Speicher entfernt wird.

BeforePrint

```
event BeforePrint(ASender: ICalculationDialog; [inout] CanPrint: Boolean);
```

Dieses Ereignis tritt ein, bevor aus einem Dialog heraus eine Berechnung gedruckt wird. Mittels des Parameters *CanPrint* kann der Ausdruck unterbunden werden.

AfterPrint

```
event AfterPrint(ASender: ICalculationDialog);
```

Dieses Ereignis tritt ein, nachdem aus einem Dialog heraus eine Berechnung gedruckt wurde.

BeforeParamChange

```
event BeforeParamChange(ASender: IParameter);
```

Dieses Ereignis tritt ein, bevor ein Parameter verändert wird.

AfterParamChange

```
event AfterParamChange(ASender: IParameter);
```

Dieses Ereignis tritt ein, nachdem ein Parameter verändert wurde.

BeforeOpen

```
event BeforeOpen(ASender: Variant;
                 [inout] FileName: String;
                 [inout] CanOpen: Boolean);
```

Dieses Ereignis tritt ein, bevor eine Berechnung geöffnet wird. Der Dateiname kann mit dem Parameter *FileName* verändert werden. Mit dem Parameter *CanOpen* kann verhindert werden, dass die Berechnung tatsächlich geöffnet wird.

AfterOpen

```
event AfterOpen(ASender: Variant; FileName: String);
```

Dieses Ereignis tritt ein, nachdem eine Berechnung geöffnet wurde. Über den Parameter *FileName* kann der Dateiname der Berechnung ermittelt werden.

BeforeSave

```
event BeforeSave(ASender: Variant;
                 [inout] FileName: String;
                 [inout] CanSave: Boolean);
```

Dieses Ereignis tritt ein, bevor eine Berechnung geschlossen wird. Der Dateiname kann mit dem Parameter *FileName* verändert werden. Mit dem Parameter *CanSave* kann verhindert werden, dass die Berechnung tatsächlich gespeichert wird.

AfterSave

```
event AfterSave(ASender: Variant; FileName: String);
```

Dieses Ereignis tritt ein, nachdem eine Berechnung gespeichert wurde. Über den Parameter *FileName* kann der Dateiname der Berechnung ermittelt werden.

ICalculationDialog

Dieses Interface kapselt einen Dialog, wie er in CONVAL® für jede Berechnung implementiert ist. Über die Methoden und Eigenschaften können die Funktionalitäten, die der Benutzer per Menü anwählen kann, auch über COM genutzt werden. Ferner kann auf die entsprechende Berechnung via *Calculation* zugegriffen werden.

Tag

```
property Tag: Variant; (r/w)
```

Dient zur Speicherung beliebiger Informationen.

Application

```
property Application: IConval12; (r/o)
```

Gibt das Anwendungsobjekt zurück.

Parent

```
property Parent: Variant; (r/o)
```

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist es identisch mit dem Anwendungsobjekt *Application.*

Show

```
procedure Show;
```

Zeigt den Dialog an.

Hide

```
procedure Hide;
```

Versteckt den Dialog.

Visible

```
property Visible: Boolean; (r/w)
```

Die Eigenschaft *Visible* gibt an, ob der Dialog sichtbar ist oder nicht. (Siehe auch: *Show* und *Hide*)

Left, Top, Width, Height

```
property Left: Integer; (r/w)
property Top: Integer; (r/w)
property Width: Integer; (r/w)
property Height: Integer; (r/w)
```

Diese Attribute dienen der Positionsbestimmung auf dem Bildschirm. Lesen Sie so die Position des Dialogs aus oder verschieben Sie ihn.

SetWindowPos

```
procedure SetWindowPos(Left, Top, Width, Height: Integer);
```

Setzen Sie mit dieser Routine die Position des Dialogs.

ExpandAll

```
procedure ExpandAll;
```

Alle Abschnitte ausklappen, d. h. in allen Abschnitten werden die Details angezeigt.

CollapsAll

```
procedure CollapsAll;
```

Alle Abschnitte einklappen, d. h. in allen Abschnitten werden die Details ausgeblendet.

WindowMaximize, WindowMinimize, WindowRestore

```
procedure WindowMaximize;
procedure WindowMinimize;
procedure WindowRestore;
```

Vergrößern Sie mit der Routine *WindowMaximize* das Fenster auf Bildschirmgröße, verkleinern Sie mit *WindowMinimize* und stellen Sie die vorherige Größe wieder her mit *WindowRestore*.

AlwaysInFront

```
property AlwaysInFront: Boolean (r/w)
```

Mit diesem Attribut können Sie den Dialog in den Vordergrund bringen. Beachten Sie aber, dass der Dialog erst wieder hinter anderen Fenstern verschwindet, wenn Sie *AlwaysInFront* auf *False* setzen.

Calculation = ActiveCalculation

```
property Calculation: ICalculation; (r/o)
```

Gibt ein Interface der Klasse *ICalculation* zurück. Dieses kapselt die eigentliche Berechnung.

Beispiel

```
MyModuleName = AConval.Dialogs[1].Calculation.ModuleName
```

Hier wird der Variablen *MyModuleName* der Name der Berechnung des ersten Dialogs zugewiesen.

Anmerkung

Bitte beachten Sie, dass keine Prüfung stattfindet, ob der Dialog überhaupt existiert. Wenn der Dialog nicht instanziiert worden ist, gibt der COM-Server einen „schweren Fehler“ zurück. Ob ein Dialog bereits geschlossen worden ist, kann mit der Eigenschaft *Assigned* abgefragt werden.

Characteristics

```
property Characteristics: ICVCharacteristics;
```

Gibt ein Interface ICVCharacteristics zurück. Dieses erlaubt den Zugriff auf die Linealwerte der Kennlinien einer Stellventilberechnung.

CalculationCount

```
property CalculationCount: Integer;
```

Gibt die Anzahl der in der Maske geöffneten Berechnungen zurück (Siehe auch: *Calculations*).

Calculations

```
property Calculations[Index: Integer]: String; (r/o)
```

Gibt die mit *Index* indizierte Berechnung als Zeichenfolge zurück. Die Berechnungen sind von 1 bis *CalculationCount* durchnummeriert. Um zu einer Berechnung zu wechseln und diese über *Calculation* zu verwenden, wird die Prozedur *SelectCalculation* verwendet.

SelectCalculation

```
procedure SelectCalculation(Ident: Variant);
```

Diese Prozedur wählt eine Berechnung aus der Liste der aktuell geöffneten Berechnungen aus. Als Parameter *Ident* kann dabei entweder der Index als ganze Zahl oder der Name als Zeichenfolge übergeben werden. Dabei gilt: Name = *Calculations*[Index].

Anmerkung

Es ist zu beachten, dass nur die aktuelle Berechnung *Calculation* (= *ActiveCalculation*) verändert werden kann. Alle anderen Berechnungen, die in der Maske geöffnet sind, müssen erst mit SelectCalculation ausgewählt werden, wenn beispielsweise Parameter ausgelesen oder verändert werden sollen.

CalculationKind

```
property CalculationKind: Integer; (r/o)
```

Diese Eigenschaft zeigt an, um welche Art Berechnung es sich bei dem Dialog handelt. Der Index entspricht dem im Interface *IConval12* verwendeten Index.

New = NewCalculation

```
procedure New;
```

Diese Pozedur erzeugt eine neue Berechnung. Der Aufruf dieser Prozedur hat denselben Effekt, wie die Menüauswahl [Datei | Neu] im Dialog.

NewFromTemplate

```
procedure NewFromTemplate(FileName: string; IsPrivate: Boolean);
```

Verwenden Sie diese Funktion, um eine neue Berechnung auf der Grundlage einer zuvor gespeicherten Vorlage zu erstellen.

Als Parameter muss der Name des Schemas angegeben werden sowie ein Wahrheitswert, der angibt, ob es sich um ein privates Schema handelt.

OpenDialog

```
procedure OpenDialog;
```

Der Aufruf dieser Prozedur hat denselben Effekt, wie die Menüauswahl [Datei | Öffnen...] im Dialog.

Open = OpenCalculation

```
procedure Open(FileName: String; WantRecover: Boolean);
```

Mit dieser Prozedur wird eine Berechnung mit dem Dateinamen *FileName* geöffnet. Wenn diese bereits geöffnet ist, kann mit dem Parameter *WantRecover* festgelegt werden, ob diese wiederhergestellt werden soll (d. h. die Änderungen seit der letzten Speicherung gehen verloren) oder ob sie unangetastet bleiben soll.

Save

```
procedure Save;
```

Der Aufruf dieser Prozedur hat denselben Effekt, wie die Menüauswahl [Datei | Speichern] im Dialog.

SaveAs

```
procedure SaveAs(FileName: String; WantOverwrite: Boolean);
```

Mit dieser Prozedur wird eine Berechnung unter dem Dateinamen *FileName* gespeichert. Mit dem Parameter *WantOverwrite* kann festgelegt werden, ob die Datei, wenn sie bereits existiert, überschrieben oder gar nicht gespeichert werden soll.

SaveAsTemplate

```
procedure SaveAsTemplate(FileName: string; IsPrivate: Boolean);
```

Mit dieser Funktion können Sie die aktuelle Berechnung als Vorlage speichern. Eine solche Vorlage kann beim Erstellen einer neuen Berechnung oder auch generell für neue Berechnungen verwendet werden. Die Voreinstellungen für neue Berechnungen werden über die Programmoptionen gesteuert.

Als Parameter muss der Name des Schemas angegeben werden sowie ein Wahrheitswert, der angibt, ob es sich um ein privates Schema handelt.

CloseCalculation

```
procedure CloseCalculation(WantOverwrite: Boolean; WantSave: Boolean);
```

Die aktuelle Berechnung wird mit dieser Prozedur geschlossen. Der Parameter *WantOverwrite* gibt an, ob eine evt. bereits vorhandene Datei mit demselben Namen überschrieben werden soll, der Parameter *WantSave* gibt an, ob die Berechnung vor dem Schließen gespeichert werden soll.

CloseAll

```
procedure CloseAll(WantOverwrite: Boolean; WantSave: Boolean);
```

Alle geöffneten Berechnungen werden mit dieser Prozedur geschlossen. Der Parameter *WantOverwrite* gibt an, ob eine evt. bereits vorhandene Datei mit demselben Namen überschrieben werden soll, der Parameter *WantSave* gibt an, ob die Berechnungen vor dem Schließen gespeichert werden sollen.

HasChanged

```
property HasChanged: Boolean; (r/o)
```

Diese Eigenschaft ist True, wenn Änderungen an der aktuellen Berechnung seit dem letzten Speichern vorgenommen wurden.

ImportCV4

```
procedure ImportCV4;
```

Der Aufruf dieser Prozedur hat denselben Effekt, wie die Menüauswahl [Datei | Importieren aus CONVAL 4.0] im Dialog.

PrintDialog

procedure PrintDialog;

Der Aufruf dieser Prozedur hat denselben Effekt, wie die Menüauswahl [Datei | Drucken...] im Dialog.

Print = PrintOut = PrintCalc

procedure Print(WantDialog: Boolean);

Mit dieser Prozedur wird die aktuelle Berechnung ausgedruckt. Der Parameter *WantDialog* gibt dabei an, ob Dialoge für Druckeinstellungen und Druckerauswahl angezeigt werden sollen oder nicht.

Preview

procedure Preview(WantDialog: Boolean);

Mit dieser Routine kann die Druckvorschau geöffnet werden. Der Parameter *WantDialog* gibt dabei an, ob der Dialog mit den Druckoptionen zuvor geöffnet werden soll.

CreatePDF

procedure CreatePDF(FileName: String)

Benutzen Sie diese Routine, um ein pdf-Dokument aus der aktuellen Berechnung zu erstellen. Der Parameter *FileName* gibt dabei den Namen des pdf-Dokuments an.

CalculationExport

procedure CalculationExport(Format: string);

Verwenden Sie diese Routine, wenn Sie eine Berechnung in ein anderes Format exportieren möchten. Zzt. werden die Formate „pdf“ für Adobe Acrobat Dateien und „xls“ für Excel-Dateien unterstützt.

SendMail

procedure SendMail(WantCalculation: Boolean);

Mit dieser Funktion kann eine Berechnung per Mail verschickt werden. Ist der Parameter *WantCalculation True*, so wird die Berechnung als Dateianhang geschickt, ansonsten wird die Berechnung wie im Ausdruck als pfd-Datei verschickt.

Beachten Sie aber, dass diese Funktion nur einen entsprechenden MAPI-Aufruf generiert. D. h. es öffnet sich ein Dialog, in dem mindestens noch der Empfänger angegeben werden muss.

Undo

procedure Undo;

Mit dieser Prozedur machen Sie den letzten Schritt rückgängig.

ModuleFunc

procedure ModuleFunc(FuncName: string);

Mit dieser Funktion können Funktionen aufgerufen werden, die nur ein bestimmten Modulen verwendet werden. Z. Zt. wird nur die Funktion *ShowGraphics* unterstützt, die z. B. die Kennlinien im Modul „Stellventile“ aufruft.

TreeVisible

property TreeVisible: Boolean; (r/w)

Über diese Eigenschaft kann gesteuert werden, ob die Baumübersicht (links im Dialog) sichtbar sein soll.

Beispiel

```
AConval.Dialogs[2].TreeVisible = False
```

Hier wird die Baumübersicht im zweiten Dialog ausgeblendet.

MessagesVisible

property MessagesVisible: Boolean; (r/w)

Über diese Eigenschaft kann gesteuert werden, ob die Meldungen (unten im Dialog) sichtbar sein sollen.

Beispiel

```
AConval.Dialogs[2].MessagesVisible = True
```

Hier werden die Meldungen im zweiten Dialog eingeblendet.

ToolbarVisible

property ToolbarVisible: Boolean; (r/w)

Über diese Eigenschaft kann gesteuert werden, ob die Schalterleiste (oben im Dialog) sichtbar sein soll.

Beispiel

```
AConval.Dialogs[2].ToolbarVisible = True
```

Hier wird die Schalterleiste im zweiten Dialog eingeblendet.

HelpVisible

property HelpVisible: Boolean; (r/w)

Über diese Eigenschaft kann gesteuert werden, ob die mitlaufende Hilfe rechts im Dialog sichtbar ist.

Assigned

property Assigned: Boolean; (r/o)

Diese Eigenschaft zeigt an, ob der Dialog noch existiert. Wenn der Benutzer den Dialog selber geschlossen hat, ist diese Eigenschaft *False.* Diese Eigenschaft sollte vor jeder Aktion abgefragt werden, um sicherzustellen, dass die Aktion auch ausgeführt werden kann.

Ist *Assigned False*, so ist außer *Assigned* nur noch *CalculationKind* verfügbar.

Close

procedure Close;

Diese Prozedur schließt den Dialog.

Events

BeforeClose

event BeforeClose([inout] CanClose: Boolean);

Dieses Ereignis tritt ein, bevor der Dialog geschlossen wird. Mit dem Parameter *CanClose* kann verhindert werden. dass der Dialog tatsächlich geschlossen wird.

BeforeVisibleChange

event BeforeVisibleChange;

Dieses Ereignis tritt ein, bevor der Dialog ein- oder ausgeblendet wird. Anhand der Eigenschaft *Visible* kann der aktuelle Zustand ermittelt werden.

AfterVisibleChange

event AfterVisibleChange;

Dieses Ereignis tritt ein, nachdem der Dialog ein- oder ausgeblendet wird. Anhand der Eigenschaft *Visible* kann der aktuelle Zustand ermittelt werden.

BeforePrint

event BeforePrint([inout] CanPrint: Boolean);

Dieses Ereignis tritt ein, bevor eine Berechnung ausgedruckt wird. Mit dem Parameter CanPrint kann das Drucken noch verhindert werden.

AfterPrint

event AfterPrint;

Dieses Ereignis tritt ein, nachdem eine Berechnung ausgedruckt wird.

