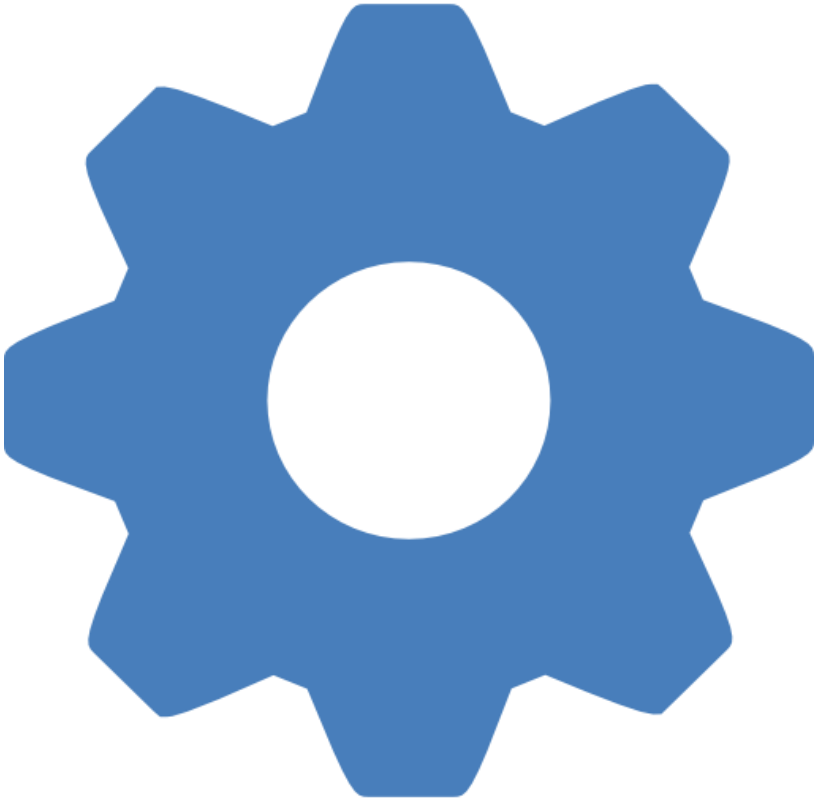


mdsScript – Scripting Engine

1. Inhaltsverzeichnis

Datenstand: 17. 08. 2017

Hinweis:

Einige Icons in diesem Dokument und in der mdsScript-Bibliothek wurden erstellt von Freepik auf der Webseite www.flaticon.com.

Some icons in this document and the mdsScript application were made by Freepik from www.flaticon.com.

2. Einleitung, Voraussetzungen und Haftung

Einleitung

Oft werden Programme entwickelt, für die sich eine Funktionserweiterung über Skripte anbieten würde. Leider ist diese aber oft so aufwändig zu programmieren, dass der Entwickler diese nicht implementiert.

Hier kann „mdsScript“ helfen, denn es bietet neben einer leicht erlernbaren Programmiersprache, die an Basic angelehnt ist (ähnlich zu VBA), auch die Möglichkeit, eigene .NET-Programmfunktionen aus den Skripten aufzurufen. Die Einbindung in eigene Projekte ist einfach, so dass mit wenig Aufwand ein großer Nutzen erreicht werden kann.

Voraussetzungen und Nutzungsbedingungen

„mdsScript“ kann in alle Entwicklungsprojekte eingebunden werden, die unter MS Windows mit .NET erstellt werden. Die vorliegende Programmversion baut auf das .NET-Framework Version 4.0 auf und wurde mit „MS Visual Studio 2015“ entwickelt. Zur Nutzung benötigen Sie daher eine Entwicklungsumgebung (z.B. „MS Visual Studio“) sowie das .NET-Framework in der Version 4.0. Beide können als freie Versionen bei Microsoft heruntergeladen werden.

Die „mdsScript“-Bibliothek ist so implementiert, dass sie ohne Lizenzdateien in eigene Programme eingebunden werden kann. Das heißt aber nicht, dass die Nutzung frei ist. Die detaillierten Nutzungsbedingungen finden Sie auf der Webseite www.ms-mds.de.

Haftung

Wir haben die Scripting-Engine „mdsScript“ sorgfältig programmiert und getestet. Allerdings kann bei der Komplexität derartiger Programme nicht mit absoluter Sicherheit garantiert werden, dass das Programm keine Fehler enthält. Sollten Programmfehler auftreten, so werden wir uns nach bestem Wissen bemühen, diese schnellstmöglich zu beheben.

Ein Haftungsanspruch besteht nur bei grob fahrlässiger Herbeiführung unsererseits und maximal in der Höhe des Kaufpreises dieses Programmes.

Eine Eignung der Scripting-Engine für spezielle Zwecke kann vorher mit Hilfe einer Testversion überprüft werden.

3. Installation mdsScript

Zur Installation von „mdsScript“ kopieren Sie den Inhalt des Archivs „mdsScript.zip“ in ein leeres Verzeichnis.

Sie finden danach im Verzeichnis die folgenden Dateien und Verzeichnisse:

- mdsScript.dll — Die eigentliche Bibliothek – diese Datei muss auch bei Ihren Projekten in das Installationspaket eingefügt werden.
- mdsScript.pdb — Debug-Informationen zu mdsScript.dll – diese Datei darf nicht mit Ihren Projekten vertrieben werden.
- mdsScript.txt — Informationen zu „mdsScript“ – diese Datei muss auch bei Ihren Projekten in das Installationspaket eingefügt werden. Es muss im gleichen Verzeichnis gespeichert sein, in dem auch mdsScript.dll abgelegt wird.
- Beispiele - Examples — Verzeichnis mit Beispiel-Skripten
- mdsScript-Demo — Beispielanwendung zur Benutzung von „mdsScript“ – mit dieser Anwendung können auch die Beispieldateien ausgeführt werden.

