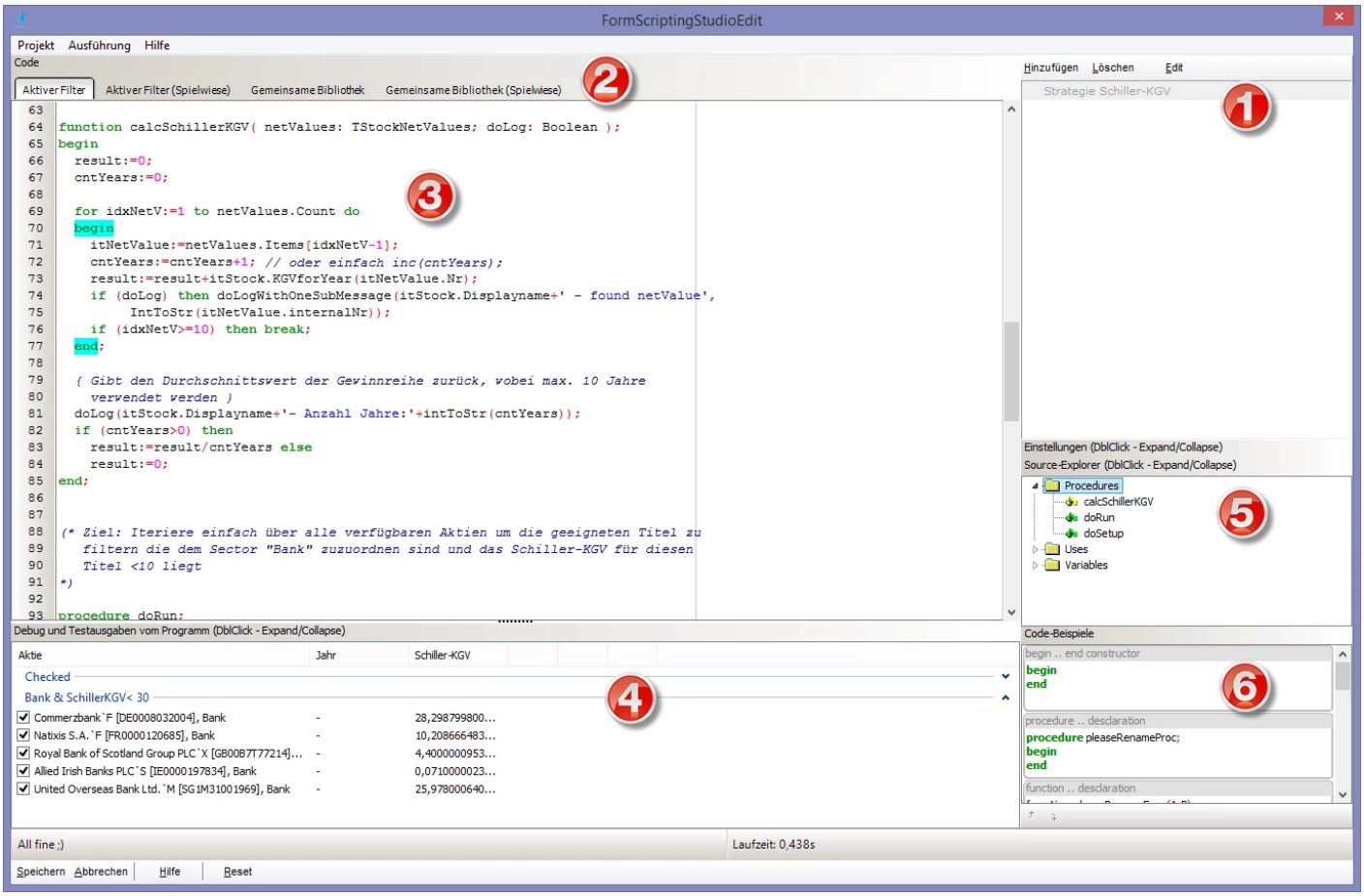
Handelsstrategie-Scripting-Studio

Grundlagen Handelsstrategie-Scripting

Die Scripting-Engine erweitert die bisherigen Filter-Module, um eine freie programmierbare Script-Engine. Es soll hiermit ein sehr hohen Freiheitsgrad in der Umsetzung eigener Filter gegeben werden und Routineaufgaben in der Titel-und Kurspflege automatisiert werden können.



Use-Cases

- Use-Case: Automatischer Export von Transaktionsdaten (Shareholder R/2 Börsensoftware)

- Use-Case: Automatische Watchliste auf Basis Indikator-Script (Shareholder R/2 Börsensoftware
- Use-Case: Automatische Stammdaten-Korrektur-Script (Shareholder R/2 Börsensoftware)

Handelsstrategie-Studio

Highlights

- Zugriff auf (alle) internen Datenobjekte
- Syntax-Highlighting
- Support einer produktiven und einer Entwicklungs-Version
- Support für Script-Bibliotheken, womit jedes Script von einem anderen aufgerufen werden kann. Dieser Mechanismus wird auch genutzt, um eine gemeinsame Bibliothek zu pflegen
- Hohe Ausführungsgeschwindigkeit (hier 0.4s über alle Aktien inkl. Log-Ausgaben)
- Source-Explorer

Überblick

Die Scripting-Engine führte freie geschriebene Scripte aus. Die Skripte sind in einem Pascal-Syntax, wobei spätere Alternativ-Sprachen denkbar sind (VB, Java, JS, Python). Der Einsatz der Scripting-Engine soll durchgängig durch SHAREholder vollzogen werden, d.h. eine Verwendung sowohl in Filter-Systemen, Druck-Templates als auch indirekt in den Handelssystemen ist vorgesehen. Der Startpunkt (v13.5) liegt hier bei einem Filtersystemen. Dabei ist ein wesentliches Feature der Zugriff auf alle relevanten internen Datenobjekte von SHAREholder. Es stehen damit beispielsweise folgende Funktionen zur Verfügung, die nicht nur lesenden Charakter haben:

- Automatische Anlegen einer Watchliste auf der Basis von Stammdaten-Filtern
- Automatische Bereinigung von Kommentaren in den vorhandenen Transaktionen, um hier nachträglich Ergänzungen vorzunehmen
- Automatische Bereinigung von Kursdaten
- Automatisches manuelles Erzeugen von Dateien (z.B. CSV-Export-Dateien) mit eigenen komplexeren Ausdrücken
- Anzeige von interessanten Titeln nach eigenen Regeln in Filterlisten

Folgender Syntax wird grundsätzlich unterstützt, womit komplette Programmstrukturen unterstützt werden:

begin .. end

procedure & function

if .. then .. else

for .. to .. do .. step

while .. do

repeat .. until

try .. except & try .. finally blocks

case statements

array constructors (x:=[1, 2, 3];)

^ , * , / , and , + , - , or , <> , >=, <= , = , > , < , div , mod , xor , shl , shroperators

access to object properties and methods (ObjectName.SubObject.Property)

Damit sind in Summe komplexere Scripte in SHAREholder nutzbar für verschiedene Strategie-Filter-Umsetzungen aber auch für eigene Massenoperationen bis hin zu Im-Exportszenarien. Die in SHAREholder genutzt objektorientierte Programmierung kann dabei in Skripten werden, um auf Daten und Methoden hierarchisch zuzugreifen. Auch das Schreiben eigener Klassen ist möglich (siehe eigenen Abschnitt).