BeforeOpen

event BeforeOpen([inout] FileName: String; [inout] CanOpen: Boolean);

Dieses Ereignis tritt ein, bevor eine Berechnung geöffnet wird. Der Dateiname kann mit dem Parameter *FileName* verändert werden. Mit dem Parameter *CanOpen* kann verhindert werden, dass die Berechnung tatsächlich geöffnet wird.

AfterOpen

```
event AfterOpen(FileName: String);
```

Dieses Ereignis tritt ein, nachdem eine Berechnung geöffnet wurde. Über den Parameter *FileName* kann der Dateiname der Berechnung ermittelt werden.

BeforeSave

```
event BeforeSave([inout] FileName: String; [inout] CanSave: Boolean);
```

Dieses Ereignis tritt ein, bevor eine Berechnung geschlossen wird. Der Dateiname kann mit dem Parameter *FileName* verändert werden. Mit dem Parameter *CanSave* kann verhindert werden, dass die Berechnung tatsächlich gespeichert wird.

AfterSave

```
event AfterSave(ASender: Variant; FileName: String);
```

Dieses Ereignis tritt ein, nachdem eine Berechnung gespeichert wurde. Über den Parameter *FileName* kann der Dateiname der Berechnung ermittelt werden.

ICalculation

ICalculation kapselt eine Berechnung. Hier kann man Berechnungen öffnen, speichern, via *CalculationData* auf Werte zugreifen ect.

Tag

```
property Tag: Variant; (r/w)
```

Dient zur Speicherung beliebiger Informationen.

Application

```
property Application: IConval12; (r/o)
```

Gibt das Anwendungsobjekt zurück.

Parent

```
property Parent: Variant; (r/o)
```

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall kann es vom Typ *IConval12* oder *ICalculationDialog* sein, je nachdem, ob es als Berechnung ohne Oberfläche erzeugt wurde (*IConval12.NewCalculation*) oder die aktuelle Berechnung eines Dialogs darstellt.

CalculationData

```
property CalculationData: ICalculationData; (r/o)
```

Gibt ein Interface der Klasse *ICalculationData* zurück. Dieses kapselt den Zugriff auf die Parameter.

Beispiel

```
AConval.Calculations[1].CalculationData.ParamByName["T1"].Text = 100
```

In diesem Beispiel wird dem Parameter *T1* der ersten Berechnung der Wert 100 zugewiesen.

Anmerkung

Bitte beachten Sie, dass auch hier (wie bei den Dialogen) keine Prüfung stattfindet, ob die Berechnung überhaupt existiert. Wenn die Berechnung nicht instanziiert worden ist, gibt der COM-Server einen „schweren Fehler“ zurück.

OptionalData

```
property OptionalData: IOptionalData; (r/o)
```

Gibt ein Interface der Klasse *IOptionalData* zurück. Dieses kapselt den Zugriff auf die benutzerdefinierten Daten einer Berechnung.

Beispiel

```
MyOptionalData = AConval.NewCalculation("Lochscheiben").OptionalData
MyOptionalData.AddParam("Firma", "Firma:", odpkString)
MyOptionalData.Params[MyOptionalData.ParamCount].Text =
    "F.I.R.S.T. GmbH"
```

Data

property Data: Variant; (r/w)

Diese Eigenschaft enthält die gesamte Berechnung in binärer Form. So kann eine Berechnung ohne temporäre Dateien zwischengespeichert werden.

CalculationKind

property CalculationKind: Integer; (r/o)

Diese Eigenschaft zeigt an, um welche Art Berechnung es sich handelt. Der Index entspricht dem im Interface *IConval12* verwendeten Index.

Clear

procedure Clear;

Setzt alle Inhalte der Parameter der Berechnung zurück. D. h. Zahlenwerte werden geleert und Schalter werden auf ihren ursprünglichen Wert gesetzt.

New

procedure New;

Setzt die Berechnung so zurück, als sei sie neu geöffnet worden. Im Unterschied zu *Clear* werden auch die benutzerdefinierten Daten auf das Standardschema zurückgesetzt. Dies macht sich nur dann bemerkbar, wenn die Felddefinitionen der benutzerdefinierten Daten geändert wurden.

Open

function Open(FileName: String): Boolean;

Öffnet eine Berechnung mit dem Dateinamen *FileName*. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

Beispiel

```
MyDialog = AConval.NewDialog("Lochscheiben")
if MyDialog.Calculation.Open("C:\cv\Meine_Berechnung.COP") then
  MyRho = MyDialog.Calculation.CalculationData.ParamByName["Rho1"]
else
  ShowMessage("Die Berechnung konnte nicht geöffnet werden!")
```

In diesem Beispiel wird ein neuer Dialog erstellt, eine Berechnung geladen und – wenn diese erfolgreich geladen wurde – der Wert von *Rho1* der Variablen *MyRho* zugewiesen.

Save

function Save: Boolean;

Die Berechnung wird unter dem aktuellen Name (*FileName*) gespeichert. Wenn *FileName* leer ist oder die Berechnung aus anderen Gründen nicht gespeichert werden kann, gibt die Funktion *False* zurück.

Beispiel

```
MyCalculation = AConval.NewDialog("Lochscheiben").Calculation
...
MyCalculation.FileName = "C:\User\Meine_Berechnung."
MyCalculation.Save
```

In diesem Beispiel wird eine neue Berechnung erzeugt und gespeichert.

SaveAs

function SaveAs(FileName: String): Boolean;

Speichert die Berechnung unter dem Dateinamen *FileName*. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

Beispiel

```
MyCalculation = AConval.NewCalculation("Lochscheiben")

...
if MyCalculation.SaveAs("C:\User\Meine_Berechnung.COP") then
  ShowMessage("Die Berechnung wurde erfolgreich gesichert")
else
  ShowMessage("Fehler beim Sichern der Berechnung")
```

In diesem Beispiel wird eine neue Berechnung erzeugt und diese gespeichert. Danach wird eine entsprechende Meldung ausgegeben.

BeginUpdate

procedure BeginUpdate;

Leitet die Zuweisung einer Reihe von Werten ein. So wird verhindert, dass die Berechnung nach jeder Zuweisung neu durchgeführt wird.

Beispiel

```
MyCalculation = AConval.NewCalculation("Lochscheiben")
MyCalculation.BeginUpdate
MyCalculation.CalculationData.ParamByName["T1"].Value = 100
MyCalculation.CalculationData.ParamByName["P1"].Value = 10
MyCalculation.CalculationData.ParamByName["FuidName"].Text = "Wasser"
MyCalculation.EndUpdate
MyRho1 = MyCalculation.CalculationData.ParamByName["Rho1"].Value
```

In diesem Beispiel werden die drei Parameter *T1*, *P1* und *FuidName* gesetzt und die Dichte der Variablen *MyRho1* zugewiesen. Dabei wird der Vorgang durch den Aufruf von *BeginUpdate* und *EndUpdate* beschleunigt.

Achtung!

Verwenden Sie die Methode *BeginUpdate* mit Bedacht, da zwischen einem *BeginUpdate* und *EndUpdate* keinerlei Auswertung stattfindet. Insbesondere werden keine Schreibrechte für Parameter geändert. Wenn z. B. durch das Umschalten eines Schalters ein anderer Parameter sichtbar wird als vorher, so kann dieser dennoch nicht überschrieben werden, solange keine Auswertung stattfindet.

EndUpdate

procedure EndUpdate;

Beendet die Zuweisung einer Reihe von Werten. *EndUpdate* ist das Gegenstück zu *BeginUpdate.*

Anmerkung

Wenn vergessen wird, *EndUpdate* nach einem *BeginUpdate* aufzurufen, wird keine Berechnung mehr durchgeführt!

MessageCount

property MessageCount: Integer; (r/o)

Gibt die Anzahl der Meldungen zurück, die über *Messages* abgefragt werden können.

Messages

property Messages[Index: Integer]: String; (r/o)

Gibt die Meldung zurück, die mit *Index* indiziert ist. Die Meldungen sind von 1 bis *MessageCount* durchnummeriert.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculation = AConval.Calculations[AConval.CalculationCount]
for i = 1 to MyCalculation.MessageCount
  MyStringList.Add(MyCalculation.Messages[i])
```

In diesem Beispiel werden alle Meldungen einer Berechnung an eine Liste von Zeichenfolgen angehängt.

MessageClasses

property MessageClasses(Index: Integer): ENumMessageType; (R/O)

Gibt die Klasse der Meldung zurück. Der Index darf ein Wert von 1 bis MessageCount sein.

MessageID

property MessageID(Index: Integer): Integer; (R/O)

Gibt die ID der Meldung zurück. Der Index darf ein Wert von 1 bis MessageCount sein. Die ID kann verwendet werden um herauszufinden, ob eine bestimmte Meldung aufgetreten ist. Der Text der Meldung kann z.B. wegen der Zahlenwerte im Text variieren.

MessageParam

property MessageParam(Index: Integer): string; (R/O)

Gibt den Parameter der Meldung zurück. Der Index darf ein Wert von 1 bis MessageCount sein.

HasMessage

function HasMessage(ID: Integer): Boolean;

Püft, ob eine Meldung mit einer bestimmten ID aufgetreten ist. Die ID ist dieselbe wie MessageID.

FileExtension

property FileExtension: String; (r/o)

Gibt die Dateinamenserweiterung für die Berechnung zurück. Jede Art von Berechnung hat eine eigene Dateinamenserweiterung.

ModuleName

property ModuleName: String; (r/o)

Gibt den Namen der Berechnung zurück. Dieser ist gleich der Bezeichnung im Hauptprogramm und in der Titelleiste des entsprechenden Dialogs.

CV4FileName

property CV4FileName: String; (r/o)

Gibt den Dateinamen der entsprechenden CONVAL 4 Berechnung zurück.

Calculate

procedure Calculate;

Forciert die Berechnung. Mit dieser Anweisung kann die Berechnung „von Hand“ gestartet werden. (z. B. nach einem *BeginUpdate* oder nachdem *Active* auf *True* gesetzt wurde.)

Anmerkung

Normalerweise wird die Berechnung automatisch durchgeführt, d. h. nach jeder relevanten Änderung sind auch die berechneten Werte aktuell.

Active

property Active: Boolean; (r/w)

Mit der Eigenschaft *Active* kann die automatische Berechnung ein- und ausgeschaltet werden.

IsCalculated

property IsCalculated: Boolean; (r/o)

Zeigt an, ob die Berechnung mit den aktuellen Eingabewerten ausgeführt wurde.

FileName

property FileName: String; (r/w)

Die Eigenschaft *FileName* bezeichnet den aktuellen Dateinamen der Berechnung. Dieser wird bei einem Aufruf von *Save* verwendet und von *Open* und *SaveAs* neu gesetzt.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculation = AConval.Calculations[AConval.CalculationCount]
...
if MyCalculation.FileName = "" then
  MyCalculation.FileName = "C:\User\Meine_Berechnung.COP"
MyCalculation.Save
```

In diesem Beispiel wird eine neue Berechnung erzeugt und – wenn der aktuelle Dateiname leer ist – unter „C:\User\Meine_Berechnung.COP“ gespeichert.

GetResistors

function GetResistors: String;

Diese Funktion gibt die Tabelle der Widerstandsstrukturen zurück. Das Ergebnis dieser Funktion ist ein csv-formatierter, Tab-getrennter String mit den folgenden Spalten: t1, p1, p2, kv, dRes, rho1, rho2, La, Ma, DNX, nARes, nBores, nDNres.

GetRefPressure

function GetRefPressure(CalcUnit: Variant): Double;

Gibt den Referenzdruck zurück, der für die Funktionen Get/SetPressueValue verwendet wird. Die Einheit kann als Text oder Index übergeben werden.

SetRefPressure

```
procedure SetRefPressure(CalcUnit: Variant; Value: Double); (r/w)
```

Setzt den Referenzdruck, der für die Funktion Get/SetPressueValue verwendet wird. Die Einheit kann als Text oder Index übergeben werden.

GetPressureValue

```
function GetPressureValue(ParamName: string; ACalcUnit: Variant; IsGage:
    Boolean): Double;
```

Liest den Wert eines Druck-Parameters unter Berücksichtigung des Referenzdrucks aus.

SetPressureValue

```
procedure SetPressureValue(ParamName: string; Value: Double; ACalcUnit:
    Variant; IsGage: Boolean);
```

Setzt den Wert eines Druck-Parameters unter Berücksichtigung des Referenzdrucks.

ModuleFunc

```
procedure ModuleFunc(FuncName: string);
```

Eine Funktion über den Namen aufrufen. Die Funktionen sind immer parameterlos. Abhängig vom Modul sind verschiedene Funktionen verfügbar:

Modul	Funktion	Beschreibung
Stellventil	DoNResCalculation	Berechnet die optimale Zahl von Widerständen
Lochscheiben	DoNResCalculation	Berechnet die optimale Zahl von Widerständen

ModuleFuncList

```
function ModuleFuncList(): string;
```

Gibt eine Liste der in dem aktuellen Modul zur Verfügung stehenden Funktionen. Die Namen der Funktionen sind durch einen Zeilenumbruch getrennt. Diese Liste kann je nach Modul auch leer sein.

CVFile

```
property CVFile: ICVFile;
```

Gibt ein CVFile zur aktuellen Berechnung zurück. Über dieses Format können ein paar zusätzliche Informationen über die Berechnung ausgelesen werden.

LoadUnitScheme

```
function LoadUnitScheme(const FileName: String;
                        const IsPrivate: Boolean): Boolean;
```

Mit Hilfe dieser Funktion kann man ein Einheitenschema für die aktuelle Berechnung laden. D. h. es werden alle Einheiten der betreffenden Berechnung nach den Vorgaben des Schemas gesetzt.

Als Parameter muss der Name des Schemas angegeben werden sowie ein Wahrheitswert, der angibt, ob es sich um ein privates Schema handelt.

LoadSigniScheme

```
function LoadSigniScheme(const FileName: String;
                         const IsPrivate: Boolean): Boolean;
```

Mit Hilfe dieser Funktion kann man ein Schema für die signifikanten Stellen für die aktuelle Berechnung laden. D. h. es werden alle signifikanten Stellen der betreffenden Berechnung nach den Vorgaben des Schemas gesetzt. Die Anzahl der signifikanten Stellen hat Einfluss auf die Anzeige im Dialog sowie auf die Eigenschaft *DisplayText*.

Als Parameter muss der Name des Schemas angegeben werden sowie ein Wahrheitswert, der angibt, ob es sich um ein privates Schema handelt.

Assigned

```
property Assigned: Boolean; (r/o)
```

Diese Eigenschaft zeigt an, ob die Berechnung noch existiert. Diese Eigenschaft sollte vor jeder Aktion abgefragt werden, um sicherzustellen, dass die Aktion auch ausgeführt werden kann.

Ist *Assigned False*, so ist außer *Assigned* nur noch *CalculationKind* verfügbar.

Close

```
procedure Close;
```

Diese Prozedur schließt die Berechnung und gibt den entsprechenden Speicherplatz wieder frei.

Events

BeforeClose

```
event BeforeClose(ASender: ICalculation);
```

Dieses Ereignis tritt ein, bevor eine Berechnung geschlossen und aus dem Speicher entfernt wird.

BeforeClear

```
event BeforeClear([inout] CanClear: Boolean);
```

Dieses Ereignis tritt ein, bevor die Werte einer Berechnung zurückgesetzt werden. (Siehe auch: *Clear*)

BeforeParamChange

```
event BeforeParamChange(ASender: IParameter);
```

Dieses Ereignis tritt ein, bevor ein Parameter verändert wird.

AfterParamChange

```
event AfterParamChange(ASender: IParameter);
```

Dieses Ereignis tritt ein, nachdem ein Parameter verändert wurde.

BeforeOpen

```
event BeforeOpen([inout] FileName: String; [inout] CanOpen: Boolean);
```

Dieses Ereignis tritt ein, bevor eine Berechnung geöffnet wird. Der Dateiname kann mit dem Parameter *FileName* verändert werden. Mit dem Parameter *CanOpen* kann verhindert werden, dass die Berechnung tatsächlich geöffnet wird.