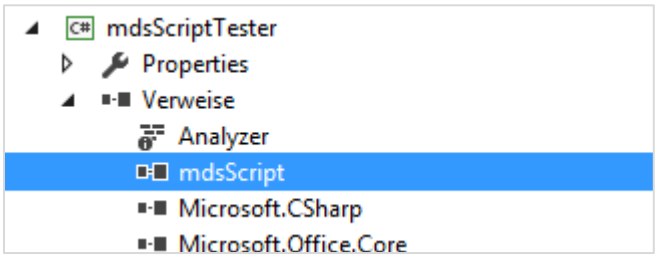
In Ihren Projekten müssen Sie einen Verweis auf die Datei „mdsScript.dll“ hinzufügen. Details dazu finden Sie im folgenden Kapitel.

Die mdsScript-Demo zeigt die unterschiedlichen Aufruf-Varianten (siehe unten) und Beispiele für die Interaktion mit externen Funktionen. Dazu ist es hilfreich, die Beispiel-Skripte mit dem Programm „msScript-Demo“ zu laden und im Debugger schrittweise zu durchlaufen.

4. Benutzung der Scripting-Engine

Skripte werden in einer Textdatei oder in einer String-Liste bereitgestellt. In jedem Skript muss ein Unterprogramm „Main“ vorkommen, welches beim Starten des Skripts angesprungen wird.

Um die Scripting-Engine im eigenen .NET-Code nutzen zu können, wird zuerst dem eigenen Projekt ein Verweis auf die Assembly `mdsScript` hinzugefügt:



Dazu klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf den Eintrag „Verweise“ und wählen dort „Verweis hinzufügen…“. Im Verweis-Manager wählen Sie unten den Button „Durchsuchen…“ und wählen danach die Datei „mdsScript.dll“ im Installationsverzeichnis aus.

Mit dem folgenden Programmcode wird eine Skriptdatei im Debugger gestartet:

```
// Instanz der Scripting Engine erstellen
mdsScript.ScriptingEngine se = new mdsScript.ScriptingEngine();
// Skript „TestSkript.txt“ ohne Variablenwerte im Debugger starten
string sRC = se.RunScript("d:\TestSkript.txt", null, true, null);
```

Soll ein Skript aus einer String-Liste (`List<string>`) ausgeführt werden, wird diese vor dem Aufruf der Methode `RunScript` mit den Sourcetextzeilen gefüllt. Allerdings ist dann kein `INCLUDE`-Befehl mehr möglich. Ein Ausgangspunkt für eigenen Code ist:

```
// Instanz der Scripting Engine erstellen
mdsScript.ScriptingEngine se = new mdsScript.ScriptingEngine();
List<string> ls_Source = new List<string>();
// ... Laden des Sourcetextes in „ls_Source“ ...
// Sourcetext in „ls_Source“ ohne Variablenwerte im Debugger starten
string sRC = se.RunScript(ls_Source, null, true, null);
```

Ist das Debuggen nicht notwendig, so wird der dritte Parameter auf „false“ gesetzt:

Sollen globale Variablen vor dem Starten des Skriptes geändert werden, so muss eine String-Liste (`List<string>`) gefüllt werden – jeder Eintrag der Liste enthält die Zuordnung in der Form

```
<globalerVariablenname>=<neuerWert>
```

Diese Liste wird als Parameter 2 (`lsGlobalVars`) an die `RunScript`-Methode übergeben.

Die Parameter der Methode `RunScript`:

Nr.	Parameter	Datentyp	Beschreibung

1	`sScriptFile`
	oder
	`lsSourcetext`
		string
		oder
		List<string>
			Voll qualifizierter Name der Skript-Datei
			oder
			String-Liste mit dem gesamten Sourcetext des Skripts (es sind keine `Include`-Befehle möglich)

2	`lsGlobalVars`	List<string>	Liste von Strings mit den einzelnen Zuordnungen der globalen Variablen. Jeder Eintrag hat die Form <var name>=<var value> Es kann „null“ übergeben werden, wenn keine globalen Variablen gesetzt werden sollen.
3	`bDebug`	bool	true: Starten des Skriptes im Skript-Debugger
4	`lstOutput`	bool	ListBox Element, wenn die Ausgaben in einem Fenster der eigenen Anwendung gezeigt werden sollen (nur bei `bDebug = false`). Ist dieser Parameter nicht „null“, dann wird kein separates Ausgabefenster geöffnet.

Rückgabe von Ergebnissen eines Skriptes

Rückgabedaten an das aufrufende Programm werden über eine String-Liste realisiert, auf die vom aufrufenden Programm nach Ausführung des Skriptes über die Funktion `GetResultData` zugegriffen werden kann. Diese wird beim Aufruf des Skriptes geleert.

Der Fehlerstatus des Skriptes kann mit der Funktion `GetError` ausgelesen werden. Dieser ist leer, wenn kein Fehler vorliegt – anderenfalls enthält die Funktion den Fehlertext. Die gleiche Information wird auch von der Methode `RunScript` zurückgegeben!

Bei der Funktion `GetOutputData` wird eine ListBox zurückgegeben. Diese enthält alle PRINT-Ausgabezeilen des Skripts.