Editor

Der Editor setzt folgende Anforderungen um (Status v2.5):

- freie Codeeingabe
- Autovervollständigung
- Parameter-Darstellung d.h. innerhalb von Funktionen, Proceduren werden die erlaubten Parameter gezeigt
- Syntax-Highlighting auf die eingestellte Default-Sprache (hier Pascal/Delphi) und
- die Wiederverwendung von Code-Fragmenten durch Code-Bibliotheken
- Code-Folding d.h. das Ein/Ausklappen von Code-Fragmenten
- Snippet-Bibliothek, um sofort typische Fragmente nutzen zu können
- Automatischen Source-Explorer für aktuelle procedures und Variablen

Dokumentation SHAREholder interne Methoden und

Klassen

- Eine Dokumentation der aktuell zur Verfügung stehenden Methoden die in den Scripten genutzt und referenziert werden können, finden Sie unter:

  https://www.shareholder24.com/docs/classes.html

Syntax der Scripte

Scripte enthalten a) procedure und function declarations und b) Haupt-Blöcke.

SCRIPT 1:

```
procedure DoSomething;
begin
  CallSomething;
end;

begin
  DoSomething;
end;
```

SCRIPT 2:

```
begin
  DoSomething;
end;
```

SCRIPT 3:

```
function MyFunction;
begin
  result:='Ok!';
end;
```

SCRIPT 4:

```
CallSomethingElse;
```

Statements sollten immer mit einem Semikolon ";" abgeschlossen werden.
begin..end Blöcke können jederzeit genutzt werden, um Statements zu gruppieren. Diese sind damit mit der Code-Folding-Funktion später auch ein-ausklappbar.

Identifier

Identifier Namen in Scripten (Variablen-Namen, function and procedure names, etc.) folgen folgenden einfachen Regeln:

- should begin with a character (a..z or A..Z), or '_', and can be followed by alphanumeric chars or '_' char.
- Cannot contain any other character os spaces.

Gültige Identifier:

```
VarName
_Some
V1A2
```

Einige ungültige Identifier:

```
2Var
My Name
Some-more
This,is,not,valid
```

Zuweisungen

Weisen Sie einen Wert oder ein Ausdrucksergebnis einer Variable oder Objekteigenschaft einfach mit ":=" zu.

```
MyVar := 2;
Button.Caption := 'This ' + 'is ok.';
```

Zeichenketten

Zeichenketten/Strings (Folge von Zeichen) werden in einfachen Anführungszeichen (') deklariert. Charakter. Doppelte Anführungszeichen (") werden nicht verwendet. Sie können auch " verwenden, als Zeichen innerhalb einer Zeichenkette. Es ist nicht notwendig, den '+'-Operator zu verwenden, um Zeichen-Codes zu einer Zeichenkette hinzuzufügen z.B. New-Lines mit Code: #13 und Code: #10. Einige Beispiele:

```
A := 'This is a text';
Str := 'Text '+'concat';
B := 'String with CR and LF char at the end'#13#10;
C := 'String with '#33#34' characters in the middle';
```

Kommentare

Kommentare können innerhalb des Skripts eingefügt werden.
Sie können

- //-Zeichen oder
- (* *) oder
- { } verwenden.

  Bei Blöcke mit // Zeichen wird der Kommentar am Ende der Zeile beendet.

```
//This is a comment before ShowMessage
ShowMessage('Ok');

(* This is another comment *)
ShowMessage('More ok!');

{ And this is a comment
with two lines }
ShowMessage('End of okays');
```

Variablen

Es ist nicht notwendig, Variablentypen in einem Skript zu deklarieren, obwohl Sie jede beliebige Art darin verwenden können. Sie können eine Variable nur mit der var-Direktive und ihrem Namen deklarieren:

```
procedure Msg;
var S;
begin
  S:='Hello world!';
  ShowMessage(S);
end;

var A;
begin
  A:=0;
  A:=A+1;
end;

var S: string;
begin
  S:='Hello World!';
  ShowMessage(S);
end;
```

Arrays

Auch wenn der Typ der Variablen nicht erforderlich ist und in den meisten Fällen ignoriert wird, gibt es einige spezielle Typen, die eine eigene Bedeutung haben. Sie können so Variablen als Array deklarieren und die Variable wird automatisch
als ein Variantenarray dieses Typs initialisiert:

```
var Arr: array[0..10] of string;
begin
  Arr[1] := 'first';
end;
```

Der Typ der Array-Elemente und der niedrige Index sind optional. Die nachfolgenden Beispiele sind alle gültig und resultieren in demselben Array-Typ:

```
var
  Arr1: array[0..10] of string;
  Arr2: array[10] of string;
  Arr3: array[0..10];
  Arr4: array[10];
```

Skript-Arrays beginnen immer bei 0, wenn nicht eine andere Start-Zahl festgelegt worden sind.

Es existiert eine Skriptunterstützung für Array-Konstruktoren und Unterstützung auch für Varianten-Arrays. Zum Konstruieren ein Array, verwenden Sie die Zeichen "[" und "]". Sie können eine Array-Verschachtelung mit mehreren Indizes konstruieren. Sie können dann über Indizes auf Arrays zugreifen. Wenn ein Array aus mehreren Indizes besteht, trennen Sie die Indizes mit ",". Wenn Variable ein Varianten-Array ist, unterstützt das Skript automatisch die Indizierung in diesem variabel. Eine Variable ist ein Variantenarray, wenn sie über ein Array zugewiesen wurde. Konstruktor, wenn es eine direkte Referenz auf eine Delphi-Variable ist, die eine Variante ist Array oder wenn es mit VarArrayCreate erstellt wurde

```
NewArray := [ 2,4,6,8 ];
Num:=NewArray[1]; //Num receives "4"
MultiArray := [ ['green','red','blue'] ,
['apple','orange','lemon'] ];
Str:=MultiArray[0,2]; //Str receives 'blue'
MultiArray[1,1]:='new orange';
```

Indexes

Strings, Arrays und Array-Eigenschaften können mit "[" und "]" Zeichen indiziert werden. Beispiel: Wenn Str eine Zeichenkettenvariable ist, gibt der Ausdruck Str[3] die dritte Zeichen in der durch Str bezeichneten Zeichenkette.

```
MyChar:=MyStr[2];
MyStr[1]:='A';
MyArray[1,2]:=1530;
Lines.Strings[2]:='Some text';
```

if-statements

Es gibt zwei Formen der if-Anweisung: if...then und die if...then...else. Wenn der if-Ausdruck wahr ist, wird die Anweisung (oder der Block) ausgeführt. Wenn es einen anderen Teil gibt und der Ausdruck falsch ist, wird die Anweisung (oder der Block) nach dem anderen ausführen.

```
if J <> 0 then Result := I/J;

if J = 0 then Exit else Result := I/J;

if J <> 0 then
begin
  Result := I/J;
  Count := Count + 1;
end else
  Done := True;
```

while-statements

Eine while-Anweisung wird verwendet, um eine Anweisung oder einen Block zu wiederholen, während eine Bedingung (Ausdruck) als wahr ausgewertet wird. Die Kontrollbedingung wird ausgewertet vor der Anweisungsfolge. Wenn also die Bedingung bei der ersten Iteration falsch ist, wird die Anweisungsfolge nie ausgeführt. Die while-Anweisung führt ihre konstituierende Aussage (oder Block) wiederholt aus, solange der Ausdruck in der Iteration gültig bleibt.

```
while Data[I] <> X do I := I + 1;
while I > 0 do
begin
  if Odd(I) then Z := Z * X;
  I := I div 2;
  X := Sqr(X);
end;

while not Eof(InputFile) do
begin
  Readln(InputFile, Line);
  Process(Line);
```

```
end;
```

repeat statements

Die Syntax einer Wiederholungserklärung ist Wiederholungserklärung1; ...; Erklärung; bis Ausdruck, wobei expression einen booleschen Wert zurückgibt. Die Wiederholungsanweisung führt seine Sequenz von konstituierenden Aussagen kontinuierlich aus und testet Ausdruck nach jeder Iteration. Wenn der Ausdruck True zurückgibt, wird die Wiederholung Anweisung beendet. Die Sequenz wird immer mindestens einmal ausgeführt, weil Ausdruck wird erst nach der ersten Iteration ausgewertet. Beispiele:

```
repeat
  K := I mod J;
  I := J;
  J := K;
until J = 0;

repeat
  Write('Enter a value (0..9): ');
  Readln(I);
until (I >= 0) and (I <= 9);
```

for statements

Scripter unterstützt Anweisungen mit folgender Syntax:

für Zähler :=initialValue bis finalValue
  do-Anweisung

Bei einer Anweisung, die den Zähler auf initialValue setzt, wird die Ausführung der Anweisung wiederholt (oder Block) und erhöhen den Wert des Zählers, bis der Zähler den Endwert erreicht hat.

```
for c:=1 to 10 do
  a:=a+c;
```

```
for i:=a to b do
begin
  j:=i^2;
  sum:=sum+j;
end;
```

case-statements

Der Scripter unterstützt Fallanweisungen mit folgender Syntax:

```
case selectorAusdruck von

  caseexpr1: Aussage1;
  ...
  caseexprn: Anweisung1; ..;

else

   elsestatement;

end
```