AfterOpen

```
event AfterOpen(FileName: String);
```

Dieses Ereignis tritt ein, nachdem eine Berechnung geöffnet wurde. Über den Parameter *FileName* kann der Dateiname der Berechnung ermittelt werden.

BeforeSave

```
event BeforeSave([inout] FileName: String; [inout] CanSave: Boolean);
```

Dieses Ereignis tritt ein, bevor eine Berechnung geschlossen wird. Der Dateiname kann mit dem Parameter *FileName* verändert werden. Mit dem Parameter *CanSave* kann verhindert werden, dass die Berechnung tatsächlich gespeichert wird.

AfterSave

```
event AfterSave(FileName: String);
```

Dieses Ereignis tritt ein, nachdem eine Berechnung gespeichert wurde. Über den Parameter *FileName* kann der Dateiname der Berechnung ermittelt werden.

BeforeCalculate

```
event BeforeCalculate;
```

Dieses Ereignis tritt ein, bevor die Berechnung neu angestoßen wird. Dies passiert in der Regel immer dann, wenn ein Parameter geändert wird.

AfterCalculate

```
event AfterCalculate;
```

Dieses Ereignis tri tritt ein, nachdem die Berechnung neu angestoßen worden ist. Dies passiert in der Regel immer dann, wenn ein Parameter geändert wurde.

NoteChange

event NoteChange(ASender: IParameter);

Dieses Ereignis tritt ein, wenn die Notiz eines Parameters geändert wird.

ICalculationData

Das Interface *ICalculationData* kapselt den Zugriff auf Parameter. Im Gegensatz zu *IOptionalData* sind diese Parameter alle festen Felder einer Berechnung, d. h. alle Parameter, die in die Berechnung eingehen, sowie die Felder aus dem Berechnungskopf.

Tag

property Tag: Variant; (r/w)

Dient zur Speicherung beliebiger Informationen.

Application

property Application: IConval12; (r/o)

Gibt das Anwendungsobjekt zurück.

Parent

property Parent: Variant; (r/o)

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist dies die Berechnung *ICalculation*.

ParamCount

property ParamCount: Integer; (r/o)

Gibt die Anzahl der Parameter der Berechnung an (Beispiel s. *Params*).

Params

property Params[Index: Integer]: IParameter; (r/o)

Gibt den mit *Index* indizierten Parameter als *IParameter* Interface zurück. Die Parameter sind von 1 bis *ParamCount* durchnummeriert.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculation = AConval.Calculations[AConval.CalculationCount]
for i = 1 to MyCalculation.ParamCount
  MyStringList.Add(MyCalculation.CalculationData.Params[i].Name)
```

In diesem Beispiel wird eine neue Berechnung erzeugt und die Namen aller Parameter der Berechnung der Liste MyStringList hinzugefügt.

ParamByName

property ParamByName[Name: String]: IParameter; (r/o)

Gibt den Parameter mit dem Namen *Name* als *IParameter* zurück.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculation = AConval.Calculations[AConval.CalculationCount]
MyCalculation.CalculationData.ParamByName("T1").Value = 100
```

In diesem Beispiel wird eine neue Berechnung erzeugt und der Parameter *T1* auf den Wert 100 gesetzt.

IndexOfParam

property IndexOfParam[Name: String]: Integer; (r/o)

Gibt den Index eines Parameters zurück. Wenn der Parameter nicht existiert, so ist der Rückgabewert –1.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculation = AConval.Calculations[AConval.CalculationCount]
if MyCalculation.CalculationData.IndexOfParam("T1") <> -1 then
  MyCalculation.CalculationData.ParamByName("T1").Value = 100
```

In diesem Beispiel wird geprüft, ob die Lochscheiben-Berechnung den Parameter *T1* kennt, und dieser wird ggf. auf den Wert 100 gesetzt.

ParamList

property ParamList: string;

Gibt die komplette Liste der Parameter zurück. Diese werden durch Zeilenumbrüche von einander getrennt.

WantOrderedList

property WantOrderedList: Boolean; (r/w)

Bestimmt, ob die Liste der Parameter alphanumerisch sortiert sein soll. Dies ist die Voreinstellung. Die unsortierte Liste gibt die Reihenfolge wieder, in der die Parameter gesetzt werden sollten, um Wechselwirkungen zwischen den Parametern zu vermeiden. So sollten z. B. Schalter i. A. vor numerischen Werten gesetzt werden, da letztere ansonsten u. U. blockiert sind und noch gar nicht gesetzt werden können. (s. a. Bemerkung zu ICalculation.BeginUpdate)

IOptionalData

Das Interface *IOptionalData* kapselt die benutzerdefinierten Daten einer Berechnung. Im Gegensatz zu *ICalculationData* handelt es sich bei diesen Daten um Felder der Berechnung, die nicht in die Berechnung eingehen und somit nur zusätzliche Informationen bieten.

Tag

property Tag: Variant; (r/w)

Dient zur Speicherung beliebiger Informationen.

Application

property Application: IConval12; (r/o)

Gibt das Anwendungsobjekt zurück.

Parent

property Parent: Variant; (r/o)

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist dies die Berechnung *ICalculation*.

ParamCount

property ParamCount: Integer; (r/o)

Diese Eigenschaft gibt die Anzahl der Parameter der benutzerdefinierten Daten an. (Beispiel s. *Params*)

Params

property Params[Index: Integer]: IParameter; (r/o)

Gibt den mit *Index* indizierten benutzerdefinierten Parameter als *IParameter* Interface zurück. Die Parameter sind von 1 bis *ParamCount* durchnummeriert.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyOptionalData =
AConval.Calculations[AConval.CalculationCount].OptionalData
MyOptionalData.AddParam("Firma", "Firma:", odpkString)
MyOptionalData.Params[MyOptionalData.ParamCount].Text =
    "F.I.R.S.T. GmbH"
```

In diesem Beispiel werden eine neue Berechnung und ein neuer benutzerdefinierter Parameter erzeugt. Diesem wird dann die Zeichenfolge „F.I.R.S.T. GmbH“ zugewiesen.

ParamByName

property ParamByName[Name: String]: IParameter; (r/o)

Gibt den Parameter mit dem Namen *Name* als *IParameter* zurück.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyOptionalData =
AConval.Calculations[AConval.CalculationCount].OptionalData
MyOptionalData.AddParam("Firma", "Firma:", odpkString)
MyOptionalData.ParamByName["Firma"].Text = "F.I.R.S.T. GmbH"
```

In diesem Beispiel werden eine neue Berechnung und ein neuer benutzerdefinierter Parameter erzeugt. Diesem wird dann die Zeichenfolge „F.I.R.S.T. GmbH“ zugewiesen.

IndexOfParam

property IndexOfParam[Name: String]: Integer; (r/o)

Gibt den Index eines Parameters zurück. Wenn der Parameter nicht existiert, so ist der Rückgabewert –1.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculation = AConval.Calculations[AConval.CalculationCount]
if MyCalculation.OptionalData.IndexOfParam("Firma") <> -1 then
  MyCalculation.OptionalData.ParamByName("Firma").Text = "F.I.R.S.T. GmbH"
```

In diesem Beispiel wird geprüft, ob die Lochscheiben-Berechnung den Parameter *Firma* kennt, und dieser wird ggf. auf den Wert „F.I.R.S.T. GmbH“ gesetzt.

ClearParam

function ClearParam(Index: Integer): Boolean;

Setzt das durch *Index* indizierte benutzerdefinierte Feld zurück. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

DeleteParam

function DeleteParam(Index: Integer): Boolean;

Löscht das durch *Index* indizierte benutzerdefinierte Feld. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

AddParam

function AddParam(Name, Caption: String;
ParamType: ENumOptionalDataParamKind): Boolean;

Fügt ein benutzerdefiniertes Feld hinzu. Dabei ist *Name* der Wert, über den mit *ParamByName* indiziert wird, *Caption* die Beschriftung des entsprechenden Eingabefeldes im Dialog und *ParamType* der Typ des neuen Parameters (s. o.).

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyOptionalData =
AConval.Calculations[AConval.CalculationCount].OptionalData
MyOptionalData.AddParam("Firma", "Firma:", odpkString)
```

In diesem Beispiel wird eine neue Berechnung und ein neuer benutzerdefinierter Parameter erzeugt, der die Firma beinhalten soll.

ChangeParam

function ChangeParam(Index: Integer; Name, Caption: String;
ParamType: ENumOptionalDataParamKind): Boolean;

Ändert die Felddefinition eines Parameters.

Anmerkung Bitte beachten Sie, dass mit dieser Änderung der Inhalt des Feldes verloren geht!

MoveParam

function MoveParam(SourceIndex, DestIndex: Integer): Boolean;

Verschiebt ein benutzerdefiniertes Feld in der Liste der Felder.

ClearAll

function ClearAll: Boolean;

Setzt alle Inhalte der benutzerdefinierten Felder zurück. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

DeleteAll

```
function DeleteAll: Boolean;
```

Löscht alle benutzerdefinierten Felder. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

ParamList

```
property ParamList: string;
```

Gibt die komplette Liste der Parameter zurück. Diese werden durch Zeilenumbrüche von einander getrennt.

Template-Handling

Als Parameter muss jeweils der Name des Templates angegeben werden sowie ein Wahrheitswert, der angibt, ob es sich um ein privates Templates handelt.

AddTemplate

```
function AddTemplate(TemplateName: String; IsPrivate: Boolean): Boolean;
```

Diese Funktion fügt ein zuvor gesichertes Schema zu den benutzerdefinierten Daten hinzu. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

DeleteTemplate

```
procedure DeleteTemplate(FileName: String; IsPrivate: Boolean);
```

Diese Prozedur löscht ein zuvor gesichertes Schema vom Datenträger. Ist das Schema nicht vorhanden, geschieht nichts.

SaveAs

```
procedure SaveAs(FileName: String; IsPrivate: Boolean);
```

Diese Funktion speichert die aktuelle Definition der benutzerdefinierten Daten

ICVCharacteristics

Das Interface *ICVCharacteristics* kapselt den Zugriff auf die Linealwerte einer Kennliniengrafik. Im Gegensatz zu *ICalculationData* sind die Parameter abhängig vom Hub *h*. Der Hub wird sowohl mit der Methode *SetStroke* als auch mit *ParamOfStroke* gesetzt.

Tag

```
property Tag: Variant; (r/w)
```

Dient zur Speicherung beliebiger Informationen.

Application

```
property Application: IConval12; (r/o)
```

Gibt das Anwendungsobjekt zurück.

Parent

```
property Parent: Variant; (r/o)
```

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist dies der Berechnungsdialog *ICalculationDialog*.

ParamCount

```
property ParamCount: Integer; (r/o)
```

Gibt die Anzahl der Parameter der Linealwerte an.

Params

```
property Params[Index: Integer]: IParameter; (r/o)
```

Gibt den mit *Index* indizierten Parameter als *IParameter* Interface zurück. Die Parameter sind von 1 bis *ParamCount* durchnummeriert.

ParamByName

property ParamByName[Name: String]: IParameter; (r/o)

Gibt den Parameter mit dem Namen *Name* als *IParameter* zurück.

IndexOfParam

property IndexOfParam[Name: String]: Integer; (r/o)

Gibt den Index eines Parameters zurück. Wenn der Parameter nicht existiert, so ist der Rückgabewert –1.

ParamList

property ParamList: string; (r/o)

Gibt die komplette Liste der Parameternamen zurück. Diese werden durch Zeilenumbrüche voneinander getrennt.

Refresh

procedure Refresh;

Führt die Berechnung der Kennlinien erneut aus, wenn sich die Ventil-Berechnung seit der letzten Berechnung der Kennlinien geändert hat.

SetStroke

procedure SetStroke(Stroke: Double);

Setzt den Hub h auf den angegebenen Wert. Die Parameter des Characteristics-Interfaces (Linealwerte) werden entsprechend neu berechnet.

ParamOfStroke

function ParamOfStroke(Stroke: Double; ParamName: String): Double;

Setzt den Hub h auf den angegebenen Wert. Die Parameter des Characteristics-Interfaces (Linealwerte) werden entsprechend neu berechnet und der Wert des Parameters *ParamName* wird zurückgegeben.

StrokeOfQm

function StrokeOfQm(Qm: Double): Double;

Gibt den Hub für einen bestimmten Massendurchfluss zurück. Der Wert liegt immer zwischen 5 und 100% (jeweils einschließlich). Bitte beachten Sie, dass die Einheit von qm immer der von qm in der jeweiligen Berechnung entspricht.

ParamOfQ

function ParamOfQ(Q: Double; ParamName: String): Double;

Analog zu ParamOfStroke wird ein Linealwert zurückgegeben, wobei Q/Q100 in % übergeben wird.

SaveGraphic

procedure SaveGraphic(FileName: String; GraphIndex: Integer; Width, Height: Integer);

Speichert die Kennliniengrafik in einer Datei. Der Dateiname wird über *FileName* übergeben, die Größe in Pixeln steht in *Width* und *Height*. Verwenden Sie den GroupIndex, um die Grafik zu wählen, die Sie speichern wollen:

0 speichert alle 4 Grafiken
1 speichert die Anlagenkennlinien
2 speichert die Druckbetriebskennlinien
3 speichert die Ventilfaktoren und Schallbetriebskennlinien
4 speichert die Durchflussbetriebskennlinien

Die Grafik kann in den Formaten Bitmap (bmp) oder JPEG (jpg) gespeichert werden. Das Format richtet sich nach der Endung des Dateinamens.

IParameter

Dieses Interface kapselt die Parameter einer Berechnung und die Felder der benutzerdefinierten Daten.

Tag

property Tag: Variant; (r/w)

Dient zur Speicherung beliebiger Informationen.

Application

property Application: IConval12; (r/o)

Gibt das Anwendungsobjekt zurück.

Parent

property Parent: Variant; (r/o)

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall kann es vom Typ *ICalculationData* oder *IOptionalData* sein, je nachdem, ob der Parameter einen festen oder einen benutzerdefinierten Parameter repräsentiert.

Assigned

property Assigned: Boolean; (r/o)

Diese Eigenschaft hat den Wert *True*, wenn der Parameter in der Berechnung verwendet wird. Dies kann sich z. B. bei Veränderung eines Schalters ändern. Hat die Eigenschaft *Assigned* den Wert *False*, so geben die Eigenschaften, die sich auf den Wert des Parameters beziehen, einen Standardwert zurück (z. B. einen Leerstring für die Eigenschaft *Text*).

Name

property Name: String; (r/o)

Diese Eigenschaft liefert den Namen des Parameters.

ParamType

property ParamType: ENumParamType; (r/o)

Diese Eigenschaft liefert den Typ des Parameters. Die Bedeutungen der Werte entnehmen Sie bitte der oben stehenden Tabelle, die den Datentyp *ENumParamType* beschreibt.

ODParamType

property ODParamType: ENumOptionalDataParamKind; (r/o)

Diese Eigenschaft liefert den Typ eines benutzerdefinierten Parameters zurück. Die Bedeutungen der Werte entnehmen Sie bitte der oben stehenden Tabelle, die den Datentyp *ENumOptionalDataParamKind* beschreibt.

InputState

property InputState: ENumInputState; (r/o)

Diese Eigenschaft liefert den Eingabemodus des Parameters.

Beispiel