Implementieren von eigenen Skript-Erweiterungen

Die Skript-Engine kann durch eigene Funktionen und Methoden erweitert werden. Dazu ist z.B. der folgende Code anzugeben:

```
// Instanz der Scripting Engine erstellen
mdsScript.ScriptingEngine se = new mdsScript.ScriptingEngine();
// Eventprozedur einbinden
se.ExternalFunctionEvent += DoExternalFunction;
// Aufruf des Skriptes
string sRC = se.RunScript ("d:\TestSkript.txt", null, true, null);
```

Durch die zweite Zeile wird eine Eventprozedur definiert, über welche die Eigenprogrammierung eingebunden werden kann. Diese Prozedur könnte dann sein:

```
private void DoExternalFunction
    (object sender, mdsScript.ExternalFunctionEventArgs e)
{
    // Der Funktionsname steht in  e.FunctionName
    // Die Parameterliste steht in e.ParmsList

     // Hier steht der eigene Programmcode ...
}
```

Die Datenstruktur `mdsScript.ExternalFunctionEventArgs` besteht aus zwei Komponenten:

- Funktionsname: `string FunctionName;`
- Liste der Parameter: `List<string> ParmsList;`

Damit kann die aufzurufende Funktion identifiziert werden und es werden die übergebenen Parameter bereitgestellt

Zur Interaktion der Script Engine-Erweiterung stehen noch drei Funktionen zur Verfügung, die über die Instanz der Script Engine aufgerufen werden können:

`<ScriptingEngine>.ExtFunc_SetReturnCode(<ergebnis>,<error>)`

Setzt eine Ergebniszeichenkette als Rückgabewert der externen Funktion und/oder einen Fehlerwert.

`<ScriptingEngine>.ExtFunc_WriteOutputText(<text>,<newline>)`

Schreibt einen Text in das Skript-Ausgabefenster. Wird `<newline>` auf „0“ gesetzt, so wird die letzte Ausgabezeile durch den neuen Wert ersetzt. Bei anderen Werten wird eine neue Ausgabezeile angefügt.

`<ScriptingEngine>.ExtFunc_WriteResultText(<text>)`

Schreibt einen Text (wie in dem Befehl `PRINTRESULT`) in die Ergebnis-Daten des Skriptes.

```
<ScriptingEngine>.ExtFunc_WriteFileText(<file>,<text>,<newline>)
```

Der Text `<text>` wird in die Datei mit der Nummer `<file>` geschrieben. Wenn `<newline>` ungleich „0“ ist, erfolgt nach dem Text ein Zeilenvorschub.

```
<ScriptingEngine>.ExtFunc_SetGlobalVar(<variable>,<value>)
```

Setzt den Wert einer globalen Skript-Variablen

```
x = <ScriptingEngineInstanz>.ExtFunc_GetGlobalVar(<variable>)
```

Liest eine globale Skript-Variable

Innerhalb des Skriptes werden die Skripterweiterungen über einen Funktions- oder Prozedurnamen aufgerufen, der nicht als interne Funktion verwendet wird und auch nicht als Skript-Funktion oder -Prozedur genutzt wird. Es empfiehlt sich bei intensiver Nutzung von Skript-erweiterungen ein zweiteiligen Funktions- oder Prozedurname. Die beiden Teile des Namens werden durch einen Punkt getrennt, so dass z.B. über den ersten Namensteil die Erweiterungen gruppiert werden können:

```
A = XLS.ReadCell(5, 2)
```

könnte eine Excel-Zelle „B5“ lesen. Eine eigene Test-Methode könnte über

```
Call Test.Step1("Hallo")
```

oder

```
Call Test.Step1
```

aufgerufen werden.

Wichtig!

Funktionen müssen mindestens einen Parameter haben. Dieser kann aber in der eigen-programmierten Funktion ignoriert werden.

Methoden (Aufruf durch CALL) benötigen keinen Parameter.

5. Technische Beschreibung

In diesem Kapitel werden die möglichen Befehle eines Skriptes beschrieben. Die Syntax lehnt sich an das in den Microsoft Office-Programmen verwendete VBA (Visual Basic for Applications) an.

Aufbau eines Skripts

Ein Skript besteht aus

- Globalen Deklarationen (falls benötigt)
- SUB Main (muss vorhanden sein als Programm-Startpunkt)
- Weiteren SUBs (optional)
- FUNCTIONs (optional)
- INCLUDE-Zeilen (über die Codeteile aus anderen Dateien verwendet werden können)

Die genaue Syntax der einzelnen Befehle wird weiter unten beschrieben. Dabei ist eine rekursive Nutzung von Prozeduren (SUB) nicht erlaubt. Funktionen (FUNCTION) können aber rekursiv verwendet werden.

Beispiel:

```
' Globale Variable
DIM a = 100

' Start-SUB
SUB Main
    ' Lokale Variable "i"
    DIM i
    FOR i = 1 TO a
        PRINT i, XFunc(i), i^2
    NEXT
END SUB

' Skript-Funktion
FUNCTION XFunc(i)
    XFunc = "(" & i & ")"
END FUNCTION
```