Wenn selectorAusruck mit dem Ergebnis eines der case-Ausdrücke übereinstimmt, wird die entsprechende Anweisung (oder der entsprechende Block) ausgeführt. Andernfalls wird das elsestatement ausgeführt. Ansonsten ist ein Teil der caseexprn-Anweisung optional. Anders als in Delphi muss die case-Anweisung im Skript nicht nur ordinale Werte verwenden. Sie können sowohl im Selektorausdruck als auch im Case-Ausdruck Ausdrücke beliebigen Typs verwenden. Beispiel:

```
case uppercase(Fruit) of
   'lime': ShowMessage('green');
   'orange': ShowMessage('orange');
   'apple': ShowMessage('red');
else
   ShowMessage('black');
end;
```

function and procedure declaration

Die Deklaration von Funktionen und Prozeduren ist ähnlich wie bei Object Pascal in Delphi, mit dem Unterschied, dass man keine Variablentypen angibt. Genau wie bei OP verwenden Sie implizit deklarierte Ergebnisvariablen, um Funktionswerte zurückzugeben. Es können auch Parameter per Referenz verwendet werden, mit der erwähnten Einschränkung: es ist nicht notwendig, Variablentypen anzugeben. Für eine einfacher Lesbarkeit kann dies jedoch genutzt werden ergänzend zu einer Namensgebung, die den Ergebnis-Typ mit in den Funktionsnamen aufnimmt z.B. TodayAsString ,TodayAsDate ...

```
procedure HelloWord;
begin
  ShowMessage('Hello world!');
end;

procedure UpcaseMessage(Msg);
begin
  ShowMessage(Uppercase(Msg));
end;

function TodayAsString;
begin
  result:=DateToStr(Date);
end;

function Max(A,B);
begin
  if A>B then
    result:=A
  else
    result:=B;
end;

procedure SwapValues(var A, B);
Var Temp;
begin
  Temp:=A;
  A:=B;
  B:=Temp;
end;
```

Script-based libraries

Skript-basierte Bibliothek ist das Konzept, bei dem ein Skript andere Skripte "benutzen" kann (zum Aufruf von Prozeduren, zum Setzen globaler Variablen usw.).

Nehmen Sie zum Beispiel die folgenden Skripte:

```
//Script 1
uses Script2;

begin
  Script2GlobalVar := 'Hello world!';
  ShowScript2Var;
end;




//Script2
var
  Script2GlobalVar: string;

procedure ShowScript2Var;
begin
  ShowMessage(Script2GlobalVar);
end;
```

Wenn Sie das erste Skript ausführen, "benutzt" es Script2, und dann ist es in der Lage, globale Variablen zu lesen/schreiben und Prozeduren aus Script2 aufzurufen.

Das einzige Problem hier ist, dass Skript 1 "wissen" muss, wo es Skript2 finden kann.

Wenn der Compiler z.B. einen Bezeichner in der uses-Klausel erreicht, muss er wissen, wo es zu finden ist:

```
uses Classes, Forms, Script2;
```

Hier wird nun folgender Ablauf abgebildet, um die Bibliothek auf verschiedene Weise zu "laden":

1. Er versucht, eine registrierte Delphi-basierte Bibliothek mit diesem Namen zu finden.

Mit anderen Worten, jede Bibliothek, die bei RegisterScripterLibrary registriert wurde. Dies gilt sowohl für die importierte VCL, die mit Scripter Studo bereitgestellt wird, als auch für die vom Import-Tool importierten Klassen. Dies ist der Fall für Klassen, Formulare und andere Einheiten.

2. Versucht, ein Skript in der Scripts-Sammlung zu finden, bei dem UnitName mit dem Bibliotheksnamen übereinstimmt.

Jedes TatScript-Objekt in der Scripter.Scripts-Sammlung verfügt über eine UnitName-Eigenschaft. Sie können diese Eigenschaft manuell einstellen, so dass das Skript-Objekt in diesen Situationen wie eine Bibliothek behandelt wird. Im obigen Beispiel könnten Sie ein Skript-Objekt hinzufügen, seine SourceCode-Eigenschaft auf den Skript-2-Code setzen und dann UnitName auf 'Skript2' setzen. Auf diese Weise könnte das Skript1 das Skript2 als Bibliothek finden und seine Variablen und Funktionen verwenden.

3. Versucht, eine Datei zu finden, deren Name mit dem Bibliotheksnamen übereinstimmt (wenn LibOptions.UseScriptFiles auf 'true' gesetzt ist)

Wenn LibOptions.UseScriptFiles auf true gesetzt ist, versucht der Skripter, die Bibliothek in Dateien zu finden. Wenn das Skript z. B. "uses Script2;" hat, sucht es nach Dateien mit dem Namen "Script2.psc".

Declaring classes in script (script-based classes)

Es ist jetzt möglich, Klassen in einem Skript zu deklarieren. Mit dieser Funktion können Sie eine Klasse deklarieren, um sie auf ähnliche Weise wie in Delphi zu verwenden: Sie erstellen eine Instanz der Klasse und verwenden sie wieder.

Jede Klasse muss in einem separaten Skript deklariert werden, d.h. Sie müssen für jede zu deklarierende Klasse ein Skript haben.

Sie machen das Skript zu einem "Klassenskript", indem Sie die $CLASS-Direktive am Anfang des Skripts, gefolgt vom Klassennamen, hinzufügen:

```
//Turn this script into a class script for TSomeClass
{$CLASS TSomeClass}
```

Methoden und Eigenschaften

Jede in einem Klassenskript deklarierte globale Variable wird tatsächlich zu einer Eigenschaft der Klasse. Jede Prozedur/Funktion in einem Skript wird zu einer Klassenmethode.

Die Hauptroutine des Skripts wird immer dann ausgeführt, wenn eine neue Instanz der Klasse erzeugt wird, so dass sie als Klasseninitialisierer verwendet werden kann und Sie einige Eigenschaften auf Standardwerte setzen und eine ordnungsgemäße Klasseninitialisierung durchführen können.

```
//My class script
{$CLASS TMyClass}
uses Dialogs;

var
  MyProperty: string;

procedure SomeMethod;
begin
  ShowMessage('Hello, world!');
end;

// class initializer
begin
  MyProperty := 'Default Value';
end;
```

Nutzung der Klassen

Sie können die Klasse aus anderen Skripten heraus verwenden, indem Sie einfach eine neue Instanz der genannten Klasse erzeugen:Sie können die Klasse aus anderen Skripten heraus verwenden, indem Sie einfach eine neue Instanz der genannten Klasse erzeugen:

```
uses MyClassScript;
var
  MyClass: TMyClass;
begin
  MyClass := TMyClass.Create;
```

```
  MyClass.MyProperty := 'test';
  MyClass.SomeMethod;
end;
```

Einzelheiten der Implementierung

Die im Skript deklarierten Klassen sind "Pseudo"-Klassen. Das bedeutet, dass keine neuen Delphi-Klassen erstellt werden. Obwohl Sie z.B. in dem obigen Beispiel TMyClass.Create aufrufen, der Name "TMyClass" nur für das Skriptsystem bedeutet, gibt es keine Delphi-Klasse mit dem Namen TMyClass. Alle Objekte, die als skriptbasierte Klassen erstellt werden, sind eigentlich Instanzen der Klasse TScriptBaseObject. Sie können dieses Verhalten ändern, um Instanzen einer anderen Klasse zu erzeugen, aber diese neue Klasse muss von der Klasse TScriptBaseObject erben. Sie definieren die Basisklasse für alle "Pseudo"-Klassenobjekte in der Scripter-Eigenschaft ScriptBaseObjectClass.