```
if MyParameter.Inputstate = Calculated then
  MessageBox(MyParameter.ShortCaption + " ist " + MyParameter.Text)
```

In diesem Beispiel wird der Parameter *MyParameter* ausgegeben, wenn er berechnet wurde.

DisplayState

property DisplayState: ENumDisplayState; (r/o)

Diese Eigenschaft liefert den Anzeigemodus des Parameters.

ValueState

property ValueState: ENumValueState; (r/o)

Diese Eigenschaft liefert den Status eines Zahlenparameters.

ReadOnly

property ReadOnly: Boolean; (r/w)

Diese Eigenschaft zeigt an, ob ein Wert über die Maske nur gelesen (d. h. nicht geschrieben) werden kann.

Anmerkung Bitte beachten Sie, dass der Wert über den COM-Server auch dann noch gesetzt werden kann, wenn `ReadOnly` der Wert `True` zugewiesen wurde!

Enabled

property Enabled: Boolean; (r/w)

Diese Eigenschaft zeigt an, ob ein Wert über die Maske zur Verfügung steht.

Anmerkung Bitte beachten Sie, dass der Wert über den COM-Server auch dann noch bearbeitet werden kann, wenn `Enabled` der Wert `False` zugewiesen wurde!

IsOptional

property IsOptional: Boolean (r/o)

Diese Eigenschaft zeigt an, ob ein Parameter zu den benutzerdefinierten Daten gehört.

IsCalculated

property IsCalculated: Boolean; (r/o)

Diese Eigenschaft gibt an, ob der Parameter berechnet wird.

IsCalculatedOverwritten

property IsCalculatedOverwritten: Boolean; (r/o)

Diese Eigenschaft gibt an, ob der Parameter berechnet, aber vom Anwender überschrieben wurde.

IsLookedUp

property IsLookedUp: Boolean (r/o)

Diese Eigenschaft gibt an, ob der Parameter nachgeschlagen wird. D. h. der Wert kommt aus einer Tabelle / Datenbank.

IsLookedUpOverwritten

property IsLookedUpOverwritten: Boolean (r/o)

Diese Eigenschaft gibt an, ob der Parameter nachgeschlagen, aber vom Anwender überschrieben wurde. D. h. eigentlich kommt der Wert aus einer Tabelle / Datenbank, wurde aber überschrieben.

CanBeOverwritten

property CanBeOverwritten: Boolean (r/o)

Diese Eigenschaft gibt an, ob der Wert eines berechneten oder nachgeschlagenen Parameters vom Benutzer überschrieben werden kann.

IsOverwritten

property IsOverwritten: Boolean; (r/o)

Diese Eigenschaft gibt an, ob der Wert eines berechneten oder nachgeschlagenen Parameters vom Benutzer überschrieben worden ist.

Restore

procedure Restore;

Diese Methode stellt einen überschriebenen Wert wieder her (für berechnete oder nachgeschlagene Werte).

LongCaption

property LongCaption: String; (r/o)

Diese Eigenschaft liefert die Beschriftung eines Parameters (Langform). Z. B. liefert

Beispiel
```
MyCalculation.CalculationData.ParamByName["T1"].LongCaption
```

den Wert „Betriebstemperatur“.

ShortCaption

property ShortCaption: String; (r/o)

Diese Eigenschaft liefert die Beschriftung eines Parameters (Langform). Z. B. liefert

Beispiel
```
MyCalculation.CalculationData.ParamByName["T1"].ShortCaption
```

den Wert „t1“.

Value

property Value: Double; (r/w)

Mit dieser Eigenschaft kann der Wert eines Parameters, der eine Zahl enthält, verarbeitet werden. Die Einheit richtet sich nach der Eigenschaft *CalcUnit*.

Beispiel
```
MyCalculation.CalculationData.ParamByName["P1"].CalcUnit.CurrentUnit = 2
MyCalculation.CalculationData.ParamByName["P1"].Value = 10
```

In diesem Beispiel wird dem Betriebsdruck der Wert 10 mbar zugewiesen.

SIValue

property SIValue: Double; (r/w)

Mit dieser Eigenschaft kann – wie mit *Value* – der Wert eines Zahlenparameters gelesen und geschrieben werden. Im Unterschied zur Eigenschaft *Value* handelt es sich hier aber um den Wert in SI-Einheiten.

Beispiel
```
MyCalculation.CalculationData.ParamByName["P1"].Value = 10
```

In diesem Beispiel wird dem Parameter *P1* der Wert 10 bar zugewiesen.

DisplayValue

property DiaplayValue: Double; (r/o)

Mit dieser Eigenschaft kann – wie mit *Value* – der Wert eines Parameters gelesen werden. Dieser wird aber wie in der Berechnungsmaske gerundet.

CalcUnit

property CalcUnit: ICalcUnit; (r/o)

Diese Eigenschaft liefert ein Interface der Klasse *ICalcUnit*, in welchem die aktuelle Einheit des Parameters gekapselt ist.

Text

property Text: String; (r/w)

Diese Eigenschaft ermöglicht einen Zugriff auf den Inhalt eines Parameters in Textform.

Beispiel
```
MyCalculation.CalculationData.ParamByName["FuidName"].Text = "Wasser"
```

DisplayText

property DisplayText: String; (r/o)

Diese Eigenschaft liefert den Wert eines Parameters als Text, wie er im Dialog angezeigt wird.

DisplayValue

property DisplayValue: Double; (r/o)

Diese Eigenschaft liefert den gerundeten Wert eines Parameters als Zahl, wie er im Dialog angezeigt wird. Dabei werden die eingestellten signifikanten Stellen berücksichtigt.

IsEmpty

property IsEmpty: Boolean; (r/w)

Diese Eigenschaft zeigt an, ob der Parameter leer ist. Wird sie auf *True* gesetzt, so wird der Parameter geleert.

SwitchStateCount

property SwitchStateCount: Integer; (r/o)

Diese Eigenschaft liefert die Anzahl der möglichen Schalterzustände, wenn der Parameter ein Schalter ist. Sonst gibt die Eigenschaft 0 zurück.

Beispiel

```
ShowMessage("Die Stoffdatenbank umfaßt " +
  MyCalculation.CalculationData.ParamByName["FluidName"].SwitchStateCount
  + " Stoffe!")
```

In diesem Beispiel wird die Anzahl der Stoffe, die in der Stoffdatenbank erfasst sind, ausgegeben.

SwitchStateNames

property SwitchStateNames[Index: Integer]: String; (r/o)

Diese Eigenschaft liefert den Namen eines mit *Index* indizierten Schalterzustandes.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculationData =
    AConval.Calculations[AConval.CalculationCount].CalculationData
MyPhase = MyCalculationData.ParamByName["Phase"]
for i = 1 to MyPhase.SwitchStateCount
  MyStringList.Add(MyPhase.SwitchStateNames[i])
```

Dieses Beispiel fügt die möglichen Schalterzustände des Zustands eines Stoffes der Liste *MyStringList* hinzu.

SwitchState

property SwitchState: Integer; (r/w)

Diese Eigenschaft liefert den aktuellen Schalterzustand, wenn der Parameter ein Schalter ist. Sonst gibt die Eigenschaft 0 zurück.

Beispiel

```
AConval.NewCalculation("Lochscheiben")
MyCalculationData =
    AConval.Calculations[AConval.CalculationCount].CalculationData
MyPhase = MyCalculationData.ParamByName["Phase"]

...
ShowMessage("Der Zustand ist: " +
    MyPhase.SwitchStateNames[MyPhase.SwitchState])
```

In diesem Beispiel wird der aktuelle Zustand des Stoffes ausgegeben.

Anmerkung: Im Falle des Typs *ptSwitch* gibt es die beiden Zustände 1 – Nein oder 2 – Ja.

VisibleSwitchStateNames

property VisibleSwitchStateNames: String; (r/o)

Diese Eigenschaft liefert alle Namen der Zustände des Schalters, die im aktuellen Zustand der Berechnung wählbar sind.

Message

property Message: String; (r/o)

Diese Eigenschaft liefert ggf. die Meldung, die mit dem Parameter verbunden ist. Diese kann auch mehrzeilig sein.

MessageID

property MessageID: Integer; (r/o)

Diese Eigenschaft liefert ggf. die ID der Meldung, die mit dem Parameter verbunden ist.

MessageClass

property MessageClass: ENumMessageClass; (r/o)

Diese Eigenschaft liefert ggf. die Klasse der Meldung, die mit dem Parameter verbunden ist.

HasMessage

property HasMessage: Boolean; (r/o)

Mit dieser Eigenschaft fregen Sie ab, ob mit dem Parameter eine Meldung verbunden ist.

Note

property Note: String; (r/w)

Über diese Eigenschaft kann die Notiz bearbeitet werden, die mit dem Parameter verknüpft ist.

Comment

property Comment: String; (r/o)

Über diese Eigenschaft kann der Kommentar ausgelesen werden. Im Gegensatz zur Notiz wird der Kommentar für einige (wenige) Parameter von CONVAL® für weitere erklärende Details erzeugt.

DetailLevel

property DetailLevel: ENumDetailLevel; (r/w)

Dies ist die Detailstufe, in der der Parameter sichtbar ist: immer (dlLow), nur, wenn die Details eingeblendet werden (dlHigh), nie (dlInvisible). Der DetailLevel kann nur für optionale Parameter gesetzt werden.

Formula

property Formula: IFormula; (r/o)

Gibt die Formel für optionale Parameter zurück.

ICalcUnit

Das Interface *ICalcUnit* kapselt die Einheit eines Parameters (s. *IParameter* und Objekthierarchie).

Tag

property Tag: Variant; (r/w)

Dient zur Speicherung beliebiger Informationen.

Application

property Application: IConval12; (r/o)

Gibt das Anwendungsobjekt zurück.

Parent

property Parent: Variant; (r/o)

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall handelt es sich um einen Parameter *IParameter*.

Quantity

property Quantity: Integer; (r/o)

Diese Eigenschaft liefert den Index der physikalischen Größe des zugehörigen Parameters zurück.

SetQuantity

procedure SetQuantity(Value: Integer);

Diese Routine setzt die Größe über den Index. Das funktioniert nur bei benutzerdefinierten Daten, da die Größen für normale Berechnungsparameter unveränderlich sind. Diese Routine ersetzt die Schreibmöglichkeit der entsprechenden Eigenschaft *Quantity* aus Kompatibilitätsgründen.

QuantityName

property QuantityName: String; (r/o)

Diese Eigenschaft liefert den Namen der physikalischen Größe des zugehörigen Parameters zurück.

SetQuantityName

```
procedure SetQuantityName(Value: String);
```

Diese Routine setzt die Größe über den Namen. Das funktioniert nur bei benutzerdefinierten Daten, da die Größen für normale Berechnungsparameter unveränderlich sind. Diese Routine ersetzt die Schreibmöglichkeit der entsprechenden Eigenschaft *QuantityName* aus Kompatibilitätsgründen.

UnitCount

```
property UnitCount: Integer; (r/o)
```

Diese Eigenschaft liefert die Anzahl der zulässigen Einheiten für den zugehörigen Parameter.

UnitNames

```
property UnitNames[Index: Integer]: String; (r/o)
```

Diese Eigenschaft liefert den Namen einer mit *Index* indizierten Einheit zurück.

CurrentUnit

```
property CurrentUnit: Integer; (r/w)
```

Über diese Eigenschaft kann auf die aktuelle Einheit des Parameters zugegriffen werden.

UnitName

```
property UnitName: String; (r/w)
```

Über diese Eigenschaft kann auf die aktuelle Einheit des Parameters als Zeichenfolge zugegriffen werden.

IUnits

Das Interface *IUnits* kapselt die globalen Einheitendefinitionen. Hierüber können die vorgegebenen Einheiten manipuliert werden und Informationen über Größen und Einheiten abgerufen werden.

Tag

```
property Tag: Variant; (r/w)
```

Dient zur Speicherung beliebiger Informationen.

Application

```
property Application: IConval12; (r/o)
```

Gibt das Anwendungsobjekt zurück.

Parent

```
property Parent: Variant; (r/o)
```

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist es identisch mit dem Anwendungsobjekt *Application.*

QuantityCount

```
property QuantityCount: Integer; (r/o)
```

Diese Eigenschaft gibt die Anzahl der Größen an.

QuantityNames

```
property QuantityNames[Index: Integer]: String; (r/o)
```

Diese Eigenschaft liefert den mit *Index* indizierten Namen einer Größe.

Beispiel

```
for i = 1 to AConval.Units.QuantityCount
  MyStringList.Add(AConval.Units.QuantityNames[i])
```

In diesem Beispiel wird die Liste der Größen *MyStringList* hinzugefügt.

UnitCount

```
property UnitCount[QuantityIndex: Integer]: Integer; (r/o)
```

Diese Eigenschaft liefert die Anzahl der Einheiten einer Größe zurück.

UnitNames

property UnitNames[QuantityIndex: Integer; Index: Integer]: String; (r/o)

Diese Eigenschaft liefert den Namen einer mit *Index* indizierten Einheit der Größe *QuantityName.*

Beispiel

```
for i = 1 to AConval.Units.UnitCount[1]
  MyStringList.Add(AConval.Units.UnitNames[1, i])
```

In diesem Beispiel wird die Liste der Einheiten einer Temperatur *MyStringList* hinzugefügt.

ConvertFactor

property ConvertFactor[QuantityIndex: Integer; UnitIndex: Integer]: Double; (r/o)

Diese Eigenschaft liefert den Faktor der Umrechnung in die entsprechende Einheit.

ConvertTerm

property ConvertTerm[QuantityIndex: Integer; UnitIndex: Integer]: Double; (r/o)

Diese Eigenschaft liefert den Summanden der Umrechnung in die entsprechende Einheit. Dieser ist außer bei Temperaturen immer gleich Null.

DefaultUnit

property DefaultUnit[QuantityIndex: Integer]: Integer; (r/w)

Mit dieser Eigenschaft wird auf die Vorgabeeinheit einer Größe zugegriffen.

Beispiel

```
AConval.Units.DefaultUnit[1] = 3
```

SchemeCount

Entfällt! Wurde durch GetSchemeCount ersetzt.

SchemeNames

Entfällt! Wurde durch GetSchemeName ersetzt.

GetSchemeCount

function GetSchemeCount(TempScope: ENumTempScope): Integer;

Diese Funktion liefert die Anzahl der gespeicherten privaten bzw. öffentlichen Einheitenschemata zurück. Zur Bedeutung des Parameters *TempScope* beachten Sie bitte die Erläuterungen zum Typ *ENumTempScope*.

GetSchemes

```
function GetSchemes(Index: Integer;
                    PrivateMarker: ENumPrivateMarker;
                    TempScope: ENumTempScope): String;
```

Diese Funktion liefert die Namen der zur Verfügung stehenden Schemata zurück. Dabei muss ein Index angegeben werden, der Werte von 1 bis *GetSchemeCount* annehmen darf. Ferner muss wie bei *GetSchemeCount* ein *TempScope* angegeben werden und ein weiterer Parameter *PrivateMarker*, der angibt in welcher Form die zurückgegebene Zeichenfolge die Information enthalten soll, ob es sich um ein privates oder öffentliches Einheitenschema handelt. Weitere Informationen zum Parameter *PrivateMarker* und mögliche Werte finden Sie in der obigen Tabelle zum Typ *ENumPrivateMarker.*

DefaultScheme

property DefaultScheme: String; (r/w)

Diese Eigenschaft gibt ein zuvor gesichertes Einheitenschema als globale Vorgabe an.

LoadScheme

function LoadScheme(const FileName: String): Boolean;

Diese Funktion lädt ein zuvor gesichertes Einheitenschema zur Bearbeitung. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

SaveScheme

function SaveScheme(const FileName: String): Boolean;

Diese Funktion speichert die aktuelle Liste der voreingestellten Einheiten (d. h. das Einheitenschema) als *FileName*. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

DeleteScheme

function DeleteScheme(const FileName: String): Boolean;

Diese Funktion löscht ein Einheitenschema. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

ISignis

Das Interface *ISignis* kapselt die globalen Voreinstellungen für die signifikanten Stellen.

Tag

property Tag: Variant; (r/w)

Dient zur Speicherung beliebiger Informationen.

Application

property Application: IConval12; (r/o)

Gibt das Anwendungsobjekt zurück.