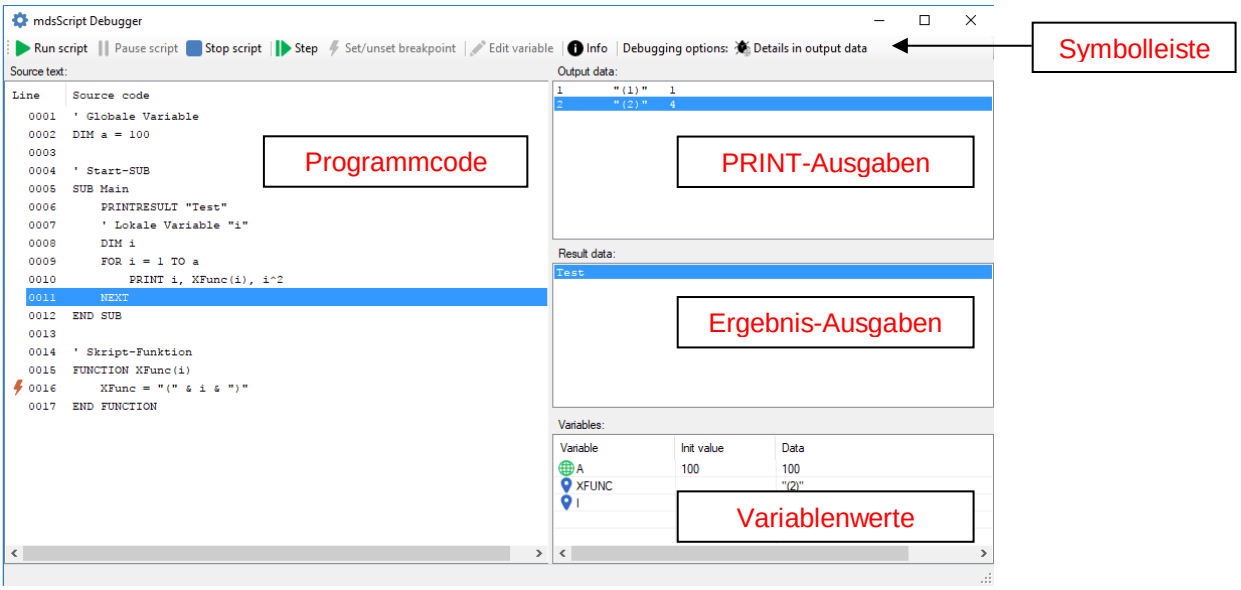
Über eine INCLUDE-Zeile können Teile eines Skriptes in andere Dateien ausgelagert werden:

```
INCLUDE include1.txt
```

Dies fügt den Inhalt der Datei „include1.txt“ an der Stelle des Include-Befehls ein. Die im Befehl angegebene Datei muss im gleichen Verzeichnis gespeichert sein, in der auch das gestartete Skript abgelegt ist (INCLUDE-Befehle sind nur beim Starten einer Skript-Datei möglich!).

Debugging

Der Skript-Debugger zeigt das folgende Fenster:



Die Buttons der Symbolleiste starten die Aktionen des Debuggings. Sie können meistens auch durch das Drücken einer Funktionstaste aufgerufen werden. Je nach aktiver Codezeile oder selektierter Variable sind nicht alle Buttons anwählbar:

- Run script [F5]: Ausführen des Skriptes (ab der aktuellen Position)
- Pause script [F6]: Unterbrechen des Skriptes an der aktuellen Position
- Stop script [F7]: Beenden des Skriptes
- Step [F8]: Ausführen des nächsten Befehls im Skript
- Set/unset breakpoint [F9]: Setzen/Löschen eines Breakpoints (wird durch einen roten Blitz vor der Code-Zeile angezeigt; siehe oben in Zeile 0016)
- Edit variable [F10]: Verändern des Wertes der selektierten Variable
- Info: Information über „mdsScript“
- Details in output data: Wenn dieser Button selektiert ist (Rahmen um den Button), werden die Befehle im Bereich „Output data“ vor der Ausführung ausgegeben, so dass bei einem Programmfehler die Fehlerzeile genau adressierbar ist.

Die gerade aktive Codezeile wird vor der Ausführung selektiert angezeigt (das ist im Beispiel oben die Zeile mit der Nummer 0011).

Rechts werden untereinander die PRINT-Ausgaben, die Ausgaben über PRINTRESULT und die Variablen dargestellt (globalen Variablen wird ein grüner Globus als Symbol vorangestellt, lokalen Variablen ein blauer Positionsmarker). Die selektierte Variable kann durch Klick auf das Stift-Icon in der Symbolleiste verändert werden (nicht möglich bei Collection-Variablen).

Befehle

Für die Befehls-Schlüsselwörter und Variablen ist die Groß-/Kleinschreibung nicht von Belang. In dieser Dokumentation werden zum besseren Verständnis Schlüsselwörter immer in Großschreibung dargestellt, Variablen werden in Kleinschrift ausgegeben.

Es wird in einer Zeile immer genau ein Befehl geschrieben. Im Gegensatz zu anderen BASIC-Dialekten ist eine Verkettung von Befehlen mit einem Doppelpunkt nicht erlaubt. Dies erhöht die Übersichtlichkeit der Programme und die Verarbeitung von den Skriptzeilen wird beschleunigt. Auch Inline-Kommentare (Kommentartext hinter einem Befehl) sind somit nicht gültig.

Zahlen werden im Skript immer mit Dezimalpunkt geschrieben („2.5“, nicht „2,5“)!

Unterprogramme und Funktionen

```
SUB <unterprogramm-name>[(<parameter-1>,…,<parameter-n>)]
    …
    [EXIT SUB]
    …
END SUB
```

Zwischen den Befehlen `SUB` und `END SUB` werden die Skriptbefehle eines Unterprogramms angegeben. Der Name des Unterprogramms ist hinter dem ersten `SUB` angegeben. Die optionalen Parameter-Variablen werden hinter dem Unterprogrammnamen in Klammern und getrennt durch Kommata aufgezählt.