Speicherverwaltung

Obwohl Sie in Skripten die Methode Free aufrufen können, um Speicher freizugeben, der mit Instanzen von skriptbasierten Klassen verbunden ist, müssen Sie das nicht tun.

Alle in Skripten erstellten Objekte, die auf Skript-Klassen basieren, werden schließlich von der Skripter-Komponente zerstört.

Einschränkungen

Da Scripter keine neuen echten Delphi-Klassen erstellt, gibt es einige Einschränkungen, was Sie damit machen können. Die wichtigste ist, dass Vererbung nicht unterstützt wird. Da alle Klassen in Skripter tatsächlich dieselbe Delphi-Klasse sind, können Sie keine Klassen erstellen, die von einer anderen Delphi-Klasse erben, außer der in der Klasse TScriptBaseObject deklarierten.

Demo-Script

Hier ein sehr einfaches Script, was über alle Aktien iteriert und eine Ausgabe macht, wenn eine RIC[isin] Variable definiert wurde.

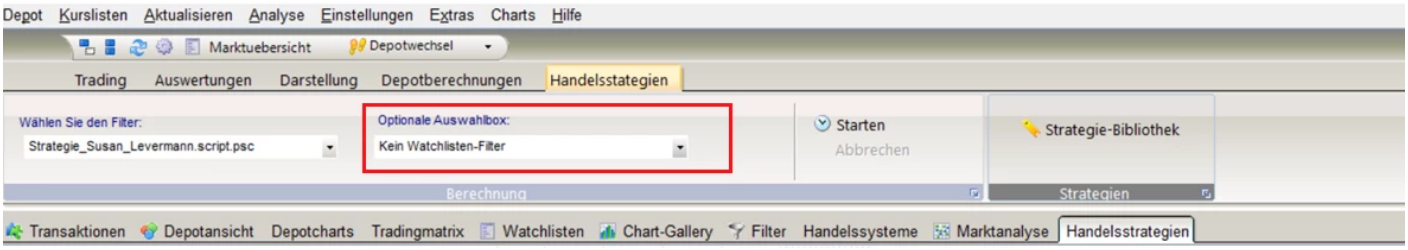
Dies ist nur eine Demo-Beispiel, zeigt aber relativ einfach die Funktionsweise. Idee des Scripts war es fehlerhaften Daten-Setups in Aktien zu finden und diese auch automatisch zu korrigieren.

```
uses Common;

(* CreateDate: 10.03.2019
   Author: Jens Werschmoeller
   Required: Shareholder-Version ab 19.3.x (neue Object-Type-Definition notwendig) *)

procedure doSetup;
var lColumn: TListColumn;
begin
  varPanelProgress.Progress.CompletionSmooth:=true;

  (* BeginUpdate/EndUpdate sollte aus Performancegründen zwingend genutzt werden! *)
  varLog.BeginUpdate;
  varLog.Items.Clear;
  (* Header für das Logging passend vorbereiten *)
  varLog.Columns.Clear;
  lColumn:=varLog.Columns.Add; lColumn.Caption:='Name'; lColumn.Width:=350;
  lColumn:=varLog.Columns.Add; lColumn.Caption:='Alter Wert'; lColumn.Width:=200;
  lColumn:=varLog.Columns.Add; lColumn.Caption:='Neuer Wert'; lColumn.Width:=200;

  varLog.GroupView:=false;
  varLog.EndUpdate;
end;

(* Ziel: Automatische Korrektur der RIC-Einträge und gleichzeitig einfachstes Demo-Programm *)
procedure doRun( pIsSimulation: Boolean );
var lStock: TStock;
    lStockVar: TStockVariable;
    lLogItem: TListItem;
```

```
begin
   (* Log-Einträge werden über die varLog-Variable geschrieben, die vom Typ TAdvListView darstellt. Diese greift
über .Items auf
      alle Log-Zeilen zu. In der Common-Bibliothek sind vereinfachte Methoden-Zugriffe möglich wie doLog *)

   varPanelProgress.Progress.Max:=varStocks.Count;
   varPanelProgress.Progress.Min:=1;
   varPanelProgress.Progress.ShowGradient:=true;

   (* Durchlauf alle Titel zur Prüfung *)
   for idxStock:=1 to varStocks.Count do if (varIsCanceled=false) then
   begin
     lStock:=varStocks.Items[idxStock-1];

     (* Zugriff auf den Fortschrittsbalken mit Aktualisierung auf die aktuelle Index-Position
*)
     varPanelProgress.Progress.Position:=idxStock;

     (* Aus Performancegründen wird nur bei jedem 500. Durchlauf überhaupt die Anzeige am Frontend
aktualisiert *)
     if (idxStock mod 500=0) then
       varApplication.ProcessMessages;

     (* Zugriff entspricht itStock.StockVariables.ItemsName['RIC[isin]']; Über die ID ist es jedoch "stabiler" *)
     lStockVar:=lStock.StockVariables.ItemsNr[12];
     if (lStockVar<>nil) then
     begin
       (* Logging der Ergebnisse über die Hilfsmethode in der Common-Lib / siehe Reiter Gemeinsame Bibliothek
*)
       itLogItem:=doLogAndCheck(lStock.Displayname+' ['+lStock.ISIN+'], '+lStock.Sector);
       itLogItem.SubItems.Add(lStockVar.Wert);

       (* Neue Wertzuweisungen der Variablen sind einfach möglich über itStockWert.Wert = ... *)
       itLogItem.SubItems.Add('-');
     end;
  end;
end;

begin
 doSetup;
```

```
  doRun( true );
end;
```

Optionale Handelsstrategie-Parameter



Handelsstrategien können einfacher benutzt werden, wenn statt statischer Code-Anpassung beim Durchlauf einer Strategie eine Vorauswahl durch den Nutzer erfolgen kann. Hierfür ist eine "Optionale Auswahlbox" (s.o.) verfügbar gemacht, die in der Strategie selbst individuell initialisiert werden kann mit Werte und auch individuell ausgewertet werden kann. Beispiele hierfür sind:

- Anzeige und Auswahl aus den verfügbaren Watchlisten auf denen die Strategie angewendet werden soll
- Anzeige und Auswahl aus den verfügbaren Marktsegmenten auf denen die Strategie angewendet werden soll
- Auswahl eines Detail-Levels für die Anzeige z.B. Log-Info, Log-Error, Log-Details

Die Auswahlbox ist im Default leer und muss auch nicht benutzt werden. Um diese in der eigenen Strategie zu nutzen, muss lediglich eine SubRoutine "doInit" eingerichtet werden. Nachfolgend ein Beispiel für die Auswahl einer optionalen Watchliste:

Initialisierungs-Routine "doInit" für die Auswahlbox

```
procedure doInit;
begin
  varWatchlists.InitStrings( varParamCombobox.Items, true);
  varParamCombobox.Items.Insert(0,'Kein Watchlisten-Filter');
  varParamCombobox.ItemIndex:=0;
  varParamCombobox.Enabled:=true;
end;
```