Parent

property Parent: Variant; (r/o)

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist es identisch mit dem Anwendungsobjekt *Application.*

QuantityCount

property QuantityCount: Integer; (r/o)

Diese Eigenschaft gibt die Anzahl der Größen an.

QuantityNames

property QuantityNames[Index: Integer]: String; (r/o)

Diese Eigenschaft liefert den mit *Index* indizierten Namen einer Größe.

Beispiel

```
for i = 1 to AConval.Signis.QuantityCount
  MyStringList.Add(AConval.Signis.QuantityNames[i])
```

In diesem Beispiel wird die Liste der Größen *MyStringList* hinzugefügt.

Signi

property Signi[QuantityIndex: Integer]: Integer; (r/w)

Diese Eigenschaft ermöglicht den Zugriff auf die voreingestellten signifikanten Stellen einer Größe.

Beispiel

```
AConval.Signis.Signi[1] = 6
```

In diesem Beispiel wird die Anzahl der signifikanten Stellen für alle Temperaturen auf 6 gesetzt.

SchemeCount

property SchemeCount: Integer; (r/w)

Diese Eigenschaft gibt die Anzahl der gesicherten Schemata zurück.

SchemeNames

property SchemeNames[Index: Integer]: String; (r/w)

Diese Eigenschaft liefert den Namen eines mit *Index* indizierten Schemas. Der Index läuft von 1 bis *SchemeCount*.

DefaultScheme

property DefaultScheme: String; (r/w)

Diese Eigenschaft gibt ein zuvor gesichertes Schema als globale Vorgabe an.

LoadScheme

function LoadScheme(const FileName: String): Boolean;

Diese Funktion lädt ein zuvor gesichertes Schema zur Bearbeitung. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

SaveScheme

function SaveScheme(const FileName: String): Boolean;

Diese Funktion speichert die aktuelle Liste der voreingestellten signifikanten Stellen als *FileName*. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

DeleteScheme

function DeleteScheme(const FileName: String): Boolean;

Diese Funktion löscht ein Schema. Der Rückgabewert zeigt an, ob der Befehl erfolgreich ausgeführt wurde.

IFormulas

Das Interface IFormulas kapselt die Liste der Formeln, die für die optionalen Parameter.

FormulaCount

property FormulaCount: Integer; (r/o)

Anzahl der Formeln in der Liste.

Formula

property Formula[Index: Integer]: IFormula; (r/o)

Gibt die Formel mit dem angegebenen Index zurück.

Add

function Add(Name: string; Formula: string): IFormula;

Erzeugt eine neue Formel mit Namen und Formel als Zeichenfolge als Parameter.

Delete

procedure Delete(Index: Integer);

Löscht die Formel mit dem angegebenen Index.

IFormula

Das Interface IFormula kapselt die Eigenschaften einer Formel für die optionalen Parameter.

Name

property Name: string; (r/w)

Zugriff auf den Namen der Formel. Jede Formel muss einen eindeutigen Namen haben, um sie in Auswahllisten voneinander unterscheiden zu können.

Formula

property Formula: string; (r/w)

Der eigentliche Formeltext. Die Conval Bedienhilfe enthält eine Dokumentationzur Syntax.

WantAllModules

property WantAllModules: Boolean; (r/w)

Gibt an, ob die Formel für alle Module zur Verfügung steht.

ModuleStr

property ModuleStr: string; (r/w)

Eine Liste der Module, für die die Formel verfügbar ist. Die Modulbezeichner werden duch ein Semikolon voneinander getrennt.

IsEmpty

property IsEmpty: Boolean; (r/o)

Gibt an, ob der Formeltext leer ist.

GUID

property GUID: string; (r/o)

Gibt eine eindeutige ID für diese Formel zurück.

IsPrivate

property IsPrivate: Boolean; (r/w)

Gibt an, ob die Formel in den Vorlagen für den Benutzer oder für alle Benutzer gespeichert wird.

ITemplates

Das Interface *ITemplates* kapselt verschiedene Funktionen, die den Umgang mit Schemata und Vorlagen erleichtern sollen.

Tag

property Tag: Variant; (r/w)

Dient zur Speicherung beliebiger Informationen.

Application

property Application: IConval12; (r/o)

Gibt das Anwendungsobjekt zurück.

Parent

property Parent: Variant; (r/o)

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist es identisch mit dem Anwendungsobjekt *Application.*

TypeCount

property TypeCount: Integer; (r/o)

Diese Eigenschaft gibt die Anzahl der Schema-Typen an.

Types

property Types[Index: Integer]: string; (r/o)

Diese Eigenschaft gibt die Schema-Typen zurück. Sie erhalten eine Zeichenfolge, die spachunabhängig ist.

TypeExtensions

property TypeExtensions[Index: Integer]: string; (r/o)

Diese Eigenschaft gibt die Dateiendungen der Schema-Typen zurück (3 Buchstaben, ohne Punkt).

GetSchemeCount

```
function GetSchemeCount(TemplateType: Variant;
                        TempScope: ENumTempScope): Integer;
```

Diese Funktion gibt die Anzahl der vorhandenen Schemata / Vorlagen zurück. Als *TemplateType* kann entweder der Index des Schema-Typen (s. o.) übergeben werden oder, was für Berechnungsvolagen sinnvoll ist, die Dateiendung. *TempScope* gibt an, ob nur private, nur öffentliche oder alle Schemata / Vorlagen gezählt werden sollen.

GetSchemeName

```
function GetSchemeName(TemplateType: Variant;
                       Index: Integer;
                       PrivateMarker: ENumPrivateMarker;
                       TempScope: ENumTempScope): string;
```

Diese Funktion gibt den Namen eines Schemas / einer Vorlage zurück. Als *TemplateType* kann entweder der Index des Schema-Typen (s. o.) übergeben werden oder, was für Berechnungsvolagen sinnvoll ist, die Dateiendung. Der *Index* läuft von 1 bis *GetSchemeCount*. *PrivateMarker* gibt an, ob der Name des Schemas / der Vorlage auch einen Hinweis enthalten soll, ob dieses private oder öffentlich ist. *TempScope* gibt an, ob nur private, nur öffentliche oder alle Schemata / Vorlagen berücksichtigt werden sollen.

GetFileName

```
function GetSchemeName(TemplateType: Variant;
                       Index: Integer;
                       TempScope: ENumTempScope;
                       WantPath: Boolean): string;
```

Diese Funktion gibt den Dateinamen eines Schemas / einer Vorlage zurück. Als *TemplateType* kann entweder der Index des Schema-Typen (s. o.) übergeben werden oder, was für Berechnungsvolagen sinnvoll ist, die Dateiendung. Der *Index* läuft von 1 bis *GetSchemeCount*. *TempScope* gibt an, ob nur private, nur öffentliche oder alle Schemata / Vorlagen berücksichtigt werden sollen. *WantPath* gibt an, ob der Dteiname den kompletten Pfad enthalten soll.

GeIdentFromFIe

```
function GetIdentFromFile(AFileName: string;
                          TempScope: ENumTempScope): string;
```

Diese Funktion gibt den Bezeichner eines Schemas / einer Vorlage zurück, wenn den Dateiname übergeben wird.

GetSchemeNames

```
function GetSchemeNames(TemplateType: Variant;
                        PrivateMarker: ENumPrivateMarker;
                        TempScope: ENumTempScope): string;
```

Diese Funktion gibt die Namen aller Schemata / Vorlagen zurück. Als *TemplateType* kann entweder der Index des Schema-Typen (s. o.) übergeben werden oder, was für Berechnungsvolagen sinnvoll ist, die Dateiendung. *PrivateMarker* gibt an, ob der Name des Schemas / der Vorlage auch einen Hinweis enthalten soll, ob dieses private oder öffentlich ist. *TempScope* gibt an, ob nur private, nur öffentliche oder alle Schemata / Vorlagen berücksichtigt werden sollen. Die Namen werden durch Zeilenumbrüche voneinander getrennt.

GetFileNames

```
function GetSchemeNames(TemplateType: Variant;
                        TempScope: ENumTempScope;
                        WantPath: Boolean): string;
```

Diese Funktion gibt die Dateinamen aller Schemata / aller Vorlagen zurück. Als *TemplateType* kann entweder der Index des Schema-Typen (s. o.) übergeben werden oder, was für Berechnungsvolagen sinnvoll ist, die Dateiendung. *TempScope* gibt an, ob nur private, nur öffentliche oder alle Schemata / Vorlagen berücksichtigt werden sollen. *WantPath* gibt an, ob der Dateiname den kompletten Pfad enthalten soll. Die Namen werden durch Zeilenumbrüche voneinander getrennt.

ICVFile

Das Interface *ICVFile* bietet eine einfache Schnittstelle zu Berechnungen, die nicht berechnet oder verändert sondern nur gelesen oder exportiert werden sollen. Sie erhalten einen weit reichenden Lesezugriff auf zahlreiche Attribute aller Parameter einer Berechnung. Das Laden einer Berechnung dauert über diese Schnittstelle nur einen Bruchteil der Zeit, die Sie mit *ICalculation* aufwenden müssten. Die Hauptschnittstelle *IConval12* führt eine Liste aller Berechnungen, die Sie geöffnet haben. Diese Liste umfasst auch alle in Dialogen und *ICalculation*-Schnittstellen geöffneten Berechnungen.

Beachten Sie bitte, dass alle Berechnungen, die nicht über einen Dialog oder eine *ICalculation*-Schnittstelle geöffnet wurden, sondern mit *IConval12.LoadCVFile(FileName)*, zuvor mit CONVAL 7 oder einer späteren Version gespeichert sein müssen. Ansonsten stehen die komfortablen Möglichkeiten, auf die Attribute der Parameter und weitere Infos zuzugreifen, nicht zur Verfügung. Verwenden Sie *IConval12.IsCV7File(FileName)*, um herauszufinden, ob eine Datei in dem neuen Dateiformat gespeichert wurde.

Load

```
function Load(FileName: WideString): Boolean;
```

Lädt eine Datei. Der Rückgabewert zeigt an, ob die Datei erfolgreich geladen werden konnte.

Save

```
function Save(FileName: String): Boolean;
```

Speichert eine Datei. Der Rückgabewert zeigt an, ob die Datei erfolgreich gespeichert werden konnte.

FileName

```
property FileName: String; (r/o)
```

Mit diesem Attribut ermitteln Sie den Dateinamen.

GetParamAttribute

```
function GetParamAttribute(ParamName, AttributeName: String): String;
```

Ermitteln Sie die Werte der Attribute, die einen Parameter beschreiben. Gängige Attribute sind z. B.:

NotesList .....Notiz
CommentsList .....Kommentar
DisplayText.....Text, wie in der Maske angezeigt
InputState .....Eingabestatus
DisplayState .....Anzeigestatus
OverRuled .....Überschrieben?
CanBeOverRuled .....Kann überschrieben werden?
ReadOnly .....Nur lesen?
Enabled .....Aktiv
Utilized.....Wird verwendet? (Wenn 0 können andere Werte undefiniert sein)
Modified.....geändert?
CanBeModified.....Kann geändert werden?
TroubleMaker .....Fehler / Warnung / Hinweis o. ä. vorhanden?
TroubleType .....Art der Meldung
CalcMessage .....Meldung
DetailLevel.....sichtbar in Darstellung mit wenig / viel Details
ShowAlways.....Immer sichtbar?
Important .....wichtiger Parameter? (werden in der Maske farbig markiert)
Name.....interner Name des Parameters
Caption .....lange Beschreibung des Parameters
ShortCaption .....Kurzbezeichnung des Parameters
Text .....der Text (ungekürzt)
Value .....(Zahlen-)wert
CalcUnitNo .....Einheit
ValueState.....ggf. spezieller Status des Wertes (z. B. bei Überlauf oder leerem Wert)
IsApproximated .....Wert genähert
CalcQuantity.....Größe

Diese Funktion ist eine Spezialisierung der Funktion *AttributeValue* (s. u.). Sie ist identisch zum Aufruf *AttributeValue(„Items\“ + ParamName, AttributeName)*.

CalculationKind

```
function CalculationKind: Integer;
```

Gibt die Art der Berechnung zurück. Der COM-Server ermittelt diese aus der Dateiendung.

Close

```
procedure Close;
```

Schließt die Berechnung. Die eigentliche Berechnung wird auf der Festplatte wieder freigegeben und das interne Objekt gelöscht. Sie können auf das Interface *ICVFile* noch immer zugreifen und den Status über *Assigned* abfragen. Dieser ist nach einem *Close False*.

ItemList

```
function ItemList(Path, Divider: String): String;
```

Die Daten, die in einer CONVAL-Datei gespeichert sind, werden in einer Baumstruktur verwaltet. Die Knoten und Blätter dieses Baums heißen *Items*. Jedes *Item* kann Attribute enthalten, die einen Namen und einen Wert haben. Verwenden Sie die Funktion *ItemList*, um zu ermitteln, welche Items an einem Knoten hängen. Um den Knoten zu beschreiben, geben Sie den Pfad zu diesem an (ähnlich den Verzeichnissen in einem Datei- und Verzeichnissystem auf der Festplatte). Verwenden Sie als Trenner „\“ (wie bei Windows-Ordnern). Das Ergebnis ist eine Liste mit Namen von Items, die durch eine Zeichenfolge von einander getrennt werden, die Sie als Parameter *Divider* übergeben.

Sie finden die Liste der Parameter beispielsweise mit *ItemList(„Items“, „\“)*, die Liste der benutzerdefinierten Daten mit *ItemList(„OptionalData\Items“, „\“)*.

AttributeList

```
function AttributeList(Path, Divider: String): String;
```

Mit dieser Funktion erhalten Sie analog zu ItemList die Liste der Attribute eines Items. Geben Sie den Pfad zum Item und den Trenner der einzelnen Attribute im Ergebnis an.

AttributeValue

```
function AttributeValue(Path, AttributeName: String): String´;
```

Mit dieser Funktion erhalten Sie den Wert eines Attributs als Zeichenfolge.

ICVExport

Dieses Interface dient dem Export von CONVAL-Berechnungen wie er in CONVAL 7 neu eingeführt worden ist (Datei -> Exportieren -> Diverse Formate ...). Zur Spezifikation des Exports sind verschiedene Angaben notwendig, die über die Routinen und Attribute des Interfaces eingestellt werden bevor der eigentliche Export mittels *DoExport* ausgeführt wird. Alle Dateien, die für den Export ausgewählt werden, erscheinen auch in der Liste IConval12.CVFiles.

AddFileByName

```
procedure AddFileByName(FileName: String);
```

Eine Datei mit dem Namen FileName zu der Liste der Dateien, die exportiert werden sollen, hinzufügen.

DelFileByName

```
procedure DelFileByName(FileName: String);
```

Eine Datei mit dem Namen FileName aus der Liste der Dateien, die exportiert werden sollen, entfernen. Damit wird die Datei nicht aus dem Speicher entfernt, sondern bleibt quasi im „Cache“.

ClearFileList

```
procedure ClearFileList;
```

Liste der Dateien löschen. Damit werden die Dateien nicht aus dem Speicher entfernt.

TemplateName

```
property TemplateName: String; (r/w)
```

Name der Vorlagen-Datei, die für den Export verwendet werden soll.

AddFile

```
procedure AddFile(AFile: ICVFile);
```

Eine Datei hinzufügen, die sich bereits im Speicher befindet. Wenn Sie schon ein Interface einer Datei zur Verfügung haben, verwenden Sie diese Routine.

DelFile

```
procedure DelFile(AFile: ICVFile);
```

Löscht eine Datei aus der Liste, die Sie anhand des Interfaces *ICVFile* übergeben. Die Datei wird nur aus der Liste entfernt, nicht aber aus dem Speicher gelöscht.

CVFiles / CVFileCount

```
property CVFiles[Index: Integer]: ICVFile; (r/o)
property CVFileCount: Integer; (r/o)
```

Liste der Dateien, die für den Export vorgesehen sind. Die Liste wird wie überall im COM-Server gehandhabt, d. h. Sie übergeben als Index einen Wert von 1 ... *CVFileCount*.