Stimmt die Anzahl der Parameter beim Aufruf nicht mit der Anzahl der Parameter in der Unterprogrammdefinition überein, so werden nicht angegebene Parameter auf einen Leerstring gesetzt. Überzählige Parameter werden ignoriert, d. h. Parameter sind grundsätzlich optional.

Wird innerhalb eines Unterprogramms der Befehl `EXIT SUB` aufgerufen, so wird das Unterprogramm sofort beendet und an die Stelle des Aufrufs zurück gesprungen.

Das Unterprogramm „Main“ hat eine besondere Funktion im Skript, denn es markiert den Startpunkt des Skriptes.

```
FUNCTION <function-name>(<parameter-1>[,…,<parameter-n>])
    …
    [EXIT FUNCTION]
    …
END FUNCTION
```

Funktionen sind eine spezielle Form von Unterprogrammen. Sie können in Berechnungsformeln verwendet werden und einen Rückgabewert ermitteln. Innerhalb der Funktion wird der Rückgabewert gesetzt, indem eine Zuweisung auf den Funktionsnamen stattfindet.

Funktionen müssen mindestens einen Parameter haben! Dieser kann aber innerhalb der Funktion ignoriert werden.

Variablendeklaration

Variablen müssen über den Befehl `DIM` deklariert werden. Alle Deklarationen außerhalb von `SUB`s und `FUNCTION`s werden als globale Variablen interpretiert, Deklarationen innerhalb von `SUB`s/`FUNCTION`s sind nur lokal in der `SUB`/`FUNCTION` gültig:

```
DIM <variablenname>[=<initialwert>]
```

Die Zuweisung des Initialwertes ist optional – Standardwert ist ein leerer String "". Es kann auch das Ergebnis einer Berechnung als Wert angegeben werden. Allerdings müssen bei der Verwendung von Variablen diese bereits vorher deklariert sein.

Variablen sind grundsätzlich typlos, d. h. es kann eine Variable im Laufe eines Skriptes zunächst numerisch verwendet werden, während ihr später Zeichenketten zugewiesen werden können. Gespeichert werden Variablen immer als Zeichenketten. Erst bei numerischen Berechnungen werden die Zeichenketten in Zahlen konvertiert (sofern möglich).

Variablen-Arrays im herkömmlichen Sinn werden von „mdsScript“ nicht unterstützt, jedoch ist es möglich, diese mit Hilfe von Collection-Variablen zu simulieren. Für diese beginnt der Variablenname mit dem Zeichen „@“. Ein Beispiel dafür wäre

```
DIM @array=0
```

Es wird eine Collection-Variable „`@array`“ angelegt, für die der Initialwert auf „0“ gesetzt wird. Collection-Variablen werden verwendet, indem hinter dem Namen der Variablen in Klammern eine „Werteadresse“ angegeben wird. Es können durch Komma getrennt mehrere (bis max. 8) Teile der Wertadresse angegeben werden, die jeweils berechnet sein können. Sie werden zu einem Gesamtschlüsselwert verkettet (getrennt durch Komma).Diese Teile können numerisch sein – das entspricht dann dem klassischen Array. Es können aber auch Strings genutzt werden – das entspricht dann einer Collection. Beim Lesen aus einem Array wird für nicht explizit angegebene Elemente der Initialwert zurückgegeben.

Beispiel für Collection-Variablen (die in einem Programm gleichzeitig verwendet werden können):

`@arr(2)`	→ 1-dimensionales Array (Key=“1“)
`@arr(2,3+4)`	→ 2-dimensionales Array (Key=“2,7“)
`@arr("Test")`	→ einfache Collection (Key=“Test“)
`@arr("Test1","Test2")`	→ geschachtelte Collection (Key=“Test1,Test2“)

Zurücksetzen von Variablen

```
CLEAR <variable>
```

Die Variable `<variable>` wird auf den Initialwert zurückgesetzt. Bei einer Collection-Variable werden alle gespeicherten Werte gelöscht.

Breakpoint und Programmende

Ein programmierter Breakpoint wird über die Skriptzeile

```
STOP
```

erzeugt. Wird das Skript im Debugger gestartet, so wird hier ein Breakpoint angenommen. In der normalen Skriptausführung wird dieser Befehl ignoriert.

```
END
```

Beendet das Programm.

Wertezuweisungen und Berechnungen

Um den Wert einer Variablen zu verändern, wird ein Zuweisungsbefehl angegeben. Der `<neue wert>` kann auch eine Berechnung sein:

```
<variablenname>=<neuer wert>
```

In Berechnungsformeln können die Grundrechenarten, Exponentialfunktion, logische Verknüpfungen, Vergleiche, interne Funktionen, Skriptfunktionen und externe Funktionen genutzt werden. Die Reihenfolge der Auflistung entspricht der Berechnungspriorität:

- Klammern „(“ … „)“
- Interne Funktionen: Dieses sind z. B. mathematische Funktionen wie `SQR(x)` → Quadratwurzel von x. Sie finden Eine Aufstellung der internen Funktionen weiter unten in dieser Dokumentation.
- Skriptfunktionen: Das Ergebnis einer Skriptfunktion wird in einer Skript-FUNCTION ermittelt (siehe unten)
- Externe Funktionen: Dieses sind Erweiterungen der Skript-Funktionalität, die im aufrufenden Programm realisiert werden.
- Exponentialfunktion („^“) → „x^2“ entspricht der mathematischen Schreibweise „x^2“
- Grundrechenarten (Punkt vor Strich):
  Addition („+“), Subtraktion („−“), Multiplikation („*“) und Division („/“)
- Verkettung von Zeichenketten (&)
- Vergleiche: (ein zutreffender Vergleich ergibt den Wert 1, ein nicht zutreffender den Wert 0) gleich („=“), ungleich („<>“), größer als („>“), kleiner als („<“), größer gleich („>=“) und kleiner gleich („<=“)