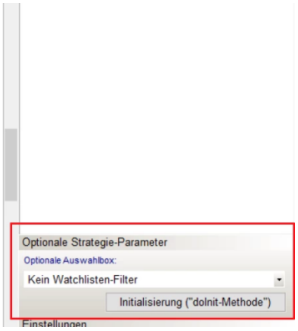
Nutzung der Nutzerauswahl in der Auswahlbox

```
procedure doRun;
var fCheckWatchlist: TWatchlist;
begin
    varPanelStatus.Text:='Los gehts ... ';
    varPanelProgress.Progress.Max:=varStocks.Count;
    varPanelProgress.Progress.Min:=1;
    itLogItem:=nil;

    if (varParamCombobox.ItemIndex>=0) and
(varParamCombobox.Items.Objects[varParamCombobox.ItemIndex]<>nil) then

fCheckWatchlist:=varWatchlists.getItemWithName(varParamCombobox.Items[varParamCombobox.ItemIndex])
else
      fCheckWatchlist:=nil;

    for idxStock:=1 to varStocks.Count do if (varIsCanceled=false) then
    begin
      itStock:=varStocks.Items[idxStock-1];
      itBenchmarkStock:=varStocks.ItemISIN[getBenchmarkISIN];
      iPoints:=0;
      varPanelProgress.Progress.ShowGradient:=true;
      varPanelProgress.Progress.Position:=idxStock;
      if (idxStock mod 500=0) then
        varApplication.ProcessMessages;

      if (itStock.StockNetValues.Count>3) and
         (calcMarktkapitalisierung>0.01) and
         //(Segments.getAndToInternalNrAsBool(itStock.StockSegments, itSegment.Nr)) and
         ((fCheckWatchlist=nil) or (fCheckWatchlist.List.FindISIN(itStock.ISIN)))
      then
      begin
```

Nutzung im Handelsstrategie-Studio

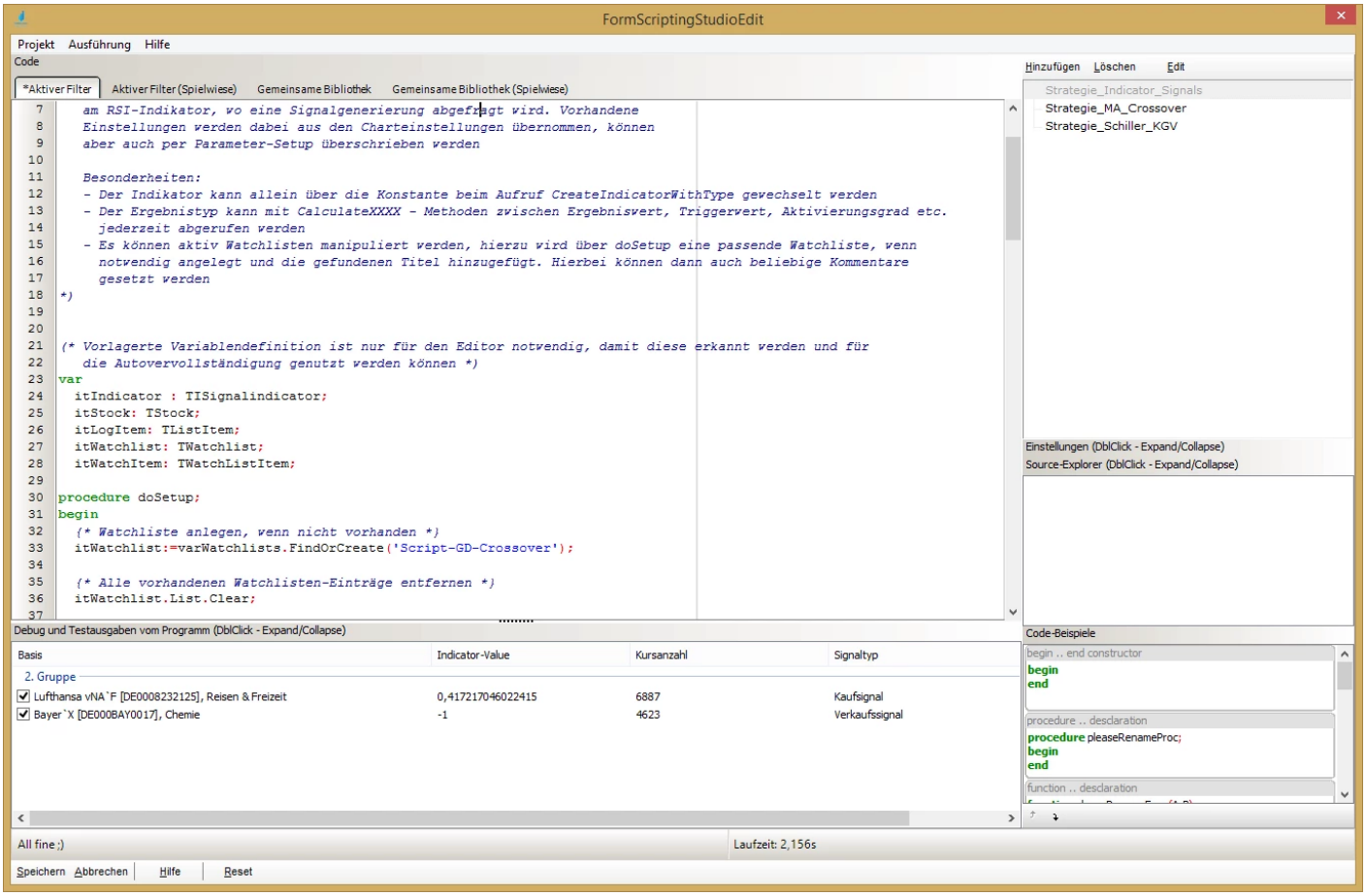
Im Studio steht die Auswahlbox für die Entwicklung ebenfalls zur Verfügung. Um die Routine zu starten, muss einmalig der Button "Initialisierung" aufgerufen werden. Die Hauptroutine (hier im doRun) sollte aber auch damit umgehen können, wenn die Auswahlbox leer ist d.h. varParamCombobox.Count=0 oder varParamCombobox.ItemIndex=-1 keine aktuelle Auswahl besitzt.

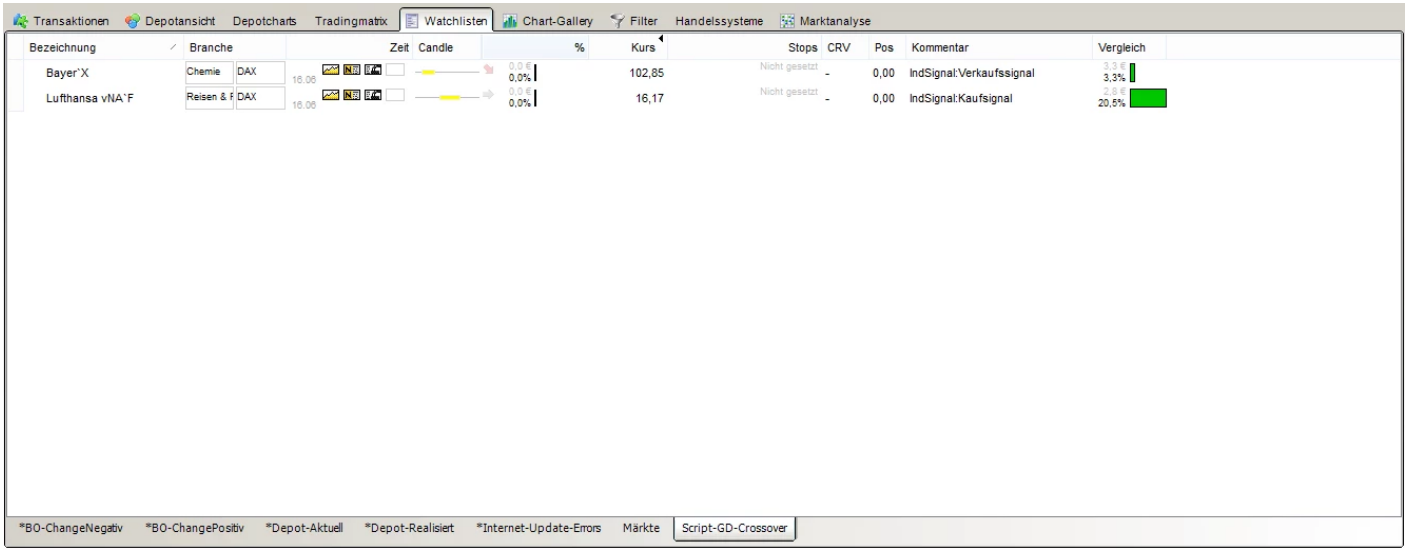
```
293 procedure doInit;
294 begin
295   varWatchlists.InitStrings( varParamCombobox.Items, true);
296   varParamCombobox.Items.Insert(0,'Kein Watchlisten-Filter');
297   varParamCombobox.ItemIndex:=0;
298   varParamCombobox.Enabled:=true;
299 end;
300
301 (* Ziel: Iteriere einfach über alle verfügbaren Aktien um die geeigneten Titel zu
302    filtern die dem Sector "Bank" zuzuordnen sind und das Schiller-KGV für diesen
303    Titel <10 liegt
304 *)
305
306 procedure doRun;
307 var fCheckWatchlist: TWatchlist;
308 begin
309     varPanelStatus.Text:='Los gehts ... ';
310     varPanelProgress.Progress.Max:=varStocks.Count;
311     varPanelProgress.Progress.Min:=1;
312     itLogItem:=nil;
313
```