DoExport

```
procedure DoExport(FileName: String);
```

Mit dieser Routine führen Sie schließlich den Export aus. Geben Sie als FileName den Dateinamen an, unter dem das Ergebnis gespeichert werden soll. Beachten Sie bitte, dass nach dem jetzigen Stand das Ergebnis des Exports auch immer angezeigt wird.

IConcentration

Das Interface *IConcentration* kapselt die Stoffzusammensetzungen, die bei den Medien ausgewählt werden können. Es können die Schemata verwaltet und bearbeitet werden.

Beachten Sie, dass Sie eine Stoffzusammensetzung, die in einer Berechnung verwendet werden soll, über den Parameter *Concentration* auswählen müssen. Sie entscheiden über *IConcentrations.IsPrivate* für den jeweiligen *MixType*, ob Sie beim Parameter *Concentration* ein privates oder ein öffentliches Schema auswählen.

In CONVAL gibt es derzeit 4 verschiedene Arten von Stoffgemischen: GERG 2008, Erdgas AGA 8, Gasgemische und Flüssigkeitsgemische. GERG und AGA waren bereits in früheren CONVAL Versionen verfügbar. Ab CONVAL 10.2 können Sie auch Gas- und Flüssigkeitsgemische mit dem COM-Server verwalten. Um dies zu ermöglichen, musste es aber ein paar Änderungen im Handling geben:

- Die neue Eigenschaft *MixType* entscheidet über den Typ der Zusammensetzung. Dies ist zuerst auszuwählen. Dabei sind AGA und GERG zu einem Typ zusammengefasst. Es gibt also die Typen *mixtypAgaGerg*, *mixtypGas* und *mixtypLiquid*.
- Nach dieser Auswahl ist die Liste der Schemata (*GetSchemeName* / *GetSchemeCount*) entsprechend gefüllt. Nun kann ein Schema geladen (*LoadScheme*) oder neu erstellt (*NewScheme*) werden.
- Nachdem ein Schema geladen oder neu erstellt wurde, sollten alle Aktionen damit ohne Unterbrechung durchgeführt werden. Am Ende sollte das Schema geschlossen werden (*CloseScheme*), bevor beispielsweise über eine Berechnung auf die Schemata zugegriffen wird. Solange ein Schema noch nicht geschlossen ist, kann u.a. der Typ (*MixType*) nicht geändert werden.
- In einem geöffneten Schema kann nun über *ItemName*, *ItemCaption*, *ItemValue*, *ItemCount* auf die Bestandteile wie gehabt zugegriffen werden. Weiterhin kann über *Normalize* auf 100% normalisiert werden und für AGA und GERG der *ConcentrationType* geändert werden.
- Für AGA und GERG Gemische sind die Bestandteile festgelegt. Dies ist bei Gas- und Flüssigkeitsgemischen anders, d.h. Die Stoffe können beliebig aus der Stoffdatenbank zusammengesetzt werden. Daher gibt es zusätzlich Routinen, um einen neuen Eintrag hinzuzufügen (*AddItem*), einen Eintrag zu löschen (*DeleteItem*) und alle Einträge zu löschen (*ClearItems*). Diese Routinen haben für AGA und GERG keine Funktion.

- Nachdem die Informationen ausgelesen wurden und ggf. Änderungen vorgenommen wurden muss nun also das Schema, das zur Ansicht und Bearbeitung geöffnet wurde, ggf. gespeichert (*SaveScheme*) und dann wieder geschlossen werden (*CloseScheme*). Wird beim Speichern kein Name angegeben, so wird der aktuelle Name verwendet. Vor dem Speichern wird das Schema normalisiert.

Tag

```
property Tag: Variant; (r/w)
```

Dient zur Speicherung beliebiger Informationen.

Application

```
property Application: IConval12; (r/o)
```

Gibt das Anwendungsobjekt zurück.

Parent

```
property Parent: Variant; (r/o)
```

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist es identisch mit dem Anwendungsobjekt *Application.*

Verwaltung der Schemata

GetSchemeCount

```
function GetSchemeCount(TempScope: ENumTempScope): Integer; (r/o)
```

Diese Funktion gibt die Anzahl der privaten bzw. öffentlichen Schemata zurück. Zur Bedeutung des Parameters *TempScope* beachten Sie bitte die Erläuterungen zum Typ *ENumTempScope*.

GetSchemeName

```
function GetSchemeName(Index: Integer;
                       PrivateMarker: ENumPrivateMarker;
                       TempScope: ENumTempScope): String;
```

Diese Funktion liefert die Namen der zur Verfügung stehenden Schemata zurück. Dabei muss ein Index angegeben werden, der Werte von 1 bis *GetSchemeCount* annehmen darf. Ferner muss wie bei *GetSchemeCount* ein *TempScope* angegeben werden und ein weiterer Parameter *PrivateMarker*, der angibt in welcher Form die zurückgegebene Zeichenfolge die Information enthalten soll, ob es sich um ein privates oder öffentliches Einheitenschema handelt. Weitere Informationen zum Parameter *PrivateMarker* und mögliche Werte finden Sie in der obigen Tabelle zum Typ *ENumPrivateMarker*.

GetSchemeIsPrivate

```
function GetSchemeIsPrivate(Index: Integer): Boolean;
```

Fragt ab, ob das Schema *Index* ein privates ist.

NewScheme

```
procedure NewScheme;
```

Erstellt ein neues, leeres Schema uns öffnet dieses. Verwenden Sie *CloseScheme* zum Schließen des Schemas, bevor Sie andere Funktionen des COM-Servers verwenden.

LoadScheme

```
function LoadScheme(const Name: String; IsPrivate: Boolean);
```

Diese Funktion lädt ein zuvor gesichertes Schema zur Bearbeitung. Beachten Sie, dass das Schema auch wieder geschlossen werden muss, bevor Sie andere Funktionen des COM-Servers verwenden!

SaveScheme

```
function SaveScheme(const Name: String; IsPrivate: Boolean);
```

Diese Funktion speichert die aktuellen Werte als Schema *Name*. Lassen Sie den Namen leer, um das Schema unter dem aktuellen Namen zu speichern. *IsPrivate* wird dann nicht berücksichtigt.

Beachten Sie, dass das Schema nach dem Speichern nicht automatisch geschlossen ist. Verwenden Sie *CloseScheme* zum Schließen des Schemas, bevor Sie andere Funktionen des COM-Servers verwenden.

DeleteScheme

```
function DeleteScheme(const FileName: String; IsPrivate: Boolean);
```

Diese Funktion löscht ein Schema.

CloseScheme

```
procedure CloseScheme;
```

Schließt das aktuell zur Bearbeitung geöffnete Schema.

MixType

```
property MixType: ENumMixType; (r/w)
```

Bestimmt den Typ der Stoffzusammensetzung, auf den sich die obigen Funktionen beziehen. Beachten Sie, dass Sie ggf. ein geöffnetes Schema mit *CloseScheme* schließen müssen, bevor Sie diese Eigenschaft ändern können!

AGA8

```
property AGA8: Boolean; (r/w)
```

Filtert die Liste der Schemata auf diejenigen, die die Norm AGA8 erfüllen.

Verwaltung des geladenen Schemas

Die folgenden Attribute und Methoden bilden im Wesentlichen den Dialog „Gemisch bearbeiten“ in CONVAL ab.

AddItem

```
procedure AddItem(Name: String; Value: Double);
```

Erzeugt ein neues Element der Stoffzusammensetzung. Diese Funktion steht nicht bei Gemischen nach AGA und GERG zur Verfügung, da hier die verfügbaren Stoffe fest vorgegeben sind.

DeleteItem

```
procedure DeleteItem(Index: Integer);
```

Löscht das Element der Stoffzusammensetzung mit dem entsprechenden Index. Diese Funktion steht nicht bei Gemischen nach AGA und GERG zur Verfügung, da hier die verfügbaren Stoffe fest vorgegeben sind.

ClearItems

```
procedure ClearItems;
```

Löscht alle Elemente der Stoffzusammensetzung. Diese Funktion steht nicht bei Gemischen nach AGA und GERG zur Verfügung, da hier die verfügbaren Stoffe fest vorgegeben sind.

Name

```
property Name: String; (r/w)
```

Der Name des aktuellen Schemas.

IsPrivate

```
property IsPrivate: Boolean; (r/w)
```

Gibt an, ob das aktuelle Schema privat oder öffentlich ist.

ItemCount

```
property ItemCount: Integer; (r/o)
```

Gibt die Anzahl der Werte des Schemas zurück.

ItemName

property ItemName[Index: Integer]: String; (r/o)

Liefert den Namen des Wertes *Index*.

ItemCaption

property ItemCaption[Index: Integer]: String; (r/o)

Liefert die Bezeichnung des Wertes *Index* in der aktuellen Sprache.

ItemValue

property ItemValue[Index: Integer]: Double; (r/w)

Bietet den Zugriff auf einen Wert des aktuellen Schemas.

IsAGA8

property IsAGA8: Boolean;

Gibt an, ob die aktuelle Stoffzusammensetzung die Norm AGA8 erfüllt.

IsGERG

property IsGERG: Boolean;

Gibt an, ob die aktuelle Stoffzusammensetzung die Norm GERG erfüllt.

ConcentrationType

property ConcentrationType: Integer; (r/w)

Bietet Zugriff auf die Einheit, in der die Stoffe erfasst sind.

Sum

property Sum: Double; (r/o)

Liefert die Summe der Konzentrationen. Diese muss beim Speichern 100% betragen, sonst erhalten Sie eine Fehlermeldung. Verwenden Sie *Normalize*, um die Konzentrationen zu normieren.

Normalize

procedure Normalize;

Normalisiert die Konzentrationen, so dass die Summe 100% ergibt. Dabei wird vorausgesetzt, dass die angegebenen Zahlen die Verhältnisse der Stoffe widerspiegeln.

LastUsed

property LastUsed: Double; (r/o)

Gibt das Datum der letzen Verwendung des aktuellen Schemas zurück.

ILanguageOptions

ILanguageOptions kapselt die Sprachoptionen, die in dem entsprechenden Dialog in CONVAL bearbeitet werden können.

LanguageCount

property LanguageCount: Integer; (r/o)

Gibt die Anzahl der Sprachen zurück, die CONVAL anbietet.

LanguageNames

property LanguageNames(Index: Integer): string; (r/o)

Namen der verfügbaren Sprachen.

LanguageISONames

property LanguageISONames(Index: Integer): string; (r/o)

ISO Bezeichner der zur verfügbaren Sprachen.

LanguageNo

property LanguageNo: Integer; (r/w)

Der Index der gewählten Sprache. Setzen Sie die Sprache, die CONVAL verwenden soll, über dieses Attribut um. Der Index bezieht sich auf die Attribute *LanguageNames* / *LanguageCount*.

StartLanguageNo

property StartLanguageNo: Integer; (r/w)

Der Index der Sprache, in der CONVAL gestartet werden soll, wenn *StartLanguageMode* auf *slmFixLangauge* steht. Ansonsten wird diese Einstellung ignoriert. Dies muss nicht die aktuell eingestellte Sprache sein. Der Index bezieht sich auf die Attribute *LanguageNames* / *LanguageCount*.

ActiveLanguageNo

property ActiveLanguageNo: Integer; (r/o)

Der Index der aktuellen Sprache. Diese Sprache wird derzeit von CONVAL verwendet. I.d.R. identisch zu LangaugeNo. Der Index bezieht sich auf die Attribute *LanguageNames* / *LanguageCount*.

LocaleCount

property LocaleCount: Integer; (r/o)

Anzahl der verfügbaren regionalen Formateinstellungen. Diese sind anhängig von der gewählten Sprache.

LocaleLanguageNames

property LocaleLanguageNames(Index: Integer): string; (r/o)

Die Bezeichner der verfügbaren regionalen Formateinstellungen. Diese sind anhängig von der gewählten Sprache. I.d.R. handelt es sich um Länder, in denen die gewählte Sprache Amtssprache ist.

LocaleNo

property LocaleNo: Integer; (r/w)

Der Index der gewählten regionalen Formateinstellungen. Der Index bezieht sich auf die Attribute *LocaleLanguageNames* / *LocaleCount*.

UseWindowsLocale

property UseWindowsLocale: Boolean; (r/w)

Gibt an, ob die Vorgabe von Windows für die regionalen Formateinstellungen verwendet werden soll. Ist dieses Attribut True (Vorgabe) wird die LocalNo ignoriert.

DBLanguageCount

property DBLanguageCount: Integer; (r/o)

Die Anzahl der verfügbaren Sprachen für die Datenbanken in CONVAL. Beachten Sie, dass die Datenbanken nicht in allen Sprachen verfügbar sind. Daher gibt es hierfür eine weitere Liste.

DBLanguageNames

property DBLanguageNames(Index: Integer): string; (r/o)

Die verfügbaren Sprachen für die Datenbanken in CONVAL. Beachten Sie, dass die Datenbanken nicht in allen Sprachen verfügbar sind. Daher gibt es hierfür eine weitere Liste.

DefaultDBLanguageNo

property DefaultDBLanguageNo: Integer; (r/w)

Der Index der gewählten Sprache für die Datenbanken in CONVAL. Beachten Sie, dass die Datenbanken nicht in allen Sprachen verfügbar sind. Daher gibt es hierfür eine weitere Liste. Der Index bezieht sich auf die Attribute *DBLanguageNames* / *DBLanguageCount*.

StartLanguageMode

property StartLanguageMode: ENumStartLanguageMode; (r/w)

Legt fest, welche Sprache beim Start von CONVAL verwendet werden soll.

Optionen

Die Interfaces *IMainOptions* und *IPrintOptions* sind gleich aufgebaut. Sie unterscheiden sich nur in den Namen der Optionen, die vom COM-Server ausgewertet werden können.