  Vergleiche werden als Zahlenvergleich durchgeführt, wenn beide Vergleichsoperanden einen numerischen Inhalt haben. Anderenfalls werden die Zeichenketten verglichen.
- Logische Verknüpfungen (UND vor ODER): UND („AND“) und ODER („OR“)

  Die Operanden der logischen Verknüpfungen werden folgendermaßen bewertet: Ist der Operand numerisch, dann wird der Wert „0“ als FALSCH interpretiert – alle anderen numerischen Werte sind WAHR. Nicht numerische Operanden sind FALSCH, wenn sie leer

sind. Nicht leere Zeichenketten werden als WAHR gewertet. Als Ergebnis der Operation wird „0“ (=FALSCH) und „1“ (=WAHR) berechnet.

Beispiel für eine Wertzuweisung („normale“ Variable):

```
testvar = 100 + 3 * var2
```

In der Variablen „testvar“ steht z. B. nach dem Befehl der Wert 136, wenn „var2“ den Wert „12“ hat

Beispiel für eine Wertzuweisung (Collection-Variable):

```
@arr(1,1) = 100          → in „@arr(1,1)“ steht der Wert „100“
testvar = @arr("a","b") → in „testvar“ steht nachher der Wert aus „@arr("a,b")“
```

Kommentare, Druckausgaben, Rückgabewerte

Programmkommentare sind immer ganzzeilig und beginnen mit einem Apostroph:

```
' Dieses ist ein Kommentar
```

Druckausgaben werden durch das Schlüsselwort `PRINT` eingeleitet:

```
PRINT "1+2="; (1 + 2), a
```

In einer Print-Zeile können Zeichenketten, Variablen und Ausdrücker angegeben werden. Ist zwischen zwei Elementen ein Semikolon angegeben, so werden die Ausgaben ohne Leerstelle hintereinander geschrieben. Ein Komma als Trennzeichen sorgt vor dem Anfügen des Textes dafür, dass durch Einfügen von Leerstellen bis zur nächsten Tabulatorposition (alle 8 Stellen) vorgeschoben wird.

Die Ausgabe wird im Output-Bereich dargestellt.

```
PRINTRESULT "1+2="; (1 + 2), a
```

Die hier angegebenen Daten werden wie beim `PRINT`-Befehl aufbereitet und in die Liste der Ergebnis-Daten (`lsResultData`) eingefügt.

Programmausführung

Es sind verschiedene Verfahren zur Steuerung der Programmausführung implementiert:

```
IF <bedingung-1> THEN
    …
[ELSEIF <bedingung-n> THEN
    …]
[ELSE
    …]
END IF
```

Über die IF-Verzweigung wird ein Programmblock nur ausgeführt, wenn die zugehörige Bedingung erfüllt ist. Bedingungen sind zutreffend, wenn die Berechnung einen Wert ungleich 0 ergibt.

Über den optionalen ELSEIF-Befehl kann eine Mehrfach-Verzweigung realisiert werden – ELSEIF kann mehrfach mit unterschiedlichen Bedingungen verwendet werden.

Die Befehle nach ELSE werden immer dann ausgeführt, wenn keine der vorherigen IF/ELSEIF-Bedingungen eingetroffen ist.

Mit dem END IF wird die bedingte Verzweigung abgeschlossen.

```
FOR <variable>=<startwert> TO <endwert> [STEP <schrittweite>]
    …
    [EXIT FOR]
    …
NEXT [<variable>]
```

Das FOR-Schleifen-Konstrukt erlaubt das mehrfache Durchlaufen eines Programmabschnittes. Dabei wird eine Laufvariable <variable> mit einem Startwert initialisiert. Nach dem Durchlaufen des Programmblocks wird die Laufvariable mit der bei STEP angegebenen Schrittweite addiert. Ist die Schrittweite nicht angegeben, so wird der Wert „1“ als Schrittweite angenommen. Ist der Endwert überschritten, wird keine weitere Wiederholung der Schleife mehr ausgeführt.

Der Befehl EXIT FOR verlässt die übergeordnete FOR-Schleife und setzt die Verarbeitung beim ersten Befehl hinter dem NEXT-Befehl fort.

Die Angabe der <variable> hinter NEXT ist optional. Ist eine Variable gesetzt, muss diese mit der des letzten FOR übereinstimmen – anderenfalls wird ein Fehler erzeugt.

```
DO [WHILE|UNTIL][<bedingung-1>]
    …
    [EXIT DO]
    …
LOOP [WHILE|UNTIL][<bedingung-2>]
```

Mit Hilfe der DO-LOOP-Schleife kann eine Schleife ohne Laufvariable implementiert werden. Dabei kann hinter dem DO und hinter dem LOOP eine Bedingung angegeben werden, um die Schleife weiterzuführen oder zu beenden. Sinnvoll ist aber nur eine Bedingung!

Bei einer WHILE-Bedingung wird die Schleife fortgesetzt, solange die zugehörige Bedingung erfüllt ist. Bei UNTIL würde sie bei erfüllter Bedingung abgebrochen.