Scripting für Indikatoren

Die folgenden Beispiele stehen auch im Scripting-Studio in der Profiversion zur Verfügung und ergeben nach Ausführung folgende Ergebnisse d,h. direkte Logausgabe im Scripting-Studio nach eigener Formatierung und im Watchlisten-Fenster die korrekte automatische Zuordnung und Erstellung der Titel.






```
 uses Common;
(* CreateDate: 31.07.2014
   Author: Jens Werschmoeller
   Required: 13.4.18
   Ziel ist die Demonstration der Anwendung von Indikatoren. Hier am Beispiel
   am RSI-Indikator, wo eine Signalgenerierung abgefragt wird. Vorhandene
   Einstellungen werden dabei aus den Charteinstellungen übernommen, können
   aber auch per Parameter-Setup überschrieben werden
   Besonderheiten:
   - Der Indikator kann allein über die Konstante beim Aufruf CreateIndicatorWithType gewechselt werden
   - Der Ergebnistyp kann mit CalculateXXXX - Methoden zwischen Ergebniswert, Triggerwert, Aktivierungsgrad
etc.
     jederzeit abgerufen werden
   - Es können aktiv Watchlisten manipuliert werden, hierzu wird über doSetup eine passende Watchliste, wenn
     notwendig angelegt und die gefundenen Titel hinzugefügt. Hierbei können dann auch beliebige Kommentare
     gesetzt werden
*)

(* Vorlagerte Variablendefinition ist nur für den Editor notwendig, damit diese erkannt werden und für
   die Autovervollständigung genutzt werden können *)
var
  itIndicator : TISignalindicator;
  itStock: TStock;
  itLogItem: TListItem;
  itWatchlist: TWatchlist;
  itWatchItem: TWatchListItem;
procedure doSetup;
begin
```

```
  {* Watchliste anlegen, wenn nicht vorhanden *}
  itWatchlist:=varWatchlists.FindOrCreate('Script-GD-Crossover');
  {* Alle vorhandenen Watchlisten-Einträge entfernen *}
  itWatchlist.List.Clear;
  (* Header für das Logging passend vorbereiten *)
  varLog.Column[1].Caption:='Indicator-Value';
  varLog.Column[1].Width:=200;
  varLog.Column[2].Caption:='Kursanzahl';
  varLog.Column[2].Width:=200;
  varLog.Column[3].Caption:='Signaltyp';
  varLog.Column[3].Width:=200;
end;
procedure doRun( indicatorTypeID: Integer);
var fSegment: TSegment;
begin
    varLog.BeginUpdate;
    varLog.Items.Clear;
    (* Durchlauf alle Titel zur Prüfung *)
    for idxStock:=1 to varStocks.Count     do
    begin
      itStock:=varStocks.Items[idxStock-1];
      (* Verwende nur Titel, die Deutsche Bank im Namen enthalten
         if (Pos('Deutsche Bank',itStock.Name)<>0) then
         if (Pos('DAX',varSegments.GetName( itStock.StockSegments ))<>0) then
         etc.
      *)
      if (itStock.Country='Deutschland') then
      begin
        (* Erzeuge den Indikator hier als den RSI-Indikator *)
        itIndicator:=varIndicators.CreateIndicatorWithTypeID(indicatorTypeID);
        (* Gebe dem Indikator die notwendigen Kursdaten des aktuellen Titels *)
        itIndicator.Kurse:=itStock.Kurs.createCache;
        (* Mittels Indicator.Parameter.Items[1].Wert:=10; könnten die Parameter
           des Indikators verändert werden. Die Bedeutung der Parameter muss pro
           Indikator nachgeschaut werden z.B. über Einstellungen / Indikatorengruppen / Doppelklick auf
           den Indikator *)
        (* Berechne die Ergebniswerte für das heutige Datum
           fValue:=itIndicator.CalculateValue(Trunc(Now()), itIndicator.Kurse);
        *)
        fValue:=itIndicator.CalculateErgActivation(Trunc(Now()), itIndicator.Kurse);
```

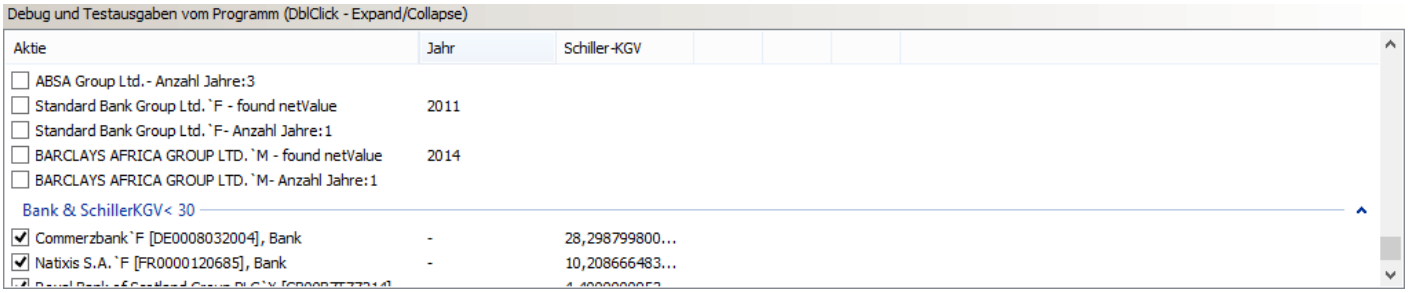
```
        (* Wenn ein Kauf/Verkaufssignal vorliegt, dann Logeintrag + Watch-Eintrag erzeugen *)
        if (fValue<>0) then
        begin
           (* Logging der Ergebnisse *)
           itLogItem:=doLogAndCheck(itStock.Displayname+' ['+itStock.ISIN+'], '+itStock.Sector);
           itLogItem.SubItems.Add(FloatToStr(fValue));
           if (itIndicator.Kurse<>nil) then
             itLogItem.SubItems.Add(InttoStr(itIndicator.Kurse.Count)) else
             itLogItem.SubItems.Add('-');
           if (fValue<0) then
             itLogItem.SubItems.Add('Verkaufssignal') else
             itLogItem.SubItems.Add('Kaufsignal');
           {* Watchlisteneintrag erzeugen *}
           itWatchItem:=TWatchListItem.Create(itStock);
           itWatchItem.CompareDate:=Now;
           if (fValue<0) then
             itWatchItem.Comment:='IndSignal:Verkaufssignal' else
             itWatchItem.Comment:='IndSignal:Kaufsignal';
           itWatchlist.List.Add(itWatchItem);
         end;
      end;
   end;
   varLog.EndUpdate;
end;
begin
  doSetup();
  doRun(cRSI);
end;
```