Optionen von *IMainOptions*:

Option	Erklärung	Format
EditBgColorMissing	Hintergrundfarbe für fehlende Parameter	Color
EditBgColorNormal	Hintergrundfarbe für normale Parameter	Color
EditBgColorFocus	Hintergrundfarbe für aktuell ausgewählten Parameter	Color
EditBgColorCalc	Hintergrundfarbe für berechnete Parameter	Color
EditBgColorCalcFocus	Hintergrundfarbe für aktuell ausgewählten Parameter, wenn dieser ein berechneter Wert ist	Color
EditBgColorDisabled	Hintergrundfarbe für nicht verfügbare Parameter	Color
EditBgColorImportant	Hintergrundfarbe für wichtige Parameter	Color
EditTxColorMissing	Schriftfarbe für fehlende Parameter	Color
EditTxColorNormal	Schriftfarbe für normale Parameter	Color
EditTxColorFocus	Schriftfarbe für aktuell ausgewählten Parameter	Color
EditTxColorCalc	Schriftfarbe für berechnete Parameter	Color
EditTxColorCalcFocus	Schriftfarbe für aktuell ausgewählten Parameter, wenn dieser ein berechneter Wert ist	Color
EditTxColorDisabled	Schriftfarbe für nicht verfügbare Parameter	Color
EditTxColorImportant	Schriftfarbe für wichtige Parameter	Color
WantImportant	Wichtige Parameter sollen farbig gekennzeichnet werden	Boolean
WantMissing	Fehlende Parameter sollen farbig gekennzeichnet werden	Boolean
BoldUnits	Geänderte Einheiten (solche, die nicht dem aktuellen Einheitenschema entsprechen), werden fett dargestellt	Boolean
UnitScheme	Das vorgegebene Einheitenschema	String
SigniScheme	Das vorgegebene Schema für signifikante Stellen	String
OldFileWarning	Warnung, wenn eine Datei mit einer älteren CONVAL-Version erzeugt worden ist	Boolean
FileHistoryCount	Anzahl der Berechnungen, die in der Liste der zuletzt bearbeiteten Berechnungen zum öffnen angeboten werden	Integer
MenuStyle	Der Stil, in dem das Menü dargestellt wird	*0* - Standard (zwei vertikale Linien), *1* - Enhanced (eine vertikale Linie), *2* - Flat ("Flache" Schaltflächen)
UndoCount	Anzahl der möglichen Schritte, die rückgängig gemacht werden können.	Integer
TrayIcon	Ein Symbol neben der Uhr soll angezeigt werden.	Boolean

Optionen von *IMainOptions*:

Option	Erklärung	Format
OnlyOneDefaultADS	Nur ein Schema für benutzerdefinierte Daten für alle Module verwenden.	Boolean
ShowAddDatas	Benutzerdefinierte Daten anzeigen.	Boolean
DefaultAddDataScheme	Vorgegebenes Schema für benutzerdefinierte Daten (für alle Module gleich; wird verwendet, wenn OnlyOneDefaultADS true ist)	String
DefaultAddDataSchemes	Vorgegebene Schemata für benutzerdefinierte Daten (für alle Module einzeln zu definieren; wird verwendet, wenn OnlyOneDefaultADS false ist)	Array of String
PageFormatList	Liste der Seitenformate	Array of String
UseDefaultDBPath	Gibt an, ob *DefaultDBPath* verwendet werden soll oder die Pfade zu den einzelnen Datenbanken.	Boolean
DefaultDBPath	Pfad zu den Datenbanken, wenn sie nicht einzeln angegeben sind	String
MaterialsDBPath	Pfad, in dem sich die Tabellen für die Materialdaten befinden	String
PropertiesDBPath	Pfad, in dem sich die Tabellen für die Stoffdaten befinden	String
ControlValvesDBPath	Pfad, in dem sich die Tabellen für die Stellventildaten befinden	String
SafetyValvesDBPath	Pfad, in dem sich die Tabellen für die Sicherheitsventildaten befinden	String
PumpMotorDBPath	Pfad, in dem sich die Tabellen für die Pumpenmotordaten befinden	String
HideAddDatas	Benutzerdefinierte Daten in der Ansicht mit wenig Details ein- und ausblenden	Boolean
PublicTemplatePath	Pfad in dem alle öffentlichen Vorlagen und Schemata liegen; das ist normalerweise ein Unterverzeichnis des CONVAL-Installationsverzeichnisses oder das Dokumente-Verzeichnis für alle Benutzer.	String
PrivateTemplatePath	Pfad in dem alle privaten Vorlagen und Schemata liegen; das ist normalerweise ein Unterverzeichnis des persönlichen Profils.	String
DefaultTemplates	Berechnungsvorlagen für alle Module; diese werden verwendet, wenn eine neue Datei angelegt wird.	Array of String
FlowElementDefScheme	Vorlage für die Parameterauswahl in der Tabelle im Messblendenmodul	String
ExpStep	Zahlenformat für Zahlen mit Exponent. Gibt an, ob der Exponent durch 3 teilbar sein soll. Das erleichtert das Lesen großer Zahlen.	Integer (Werte: 1, 3)
RecentUnitCount	Maximale Anzahl der zuletzt verwendeten Einheiten.	Integer
ScrollMouseMode	Mausrad-Modus: 0 - Standard (Eingabebereich verschieben) 1 - wie Tabulatortaste (Fokus ändern) 2 - wie <Enter>-Taste (Fokus ändern)	Integer

Optionen von *IMainOptions*:

Option	Erklärung	Format
PDF4compatible	Gewährleistet die Kompatibilität zu Acrobat 4 indem eine einfachere Verschlüsselung verwendet wird. Diese Option hat keine Auswirkung, wenn *PDFLock* nicht gesetzt ist.	Boolean
PDFLock	Beim PDF-Export wird die Datei verschlüsselt, so dass sie nicht mehr verändert werden kann. Mittels *PDF4compatible* haben Sie Einfluss auf die Verschlüsselungsart.	Boolean
ExportPrinter	Name eines Druckers, der anstelle eines PDF-Exports verwendet werden soll. Setzen Sie einen Leerstring falls der CONVAL-PDF-Printer verwendet werden soll.	String
PreviewPrinter	Names eines Druckers, der anstelle der Druckvorschau verwendet werden soll. Setzen Sie einen Leerstring falls der CONVAL-PDF-Printer verwendet werden soll.	String

Optionen von *IPrintOptions*

Option	Erklärung	Format
PageFormat	Papierformat	0 - Hochformat, 1 - Querformat
PageWidth	Seitenbreite	Integer (1/1000 mm)
PageHeight	Seitenhöhe	Integer (1/1000 mm)
PageFormatName	Name des Papierformats	String
LeftMargin	Linker Rand	Integer (1/1000 mm)
RightMargin	Rechter Rand	Integer (1/1000 mm)
TopMargin	Oberer Rand	Integer (1/1000 mm)
BottomMargin	Unterer Rand	Integer (1/1000 mm)
HeadHeight	Höhe des Bereiches für die Kopfzeilen	Integer (1/1000 mm)
FootHeight	Höhe des Bereiches für die Fußzeilen	Integer (1/1000 mm)
HeadHeight1	Höhe des Bereiches für die Kopfzeilen auf der ersten Seite	Integer (1/1000 mm)
FootHeight1	Höhe des Bereiches für die Fußzeilen auf der ersten Seite	Integer (1/1000 mm)
HeadHeight2	Höhe des Bereiches für die Kopfzeilen auf den geraden Seiten	Integer (1/1000 mm)
FootHeight2	Höhe des Bereiches für die Fußzeilen auf den geraden Seiten	Integer (1/1000 mm)
OppositePages	Gegenüberliegende Seiten	Boolean
AutoHeight	Automatische Berechnung der Kopf- und Fußzeilenhöhen	Boolean
BeforeHeadSpc	Abstand vor einer neuen Überschrift	Integer (1/1000 mm)
AfterHeadSpc	Abstand nach einer Überschrift	Integer (1/1000 mm)
AfterLineSpc	Zeilenabstand	Integer (1/1000 mm)
HeadFont1	Schriftart für die Hauptüberschriften	Font
HeadFont2	Wird z. Zt. nicht verwendet	Font
HeadFont3	Schriftart für die Überschriften	Font
HeadFont4	Schriftart für die Tabellenüberschriften	Font
FontLongCaption	Schriftart für die langen Bezeichner der Parameter	Font
FontShortCaption	Schriftart für die kurzen Bezeichner der Parameter	Font
FontValue	Schriftart für die Werte der Parameter	Font
FontUnit	Schriftart für die Einheiten der Parameter	Font
HeadFoot	Sollen Kopf- und Fußzeilen ausgegeben werden	Boolean
HeadFoot1	Sollen Kopf- und Fußzeilen auf der ersten Seite ausgegeben werden	Boolean
HeadFoot2	Sollen Kopf- und Fußzeilen auf geraden Seiten ausgegeben werden	Boolean
PrintCalculation	Berechnung ausdrucken	Boolean

Optionen von *IPrintOptions*

Option	Erklärung	Format
PrintEmpty	Leere Parameter ausgeben	Boolean
AddData	Benutzerdefinierte Daten ausgeben	Boolean
AddDataPos	Position der benutzerdefinierten Daten im Ausdruck	0 - Unter den Berechnungskopf, 3 - Am Ende der Berechnung, 4 - Wie in der Maske
Messages	Meldungen ausgeben	Boolean
MessagesPos	Position der Meldungen im Ausdruck (--)	
MessageClasses	Klassen der auszugebenden Meldungen	0 - Fehler, 1 - Warnungen, 2 - Hinweise
Legend	Legende drucken	Boolean
PrintSymbols	Symbole drucken	Boolean
Graphics	Grafiken und Tabellen ausdrucken	Boolean
Notes	Gibt an, ob Notizen ausgeben werden sollen.	Boolean
NotesPoss	Position der Notizen im Ausdruck: Werte: *ppParameter*,*ppEnd*	String
NoteFrameColor	Farbe des Rahmens um eine Notiz	Color
NoteFramFillColor	Hintergrundfarbe der Notizen	Color
Comments	Gibt an, ob Kommentare ausgeben werden sollen.	Boolean
CommentsPoss	Position der Kommentare im Ausdruck: Werte: *ppParameter*,*ppEnd*	String
CommentFrameColor	Farbe des Rahmens um einen Kommentar	Color
CommentFrameFillColor	Hintergrundfarbe der Kommentare	Color
FormatTablesLikeParams	Die Werte in Tabellen werden genauso formatiert wie die anderen Parameter.	Boolean
HeadScheme	Schema für Kopfzeilen	String ("Name:private" oder "Name:public")
FootScheme	Schema für Fußzeilen	String ("Name:private" oder "Name:public")
HeadScheme1	Schema für Kopfzeilen auf der ersten Seite	String ("Name:private" oder "Name:public")
FootScheme1	Schema für Fußzeilen auf der ersten Seite	String ("Name:private" oder "Name:public")
HeadScheme2	Schema für Kopfzeilen auf den geraden Seiten	String ("Name:private" oder "Name:public")
FootScheme2	Schema für Fußzeilen auf den geraden Seiten	String ("Name:private" oder "Name:public")
Pages	Welche Seiten sollen gedruckt werden	0 - Alle, 1 - Ungerade, 2 - Gerade, 3 - Benutzerdefiniert → UserPages
UserPages	Seiten die gedruckt werden sollen	String (Aufzählung der Seiten wie im Dialog)

Dabei definieren sich die Formate wie folgt:

Format	Erklärung
Boolean	Wahrheitswert: True oder False
Integer	Ganze Zahl
String	Zeichenfolge
Array of xy	Liste von Werten vom Typ xy. Diese werden mit dem Parameter "Index" indiziert.
Color	RGB -Farbwert, ganze Zahl
Font	Schriftart: Eine Zeichenfolge mit folgendem Aufbau: <Schriftart> (<Größe>) [<Farbe>], Stil: <Stil> Z. B.: "Arial (14) [WindowText], Stil: fett"
0 - …, 1 - …	Mögliche Werte sind die ganzen Zahlen. Dahinter steht in der obigen Tabelle ihre Bedeutung. Das Verhalten von CONVAL im Falle anderer Werte ist **undefiniert**!

In einigen Fällen finden Sie in der mittleren Spalte weitere Erläuterungen.

Tag

```
property Tag: Variant; (r/w)
```

Dient zur Speicherung beliebiger Informationen.

Application

```
property Application: IConval12; (r/o)
```

Gibt das Anwendungsobjekt zurück.

Parent

```
property Parent: Variant; (r/o)
```

Diese Eigenschaft gibt das übergeordnete Objekt zurück. In diesem Fall ist es identisch mit dem Anwendungsobjekt *Application.*

OptionCount

```
property OptionCount: Integer; (r/o)
```

Diese Eigenschaft gibt die Anzahl der verfügbaren Optionen zurück.

Options

```
property Options[Index: Integer]: String; (r/o)
```

Diese Eigenschaft liefert die Namen der mit *Index* indizierten Optionen. Der Index läuft von 1 bis *OptionCount*.

GetOption

```
function GetOption(OptionName: String): Variant;
```

Diese Funktion liefert die aktuelle Einstellung der Option *OptionName*.

SetOption

```
procedure SetOption(OptionName: String; Value: Variant);
```

Diese Prozedur setzt den Wert der Option *OptionName*. Die Namen und möglichen Werte entnehmen Sie bitte der obigen Tabelle.

Die Typbibliothek in IDL

Die genaue Definition der Typbibliothek finden Sie in der Datei *COMConval9.idl*. Sie ist normalerweise nicht relevant, da die meisten Programmiersprachen nach einer Einbindung einer Schnittstelle entsprechende Hilfen zur Verfügung stellen.

Parameterlisten

Sie können mit dem Script *Parameterliste.xls* selber Parameterlisten der einzelnen Module erzeugen. Diese sind dann konform mit der aktuell registrierten Version des COM-Servers.

Übersicht

IConval12

```
property Tag: Variant; (r/w)
procedure Show;
procedure Hide;
property Visible: Boolean; (r/o)
property CalculationKindCount: Integer; (r/o)
property CalculationKinds[Index: Integer]: String; (r/o)
function IndexOfCalculationKind(const Index: String): Integer;
function NewDialog(CalcKind: String): ICalculationDialog;
property DialogCount: Integer; (r/o)
property Dialogs[Index: Integer]: ICalculationDialog; (r/o)
function NewCalculation(CalcKind: Integer): ICalculation;
property CalculationCount: Integer; (r/o)
property Calculations[Index: Integer]: ICalculation; (r/o)
property Units: IUnits; (r/o)
property Signis: ISignis; (r/o)
property Templates: ITemplates; (r/o)
property MainOptions: IMainOptions; (r/o)
property PrintOptions: IPrintOptions; (r/o)
function IsCV7File(FileName: string): Boolean;
function LoadCVFile(FileName: string): ICVFile;
function FileOfName(FileName: string; ForceLoad: Boolean): ICVFile;
property CVFileCount: Integer; (r/o)
property CVFiles[Index: Integer]: ICVFile; (r/o)
function NewCVExport: ICVExport;
property CVExportCount: Integer; (r/o)
property CVExports[Index: Integer]: ICVExport; (r/o)
property Concentrations: IConcentrations; (r/o)
property WantEvents: Boolean; (r/w)
property MuteExceptions: Boolean; (r/w)
property ActiveLanguage: Integer; (r/w)
property LanguageCount: Integer; (r/o)
property LanguageNames[Index: Integer]: String; (r/o)
property DecimalSeparator: Char; (r/w)
property ThousandSeparator: Char; (r/w)
property ShortDateFormat: String; (r/w)
property LongDateFormat: String; (r/w)
function FindValve([inout] VManufacturer: string; [inout] VSeries: string;
     VSize, VSeat, VStroke: sring; VCv: double; VChar: Integer; [out]
     Valve: string; WantDialog, WantANSI: Boolean): Boolean;
function FindValveEx(VManufacturer, VSeries, VSize, VSeat, VStroke: sring;
     VCv: double; VChar: Integer; [out] Valve: string; [inout] Ref1, Ref2:
     string; WantDialog, WantANSI, ForceDialog: Boolean): Boolean;
property Version: String; (r/o)
property Build: String; (r/o)
property Edition: String (r/o)
property AlwaysInFront: Boolean (r/w)
```

```
property Linebreak: string (r/w)
property Instances: Integer (r/o)
property StayAlive: Boolean; (r/w)
property RestartServerCalls: Integer; (r/w)
procedure Exit;
```

Events

```
event Exception(Message: String);
event BeforeCloseDialog(ASender: ICalculationDialog;
                        [inout] CanClose: Boolean);
event BeforeCloseCalculation(ASender: ICalculation);
event BeforePrint(ASender: ICalculationDialog; [inout] CanPrint: Boolean);
event AfterPrint(ASender: ICalculationDialog);
event BeforeParamChange(ASender: IParameter);
event AfterParamChange(ASender: IParameter);
event BeforeOpen(ASender: Variant;
                 [inout] FileName: String;
                 [inout] CanOpen: Boolean);
event AfterOpen(ASender: Variant; FileName: String);
event BeforeSave(ASender: Variant;
                 [inout] FileName: String;
                 [inout] CanSave: Boolean);
event AfterSave(ASender: Variant; FileName: String);
```

ICalculationDialog

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
procedure Show;
procedure Hide;
property Visible: Boolean; (r/w)
property Left: Integer; (r/w)
property Top: Integer; (r/w)
property Width: Integer; (r/w)
property Height: Integer; (r/w)
procedure SetWindowPos(Left, Top, Width, Height: Integer);
procedure ExpandAll;
procedure CollapsAll;
procedure WindowMaximize;
procedure WindowMinimize;
procedure WindowRestore;
property AlwaysInFront: Boolean (r/w)
property Calculation: ICalculation; (r/o)
property Characteristics: ICVCharacteristics;
property CalculationCount: Integer;
property Calculations[Index: Integer]: String; (r/o)
procedure SelectCalculation(Ident: Variant);
property CalculationKind: Integer; (r/o)
procedure New;
procedure NewFromTemplate(FileName: string; IsPrivate: Boolean);
procedure OpenDialog;
```