Der Befehl EXIT DO verlässt die übergeordnete DO-Schleife und setzt die Verarbeitung beim ersten Befehl hinter dem LOOP-Befehl fort.

```
SELECT CASE <vergleichswert>
    CASE <wert1>
        …
    [CASE <wert2>
        …]
    [CASE ELSE
        …]
END SELECT
```

Über den SELECT-Befehl wird eine Mehrfachverzweigung realisiert, in der abhängig von einem Vergleichswert der Programmblock ausgeführt wird, in welchem der passende Wert beim CASE angegeben ist. Findet sich kein entsprechender CASE-Befehl mit passendem Wert, so werden die Befehle hinter CASE ELSE ausgeführt.

Im Gegensatz zu VBA ist es nicht möglich, hinter CASE mehrere Werte anzugeben.

Sprung an eine andere Programmposition

```
<label>:
```

Definition eines Sprungziels für GOTO / ON…GOTO

```
GOTO <label>
```

Die Programmausführung wird beim Sprungziel <label> fortgesetzt*.

```
ON <wert> GOTO <label-1>,<label-2>,…,<label-n>
```

Je nach <wert> wird die Programmausführung bei einem der angegebenen Sprungziele fortgesetzt. Die Ganzzahl <wert> bestimmt die Position der Zielposition – bei „<wert>=1“ wird <label-1> angesprungen, bei „<wert>=2“ das <label-2> usw.*

* Das Sprungziel von GOTO / ON…GOTO muss in der gleichen Funktion oder Prozedur liegen. Anderenfalls wird eine Fehlernachricht ausgegeben.

Warten und Popup-Fenster

```
SLEEP <wartezeit>
```

Das Programmausführung wird für `<wartezeit>` Millisekunden gestoppt.

```
MSGBOX <anzeigetext>
```

Es wird ein Popup-Fenster mit dem `<anzeigetext>` angezeigt. Dieses kann durch Anklicken des OK-Buttons verlassen werden. Die Programmausführung wird so lange angehalten.

Fehlerbehandlung

Die Fehlerbehandlung beschreibt, welche Aktionen beim Auftreten eines Programmfehlers durchgeführt werden. Ist die programmierte Fehlerbehandlung ausgeschaltet, dann wird die Programmausführung sofort beendet und die Fehlernachricht wird im Ausgabe-Fenster angezeigt.

Die folgenden Befehle erlauben es, dieses Verhalten zu modifizieren:

```
RAISE <fehlertext>
```

Es wird ein Fehler ausgelöst, der durch `<fehlertext>` genauer beschrieben wird.

```
ON ERROR GOTO <label>
```

Die Fehlerbehandlung wird eingeschaltet. Beim Auftreten eines Fehlers wird der Fehlertext in ERR und LASTERR geschrieben und die Programmausführung wird beim Sprungziel `<label>` fortgesetzt.

```
ON ERROR RESUME NEXT
```

Die Fehlerbehandlung wird eingeschaltet. Es wird der Fehlertext in `ERR` und `LASTERR` gespeichert und der Programmablauf wird mit dem folgenden Befehl fortgesetzt.

```
ON ERROR GOTO 0
```

Die Fehlerbehandlung wird ausgeschaltet.

```
CLEAR ERR
```

Eine vorher gespeicherte Fehlernachricht wird gelöscht

Eine Fehlerbehandlung wirkt auch über den Aufruf von Funktionen oder Prozeduren hinweg. So wird bei dem Sourcetext

```
ON ERROR RESUME NEXT
CALL proc1
A = 1
```

die Verarbeitung des Skriptes bei „A = 1“ fortgesetzt, wenn in der Prozedur „proc1“ ein Fehler (ohne eigene Fehlerbehandlung) auftritt.

Interne Funktionen und Konstanten

Funktionen werden im Programmtext eingegeben als

```
<funktionsname>(<parameter-1>,<parameter-2>,…)
```

Für die ABS-Funktion ist das dann `ABS(n)`, wobei „n“ ein numerischer Ausdruck ist. In den folgenden Beschreibungen steht „n“ für numerische Ausdrücke, „s“ für Zeichenkettenausdrücke. Diese Platzhalter können zur Unterscheidung noch mit Zahlen versehen sein,

Mathematische Funktionen / Konstanten

Funktion	Parameter	Beschreibung
ABS	n	Absolutwert
INT	n	Integerwert (die nächstkleinere Ganzzahl)
FIX	n	Integerwert (Nachkommastellen abgeschnitten)
SQR	n	Quadratwurzel
LOG	n	Natürlicher Logarithmus
LOG10	n	Logarithmus zur Basis 10
EXP	n	Gibt einen Wert zurück, der e^n angibt. „e“ ist die Basis des natürlichen Logarithmus.
SIN	n	Sinus
COS	n	Cosinus
TAN	n	Tangens
ATN	n	Arcustangens
SGN	n	Signum (-1 bei n<0, 0 bei n=0, 1 bei n>0)
ROUND	n1,n2	Runden von n1 auf n2 Nachkommastellen
PI		Konstante „π“:3.14159265926535897932384626433832795

Zeichenkettenfunktionen

Funktion	Parameter	Beschreibung
VAL	s	Numerischer Wert von s berechnen. Wenn s mit einer Zahl beginnt, dann wird die Zahl von diesem Teil ermittelt und der Rest des Strings ignoriert.
LEN	s	Länge von s in Zeichen
INSTR	s1,s2	Erste Position der Zeichenkette s2 in s1
INSTRREV	s1,s2	Letzte Position der Zeichenkette s2 in s1
ASC	s	ASCII-Code des ersten Zeichen von s
CHR	n	Zeichen mit dem ASCII-Code n
SPACE	n	Zeichenkette mit n Leestellen
STRING	n,s	Zeichenkette in der n mal der String s verkettet wurde
TRIM	s	Abschneiden der umgebenden Leerzeichen
LTRIM	s	Abschneiden der linken Leerzeichen