Logging in Scripten

Bereits in eines der ersten Beispiele integriert wurde das Logging in eine Standard-Debug-Ausgabe. Die hohe Flexibilität in der Darstellung des Loggings wird über eine Komponente von TMS-Software ermöglicht "TAdvListview".

https://www.tmssoftware.com/alvdoc.htm#Properties

Um die Funktionsweise möglichst einfach zu beschreiben hier konkret ein Beispiel aus eines der Standard-Beispielen:



Der dabei für verantwortliche Code ist sehr einfach für den Fall, dass ein Titel passend gefunden wurde. Die genutzten Variablen können (müssen aber nicht) zuvor definiert werden mit itLogItem: TAdvListItem; itStock und lSchillerKGV sind zuvor belegt worden im Programmcode und werden jetzt im Beispiel nur genutzt. itStock ist vom Typ TStock und besitzt daher dessen Eigenschaften. doLogAndCheck ist bereits in der Gemeinsamen Bibliothek angelegt und ist nachfolgend nur zur Vereinfachung dargestellt. Der Code kann natürlich aber auch jedesmal aufgenommen werden bzw. beliebig angepasst werden.

Siehe: TStock

```
function doLogAndCheck( logMessage ): TListItem;
begin
  result:=varLog.Items.Add;
  with result as TListItem do
  begin
    Caption:=logMessage;
    checked:=true;
    groupID:=1;
  end;
end;
```

```
itLogItem:=doLogAndCheck(itStock.Displayname+' ['+itStock.ISIN+'], '+itStock.Sector);
itLogItem.subItems.Add('-');
itLogItem.subItems.Add(FloatToStr(lSchillerKGV));
```

Hier der zugehörige Code-Teil für die Darstellung jedes geprüften Titels. Die GroupID ordnet dabei der ersten optischen Gruppe (siehe Screenshot / zwei ein-ausklappbare Bereiche) diesen Logeintrag zu.Im ersten Beispiel (s.o.) wird dies mit groupID:=1 entsprechend der zweiten Gruppe zugeordnet.

```
function doLog( logMessage): TListItem;
begin
  result:=varLog.Items.Add;
  with result as TListItem do
  begin
    Caption:=logMessage;
    groupID:=0;
  end;
end;


doLog(itStock.Displayname+'- Anzahl Jahre:'+intToStr(cntYears));
```

Verfügbare Variablen und Datenstrukturen

	Variable	Typ-Referenz	Hintergründe	Beispiel-Zugriffe
1	varApplication	TApplication	Zugriff auf das Applikationsobjekt der Anwendung wie z.B. • Application.ProcessMessages (Abarbeitung von UI-Refreshs)	
2	varParamCombobox	TAdvOfficeComboBox	Zugriff auf eine Auswahlbox, die für den Nutzer angezeigt wird, um für das Script Auswahloptionen für den Nutzer zur Verfügung zu stellen. In der Susan-Levermann-Strategie wird über diese eine Liste der verfügbaren Watchlisten angezeigt und nutzbar gemacht. Für Nutzer der Strategie steht der entsprechende Source-Code zur Verfügung. z.B. für • Watchlisten • Depots • Marktsegmente	Auswahlbox für den Nutzer füllen mit allen Watchlisten-Einträgen, danach mit allen MarktSegment-Einträgen und final als ersten Eintrag (Index=0) "Kein-Watchlisten-Filter" einfügen. varParamCombobox for llx:=1 to varWat varParamCombob for llx:=1 to varSegr varParamCombob varParamCombobox varParamCombobox varParamCombobox

	Variable	Typ-Referenz	Hintergründe	Beispiel-Zugriffe
3	varParamCheckbox	TAdvOfficeCheckBox	Zugriff auf eine Checkbox, die für den Nutzer angezeigt wird, um für das Script Auswahloptionen für den Nutzer zur Verfügung zu stellen. In der Susan-Levermann-Strategie wird diese für "Detaillierte Konfiguration" als Option genutzt.	Aktiviere überhaupt die Auswahlbox für den Nutzer. Im Standard wird die Auswahlbox überhaupt nicht angezeigt. Hinweis: Dies sollte vorzugsweise in einer "doInit-Methode" passieren. varParamCheckbox varParamCheckbox varParamCheckbox

	Variable	Typ-Referenz	Hintergründe	Beispiel-Zugriffe

4	varLog	TAdvListView	Die Primärausgabe erfolgt über das varLog-Objekt vom Typ TAdvListview. Es ist eine Komponente zur hierarchischen, gruppierten Tabellenausgabe von Informationen. Die Objekte besitzt Columns und eine GroupView-Eigenschaft zur automatischen Gruppierung von Einträgen.
			Um ein neuen Log-Eintrag zu erzeugen, kann über varLog.Items.Add eine neue Zeile eingefügt werden. Das dabei zurückgegebene Objekt ist vom Typ TListItem, was eine Caption, eine GroupID, ImageIndex, Checked-Eigenschaft besitzt.
				Kopfzeile der Logausgabe gestalten:
				(* BeginUpdate/End
				varLog.BeginUpdate
				varLog.Items.Clear;
				
				(* Header für das Lo
				varLog.Columns.Cle
				lColumn:=varLog.Co
				lColumn:=varLog.Co
				lColumn:=varLog.Co
				
				varLog.GroupView:=
				varLog.EndUpdate;
				
				Konkretes Logging:
				function doLogWith(
				begin
				result:=varLog.Iter
				with result as TList
				begin
				Caption:=logMess
				subitems.add(log
				groupID:=0;
				end;
				end;
				
				Automatische Sortierung:
				varLog.SortColumn:
				varLog.SortDirectior
				varLog.SortType:=3
				varLog.Sort;

5	varPanelStatus	TAdvOfficeStatusPanel	Status-Updates für den Nutzer über die Statuszeile. Dabei können primär Textnachrichten über die varPanelStatus.Text Eigenschaft ausgegeben werden.-	

	Variable	Typ-Referenz	Hintergründe	Beispiel-Zugriffe
6	varPanelProgress	TAdvOfficeStatusPanel	Status-Updates für den Nutzer über die Statuszeile. Dabei liegt der Fokus auf der Fortschrittsanzeige.	varPanelProgress.Pr varPanelProgress.Pr varPanelProgress.Pr
7	varIsCanceled	Boolean	Der Nutzer kann ein Script in der Berechnung abbrechen. Dabei wird das Script nicht hart abgebrochen, sondern zunächst nur die Variable auf "true" gesetzt. Innerhalb des Scripts sollte diese daher immer abgefragt werden, um einen geordneten Abbruch zu realisieren.	Haupt-Iteration über alle Aktien, wobei aber bei jeder Iteration explizit die Abbruchsvariable "varIsCanceled" geprüft wird, um geordnet abzubrechen: (* Durchlauf alle T for idxStock:=1 t begin

8	varStocks	TStocks	Zugriff auf alle Assets/Stocks in ShareHolder. Der Zugriff kann dabei sowohl lesend, als auch schreibend erfolgen. So können z.B. automatisiert Namensanpassungen vorgenommen werden.	Iteration über alle Stocks:
				(* Durchlauf alle Tit
				for idxStock:=1 to
				begin
				itStock:=varStoc
				
				Zugriff auf Stock-Variablen, um diese automatisiert zu verändern:
				(* Zugriff entspricht
				
				lStockVar:=lStock.S
				if (lStockVar<>nil) t
				begin
				...