```
procedure Open(FileName: String; WantRecover: Boolean);
procedure Save;
procedure SaveAs(FileName: String; WantOverwrite: Boolean);
procedure SaveAsTemplate(FileName: string; IsPrivate: Boolean);
procedure CloseCalculation(WantOverwrite: Boolean; WantSave: Boolean);
procedure CloseAll(WantOverwrite: Boolean; WantSave: Boolean);
property HasChanged: Boolean; (r/o)
procedure ImportCV4;
procedure PrintDialog;
procedure Print(WantDialog: Boolean);
procedure Preview(WantDialog: Boolean);
procedure CreatePDF(FileName: String)
procedure CalculationExport(Format: string);
procedure SendMail(WantCalculation: Boolean);
procedure Undo;
procedure ModuleFunc(FuncName: string);
property TreeVisible: Boolean; (r/w)
property MessagesVisible: Boolean; (r/w)
property ToolbarVisible: Boolean; (r/w)
property HelpVisible: Boolean; (r/w)
property Assigned: Boolean; (r/o)
procedure Close;
```

Events

```
event BeforeClose([inout] CanClose: Boolean);
event BeforeVisibleChange;
event AfterVisibleChange;
event BeforePrint([inout] CanPrint: Boolean);
event AfterPrint;
event BeforeOpen([inout] FileName: String; [inout] CanOpen: Boolean);
event AfterOpen(FileName: String);
event BeforeSave([inout] FileName: String; [inout] CanSave: Boolean);
event AfterSave(ASender: Variant; FileName: String);
```

ICalculation

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property CalculationData: ICalculationData; (r/o)
property OptionalData: IOptionalData; (r/o)
property Data: Variant; (r/w)
property CalculationKind: Integer; (r/o)
procedure Clear;
procedure New;
function Open(FileName: String): Boolean;
function Save: Boolean;
function SaveAs(FileName: String): Boolean;
procedure BeginUpdate;
procedure EndUpdate;
```

```
property MessageCount: Integer; (r/o)
property Messages[Index: Integer]: String; (r/o)
property MessageClasses(Index: Integer): ENumMessageType; (R/O)
property MessageID(Index: Integer): Integer; (R/O)
property MessageParam(Index: Integer): string; (R/O)
function HasMessage(ID: Integer): Boolean;
property FileExtension: String; (r/o)
property ModuleName: String; (r/o)
property CV4FileName: String; (r/o)
procedure Calculate;
property Active: Boolean; (r/w)
property IsCalculated: Boolean; (r/o)
property FileName: String; (r/w)
function GetResistors: String;
function GetRefPressure(CalcUnit: Variant): Double;
procedure SetRefPressure(CalcUnit: Variant; Value: Double); (r/w)
function GetPressureValue(ParamName: string; ACalcUnit: Variant; IsGage:
    Boolean): Double;
procedure SetPressureValue(ParamName: string; Value: Double; ACalcUnit:
    Variant; IsGage: Boolean);
procedure ModuleFunc(FuncName: string);
procedure ModuleFuncList(): string;
property CVFile: ICVFile;
function LoadUnitScheme(const FileName: String;
                        const IsPrivate: Boolean): Boolean;
function LoadSigniScheme(const FileName: String;
                         const IsPrivate: Boolean): Boolean;
property Assigned: Boolean; (r/o)
procedure Close;
```

Events

```
event BeforeClose(ASender: ICalculation);
event BeforeClear([inout] CanClear: Boolean);
event BeforeParamChange(ASender: IParameter);
event AfterParamChange(ASender: IParameter);
event BeforeOpen([inout] FileName: String; [inout] CanOpen: Boolean);
event AfterOpen(FileName: String);
event BeforeSave([inout] FileName: String; [inout] CanSave: Boolean);
event AfterSave(FileName: String);
event BeforeCalculate;
event AfterCalculate;
event NoteChange(ASender: IParameter);
```

ICalculationData

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property ParamCount: Integer; (r/o)
property Params[Index: Integer]: IParameter; (r/o)
property ParamByName[Name: String]: IParameter; (r/o)
```

```
property IndexOfParam[Name: String]: Integer; (r/o)
property ParamList: string;
property WantOrderedList: Boolean; (r/w)
```

IOptionalData

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property ParamCount: Integer; (r/o)
property Params[Index: Integer]: IParameter; (r/o)
property ParamByName[Name: String]: IParameter; (r/o)
property IndexOfParam[Name: String]: Integer; (r/o)
function ClearParam(Index: Integer): Boolean;
function DeleteParam(Index: Integer): Boolean;
function AddParam(Name, Caption: String;
    ParamType: ENumOptionalDataParamKind): Boolean;
function ChangeParam(Index: Integer; Name, Caption: String;
    ParamType: ENumOptionalDataParamKind): Boolean;
function MoveParam(SourceIndex, DestIndex: Integer): Boolean;
function ClearAll: Boolean;
function DeleteAll: Boolean;
property ParamList: string;
function AddTemplate(TemplateName: String; IsPrivate: Boolean): Boolean;
procedure DeleteTemplate(FileName: String; IsPrivate: Boolean);
procedure SaveAs(FileName: String; IsPrivate: Boolean);
```

ICVCharacteristics

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property ParamCount: Integer; (r/o)
property Params[Index: Integer]: IParameter; (r/o)
property ParamByName[Name: String]: IParameter; (r/o)
property IndexOfParam[Name: String]: Integer; (r/o)
property ParamList: string; (r/o)
procedure Refresh;
procedure SetStroke(Stroke: Double);
procedure ParamOfStroke(Stroke: Double; ParamName: String): Double;
function StrokeOfQm(Qm: Double): Double;
function ParamOfQ(Q: Double; ParamName: String): Double;
procedure SaveGraphic(FileName: String; GraphIndex: Integer; Width, Height:
Integer);
```

IParameter

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property Assigned: Boolean; (r/o)
property Name: String; (r/o)
property ParamType: ENumParamType; (r/o)
```

```
property ODParamType: ENumOptionalDataParamKind; (r/o)
property InputState: ENumInputState; (r/o)
property DisplayState: ENumDisplayState; (r/o)
property ValueState: ENumValueState; (r/o)
property ReadOnly: Boolean; (r/w)
property Enabled: Boolean; (r/w)
property IsOptional: Boolean (r/o)
property IsCalculated: Boolean; (r/o)
property IsCalculatedOverwritten: Boolean; (r/o)
property IsLookedUp: Boolean (r/o)
property IsLookedUpOverwritten: Boolean (r/o)
property CanBeOverwritten: Boolean (r/o)
property IsOverwritten: Boolean; (r/o)
procedure Restore;
property LongCaption: String; (r/o)
property ShortCaption: String; (r/o)
property Value: Double; (r/w)
property SIValue: Double; (r/w)
property DiaplayValue: Double; (r/o)
property CalcUnit: ICalcUnit; (r/o)
property Text: String; (r/w)
property DisplayText: String; (r/o)
property IsEmpty: Boolean; (r/w)
property SwitchStateCount: Integer; (r/o)
property SwitchStateNames[Index: Integer]: String; (r/o)
property SwitchState: Integer; (r/w)
property VisibleSwitchStateNames: String; (r/o)
property Message: String; (r/o)
property MessageID: Integer; (r/o)
property MessageClass: ENumMessageClass; (r/o)
property HasMessage: Boolean; (r/o)
property Note: String; (r/w)
property Comment: String; (r/o)
property DetailLevel: ENumDetailLevel; (r/w)
```

ICalcUnit

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property Quantity: Integer; (r/o)
procedure SetQuantity(Value: Integer);
property QuantityName: String; (r/o)
procedure SetQuantityName(Value: String);
property UnitCount: Integer; (r/o)
property UnitNames[Index: Integer]: String; (r/o)
property CurrentUnit: Integer; (r/w)
property UnitName: String; (r/w)
```

IUnits

```
property Tag: Variant; (r/w)
```

```
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property QuantityCount: Integer; (r/o)
property QuantityNames[Index: Integer]: String; (r/o)
property UnitCount[QuantityIndex: Integer]: Integer; (r/o)
property UnitNames[QuantityIndex: Integer; Index: Integer]: String; (r/o)
property ConvertFactor[QuantityIndex: Integer; UnitIndex: Integer]: Double;
(r/o)
property ConvertTerm[QuantityIndex: Integer; UnitIndex: Integer]: Double;
(r/o)
property DefaultUnit[QuantityIndex: Integer]: Integer; (r/w)
function GetSchemeCount(TempScope: ENumTempScope): Integer;
function GetSchemes(Index: Integer;
                    PrivateMarker: ENumPrivateMarker;
                    TempScope: ENumTempScope): String;
property DefaultScheme: String; (r/w)
function LoadScheme(const FileName: String): Boolean;
function SaveScheme(const FileName: String): Boolean;
function DeleteScheme(const FileName: String): Boolean;
```

ISignis

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property QuantityCount: Integer; (r/o)
property QuantityNames[Index: Integer]: String; (r/o)
property Signi[QuantityIndex: Integer]: Integer; (r/w)
property SchemeCount: Integer; (r/w)
property SchemeNames[Index: Integer]: String; (r/w)
property DefaultScheme: String; (r/w)
function LoadScheme(const FileName: String): Boolean;
function SaveScheme(const FileName: String): Boolean;
function DeleteScheme(const FileName: String): Boolean;
```

ITemplates

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property TypeCount: Integer; (r/o)
property Types[Index: Integer]: string; (r/o)
property TypeExtensions[Index: Integer]: string; (r/o)
function GetSchemeCount(TemplateType: Variant;
                        TempScope: ENumTempScope): Integer;
function GetSchemeName(TemplateType: Variant;
                       Index: Integer;
                       PrivateMarker: ENumPrivateMarker;
                       TempScope: ENumTempScope): string;
function GetSchemeName(TemplateType: Variant;
                       Index: Integer;
```

```
                        TempScope: ENumTempScope;
                        WantPath: Boolean): string;
function GetIdentFromFile(AFileName: string;
                          TempScope: ENumTempScope): string;
function GetSchemeNames(TemplateType: Variant;
                        PrivateMarker: ENumPrivateMarker;
                        TempScope: ENumTempScope): string;
function GetSchemeNames(TemplateType: Variant;
                        TempScope: ENumTempScope;
                        WantPath: Boolean): string;
```

ICVFile

```
function Load(FileName: WideString): Boolean;
function Save(FileName: String): Boolean;
property FileName: String; (r/o)
function GetParamAttribute(ParamName, AttributeName: String): String;
function CalculationKind: Integer;
procedure Close;
function ItemList(Path, Divider: String): String;
function AttributeList(Path, Divider: String): String;
function AttributeValue(Path, AttributeName: String): String´;
```

ICVExport

```
procedure AddFileByName(FileName: String);
procedure DelFileByName(FileName: String);
procedure ClearFileList;
property TemplateName: String; (r/w)
procedure AddFile(AFile: ICVFile);
procedure DelFile(AFile: ICVFile);
property CVFiles[Index: Integer]: ICVFile; (r/o)
property CVFileCount: Integer; (r/o)
procedure DoExport(FileName: String);
```

IConcentration

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
function GetSchemeCount(TempScope: ENumTempScope): Integer; (r/o)
function GetSchemeName(Index: Integer;
                       PrivateMarker: ENumPrivateMarker;
                       TempScope: ENumTempScope): String;
function GetSchemeIsPrivate(Index: Integer): Boolean;
function LoadScheme(const Name: String; IsPrivate: Boolean);
function SaveScheme(const Name: String; IsPrivate: Boolean);
function DeleteScheme(const FileName: String; IsPrivate: Boolean);
property AGA8: Boolean; (r/w)
property Name: String; (r/w)
property IsPrivate: Boolean; (r/w)
property ItemCount: Integer; (r/o)
property ItemName[Index: Integer]: String; (r/o)
property ItemCaption[Index: Integer]: String; (r/o)
```

```
property ItemValue[Index: Integer]: Double; (r/w)
property IsAGA8: Boolean;
property IsGERG: Boolean;
property ConcentrationType: Integer; (r/w)
property Sum: Double; (r/o)
procedure Normalize;
property LastUsed: Double; (r/o)
```

Optionen IMainOptions, IPrintOptions

```
property Tag: Variant; (r/w)
property Application: IConval12; (r/o)
property Parent: Variant; (r/o)
property OptionCount: Integer; (r/o)
property Options[Index: Integer]: String; (r/o)
function GetOption(OptionName: String): Variant;
procedure SetOption(OptionName: String; Value: Variant);
```

Tabellen

Berechnungen

Nr.	*Berechnung*
1	Stellventil
2	Begrenzungslochscheibe
3	Wirkdruckgeber
4	Rohrleitungskompensation
5	Druckverlust
6	Druckstoß
7	Dimensionierung
8	Stützweiten
9	Rohrwanddicke
10	Sicherheitsventil
11	Behälterentspannung
12	Rohrbündelwärmetauscher
13	Stoffdatenberechnung
14	Regression
15	Kondensator
16	Pumpen- und Verdichterleistung
17	Antriebskräfte
18	Dampfumformventil
19	Stellventil (Zweiphasig)
20	Werkstoffdatenberechnung
21	Thermodynamik-Modul
22	Berstscheiben
23	Thermometerschutzrohre
24	Füllstandsmessung
25	Automatisierte Armatur
26	Dampfkühlventil

Größen

Nr.	*Quantity*
1	Temperatur
2	Temperaturdifferenz
3	Druck1
4	Druck2
5	Kraft
6	Dichte
7	Spezifisches Volumen
8	Dynamische Viskosität
9	Kinematische Viskosität
10	Massenstrom
11	Volumenstrom
12	Normvolumenstrom
13	Kv bzw. Cv
14	Nennweite
15	Geschwindigkeit
16	Länge1
17	Länge2
18	Ausdehnungskoeffizient
19	Elastizität
20	Fläche1
21	Fläche2
22	Fläche3
23	Masse

Nr.	*Quantity*
24	Volumen
25	Masse/Länge
26	Volumen/Länge
27	Leistung
28	Wärmeleistung
29	Wärmekapazität
30	Entropie
31	Enthalpie
32	Wärmeleitfähigkeit
33	Wärmedurchgangswiderstand
34	Wärmedurchgangskoeffizient
35	Wärmestromdichte
36	Spezifische Gaskonstante
37	Molare Masse
38	Dipolmoment
39	Schallpegel (A-bewertet)
40	Schallpegel (unbewertet)
41	Frequenz
42	Anteile
43	Zeit 1
44	Keine Einheit
45	Keine Einheit, Positiv
46	Ganzzahliger Wert [0,1,2,...]
47	Datum und Uhrzeit
48	Keine Einheit, [0..1]
49	Abs. Feuchte
50	Spez. Feuchte
51	Partielle Ableitung (dp/dt)rho
52	Joule-Thomson Koeffizient
53	Partielle Ableitung (dp/drho)t
54	Partielle Ableitung (drho/dt)rho
55	Berieselungsdichte
56	F-Faktor
57	Druckverlust
58	Isothermischer Drosselkoeffizient
59	Zweiter Virialkoeffizient
60	Dritter Virialkoeffizient
61	Molare Dichte
62	Molares Volumen
63	Molare Entropie
64	Molare Enthalpie
65	Molare Wärmekapazität
66	Molarer Isothermer Drosselkoeffizient
67	Molarer zweiter Virialkoeffizient
68	Molarer dritter Virialkoeffizient
69	Mol. part. Ableitung (drho/dt)p
70	Mol. part. Ableitung (dp/drho)T
71	Oberflächenspannung
72	Kompressibilität
73	Zeit 2
74	Druckdifferenz 1
75	Druckdifferenz 2
76	Massenfluss
77	Winkel
78	Stromstärke

Nr.	*Quantity*
79	Indexwert
80	Drehmoment
81	Zeit 3