RTRIM	s	Abschneiden der rechten Leerzeichen
UCASE	s	Umwandlung in Großbuchstaben
LCASE	s	Umwandlung in Kleinbuchstaben
LEFT	s,n	Die linken n Zeichen der Zeichenkette s
RIGHT	s,n	Die rechten n Zeichen der Zeichenkette s
MID	s,n1[,n2]	Teilzeichenkette aus s ab Zeichenposition n1 in der Länge n2. Wird n2 weggelassen, so werden alle Zeichen ab Stelle n1 zurückgegeben
ISNUMERIC	s	1, wenn s numerisch ist, sonst 0
REPLACE	s1,s2,s3	Ersetzt in der Zeichenkette s1 alle vorkommenden s2 durch s3
INPUTBOX	s1,s2,s3	Es wird ein Popup-Fenster angezeigt, in dem ein Wert eingegeben werden kann. s1 gibt den Text an, der über dem Eingabefeld des Fensters dargestellt wird, s2 ist der Fenstertitel. Mit s3 wird der Initialwert des Eingabefeldes festgelegt.
MSGBOX	s1,s2,s3	Es wird ein Popup-Fenster angezeigt, welches eine Auswahl über Buttons erlaubt. s1 beschreibt den Text im Fenster, s2 den Fenstertitel und s3 die Optionen der Anzeige. s3 ist max. 3-stellig und jede Stelle hat eine besondere Funktion: Stelle 1 (Icon): „I“: Informations-Symbol „Q“ oder „?“: Fragezeichen-Symbol „W“ oder „!“: Rufzeichen-Symbol (Warnung) „E“ oder „X“: Fehler-Symbol sonst: kein Symbol Stelle 2 (Buttons): „O“: OK & Abbruch „R“: Wiederholen & Abbruch „y“: Ja & Nein “Y”: Ja & Nein & Abbruch “A”: Abbrechen & Wiederholen & Ignorieren sonst: OK Stelle 3 (Default-Button): „1“ oder „2“ oder „3“: Je nach der Buttonauswahl in Stelle 2 wird hier der Default-Button 1, 2 oder 3 gesetzt. Ein anderer Wert setzt keinen Default-Button.
SPLIT	s1,s2,s3	Teilt die Zeichenkette s1 an den Stellen mit s2 auf und speichert die Einzelteile in der Collection- Variable, die durch s3 festgelegt wurde. Die Funktion gibt die Anzahl der Teil-Zeichenketten zurück. s3 ist ein Zeichenketten-Ausdruck, der den Namen der Collection-Variablen enthält (z.B. „@arr“). Die Teilstrings werden dann im eindimensionalen Array „@arr(1)“, „@arr(2), …“ abgelegt.

Sonstige interne Funktionen / Konstanten

Funktion	Parameter	Beschreibung
RND	n	Zufallszahl vom 0 … n (n wird nie zurückgegeben)
TRUE		Entspricht „1“
FALSE		Entspricht „0“
DATETIME		Datum/Uhrzeit in der Form „YYYY-MM-DD HH:MM:SS“
DATE		Datum in der Form „YYYY-MM-DD“
TIME		Uhrzeit in der Form „HH:MM:SS“
TIMER		Anzahl Sekunden seit Mitternacht des aktuellen Tages
APP_PATH		Programmverzeichnis (endet immer mit Backslash „\“)
OPEN	s1,s2,n	Öffnet die Datei mit dem voll qualifizierten Dateinamen s1. Der Dateimodus ist s2 („I“=Input/Lesen, „O“=Output/Schreiben, „A“=Append/Anfügen). n legt die Dateinummer fest. Diese kann im Bereich von 1 bis 10 liegen. Es wird 0 zurückgegeben, wenn bei der Aktion ein Fehler aufgetreten ist – anderenfalls wird 1 zurückgegeben.
CLOSE	n	Die Datei mit der Dateinummer n (siehe OPEN) wird geschlossen. Der Rückgabewert „1“ zeigt den Erfolg der Aktion an – bei Fehlern wird „0“ zurückgegeben.
EOF	n	Der Funktionswert ist „0“, wenn das Dateiende der Datei mit der Dateinummer n noch nicht erreicht wurde, anderenfalls wird „1“ zurückgegeben.
READ	n	Es wird ein Datensatz aus der Datei mit der Dateinummer n gelesen und als Funktionsergebnis zurückgegeben.
WRITE	n,s	Es wird ein Datensatz in die Datei mit der Dateinummer n geschrieben. Der Dateninhalt ist s.
READFILE	s1,s2	Es wird eine Datei mit dem voll qualifizierten Dateinamen s1 eingelesen und in der Collection-Variable in s2 gespeichert. Rückgabewert ist die Anzahl der gelesenen Datensätze. Die Daten in der Collection-Variablen werden mit numerischem Satzindex beginnend bei 1 abgelegt.
WRITEFILE	s1,s2,n	Schreibt den Inhalt der durch s2 bestimmten Collection-Variablen in die vollständig qualifizierte Datei s1. Es werden nur die Datensätze gespeichert, die unter einem einteiligen und numerischen Schlüsselwert von 1 bis n in der Variablen abgelegt wurden. Es wird die Anzahl der geschriebenen Sätze zurückgegeben.
ERR		Fehlernachricht der letzten Programmzeile. Wenn keine Fehlerbehand-lung aktiviert wurde, wird das Programm beim Auftreten eines Fehlers beendet.
LASTERR		Text des letzten aufgetretenen Fehlers (nur mit aktivierter Fehlerbe-handlung)