	Variable	Typ-Referenz	Hintergründe	Beispiel-Zugriffe
9	varIndicators	TIndicatorGroups	Mittels Indicator.Parameter.Items[1].Wert:=10; könnten die Parameter des Indikators verändert werden. Die Bedeutung der Parameter muss pro Indikator nachgeschaut werden z.B. über Einstellungen / Indikatorengruppen / Doppelklick auf den Indikator.	itIndicator:=varIn
10	varSegments	TSegments	Zugriff auf alle Marktsegmente z.B. DAX, MDAX etc.	
11	varTransactions	TTransactions		

	Variable	Typ-Referenz	Hintergründe	Beispiel-Zugriffe

12	varWatchlists	TWatchlists	Zugriff auf die internen Watchlisten.	Neue Watchliste erzeugen, wenn nicht zuvor angelegt mit dem Namen "Wa:ETF-Momentum-Sortlist"
				`if (fFirst) and (fUseS`
				`begin`
				`  fWatchlist:=varWa`
				`  fWatchlist.Clear();`
				`  fFirst:=false;`
				`end;`
				
				Eintrag hinzufügen
				`if (fWatchlist<>nil)`
				`  begin`
				`  if (not fUseSepar`
				`  fWatchItem:=fW`
				`  begin`
				`  fWatchItem:=TW`
				`  fWatchlist.List.A`
				`  end;`
				
				`  if (fWatchItem<>`
				`  begin`
				`  fWatchItem.Co`
				`  //itWatchItem.C`
				`  fWatchItem.Pos`
				`  end;`
				`end;`

13	varDepot	TCalculatedDepotItems	Zugriff auf alle berechneten Depot-Positionen mit Stückzahl, gemittelten Preis, letzter Transaktion etc.	
14	varStops	TStopRates	Zugriff auf alle definierten Stopps im Programm für Titel. Dabei enthält die Liste nur definierte Stopps d.h. Titel müssen keine Stopps haben. Stopps sind unabhängig von Depotpositionen.	

	Variable	Typ-Referenz	Hintergründe	Beispiel-Zugriffe

15	varProgramSettings	TProgramSettings	Zugriff auf alle internen Programmeinstellungen wie Schriftgrößen, Kalkulationsbasis etc.	Verwende die offizielle Programmeinstellung für die KGV-Berechnung d.h. aktuelles Basisjahr (0), kommendes Jahr etc.
				`function calcPointsK`
				`begin`
				`  Result:=itStock.KG`
				`end;`

16	varStockExchanges	TStockExchanges	Zugriff auf die definierten Börsen wie F, Nasdaq, Xetra	
17	varINetVars	TINetVars	Zugriff auf die definierten Aktualisierungs-Internetvariablen	
18	varINetAddrs	TINetAddrs	Zugriff auf die definierten Aktualisierungs-Internetadressen	
19	varAssets	TAssets	Zugriff auf die definierten Asset-Klassen	
20	varStrategy	TStrategy	Zugriff auf alle definierten Strategien	
21	varNNetze	TNets	Zugriff auf alle definierten Vorhersage-Modelle	
22	varAccounts	TAccounts	Zugriff auf alle Konten	
23	varImpFormate	TImpFormats	Zugriff auf alle Importformate	
24	varTradeMethods	TTradeMethods		
25	varAssessments	TAssessments		
26	varSparplaene	TDepotSavingPlans	Zugriff auf die Sparpläne	
27	varINetUpdateGroups	TINetUpdateGroups	Zugriff auf die Kursaktualisierungsgruppen	

Registrierung von Variablen und Kontext im Detail für das Scripting-Studio

Version 19.3.2

```
with Scripter do
 begin
   LibOptions.SearchPath.Add(IDEEngine.BasePath);
   LibOptions.SourceFileExt := '.script';
   LibOptions.CompiledFileExt := '.psc';

   Scripter.OptionExplicit := false;
   logList.Items.Clear;
   Scripter.AddObject('varApplication', Application);
   Scripter.AddObject('varParamCombobox', fAdvCombobox);
   Scripter.AddObject('varParamCheckbox', fAdvCheckbox);



   Scripter.DefineClassByRTTI(TApplication);
   Scripter.DefineClassByRTTI(TBasisobject); // ,roInclude,false,'TBasisObject',[mvPublic,mvPublished]
   Scripter.DefineClassByRTTI(TAdvListView);
   Scripter.DefineClassByRTTI(TListItems);
   Scripter.DefineClassByRTTI(TStrings);
   Scripter.DefineClassByRTTI(TListItem);
   Scripter.DefineClassByRTTI(TListGroup);
   Scripter.DefineClassByRTTI(TListGroups);
   Scripter.DefineClassByRTTI(TStringList);



   Scripter.AddObject('varLog', flogList);
   Scripter.DefineClassByRTTI(TListColumns);
   Scripter.DefineClassByRTTI(TListColumn);
   Scripter.AddObject('varPanelStatus', fStatusPanel);
```

```
Scripter.AddObject('varPanelProgress', fProgressPanel);
Scripter.AddVariable('varIsCanceled', fIsCanceled);

Scripter.DefineClassByRTTI(TStocks);
Scripter.DefineClassByRTTI(TAsset);
Scripter.DefineClassByRTTI(TAssets);
Scripter.DefineClassByRTTI(TAutoImports);
Scripter.DefineClassByRTTI(TCalculatedDepotAccount);
Scripter.DefineClassByRTTI(TCalculatedDepotAccounts);
Scripter.DefineClassByRTTI(TCalculatedDepotItem);
Scripter.DefineClassByRTTI(TCalculatedDepotItems);
Scripter.DefineClassByRTTI(TCandle);
Scripter.DefineClassByRTTI(TCandleFormation);
Scripter.DefineClassByRTTI(TChartindicator);
Scripter.DefineClassByRTTI(TDepotSavingPlan);
Scripter.DefineClassByRTTI(TDepotSavingPlans);
Scripter.DefineClassByRTTI(TDynamicFilter);
Scripter.DefineClassByRTTI(TDynamicFilterCondition);
Scripter.DefineClassByRTTI(TDynamicFilterConditions);
Scripter.DefineClassByRTTI(TDynamicFilterConditionTree);
Scripter.DefineClassByRTTI(TDynamicFilterConditionTrees);
Scripter.DefineClassByRTTI(TDynamicFilterResult);
Scripter.DefineClassByRTTI(TDynamicFilterResults);
Scripter.DefineClassByRTTI(TDynamicFilters);
Scripter.DefineClassByRTTI(TEnhancedList);
Scripter.DefineClassByRTTI(TFilterTaipan);
Scripter.DefineClassByRTTI(TFilterTaipanItem);
Scripter.DefineClassByRTTI(THelperAverage);
Scripter.DefineClassByRTTI(THTMLParser);
Scripter.DefineClassByRTTI(TIAroon);
Scripter.DefineClassByRTTI(TIBollinger);
Scripter.DefineClassByRTTI(TICandleFormationen);
Scripter.DefineClassByRTTI(TICandleFormationenCache);
Scripter.DefineClassByRTTI(TICCI);
Scripter.DefineClassByRTTI(TIChaikin);
Scripter.DefineClassByRTTI(TICoppock);
Scripter.DefineClassByRTTI(TIDMI);
Scripter.DefineClassByRTTI(TIDSStochastik);
Scripter.DefineClassByRTTI(TIForceIndex);
Scripter.DefineClassByRTTI(TIGDEMA);
```

```
Scripter.DefineClassByRTTI(TIGDUmsatz);
Scripter.DefineClassByRTTI(TIHistVol);
Scripter.DefineClassByRTTI(TIMACD);
Scripter.DefineClassByRTTI(TIMFI);
Scripter.DefineClassByRTTI(TIMomentum);
Scripter.DefineClassByRTTI(TImpFormat);
Scripter.DefineClassByRTTI(TImpFormats);
Scripter.DefineClassByRTTI(TIndicator);
Scripter.DefineClassByRTTI(TIndicatorGroup);
Scripter.DefineClassByRTTI(TIndicatorGroups);
Scripter.DefineClassByRTTI(TIndicatorParam);
Scripter.DefineClassByRTTI(TIndicatorParams);
Scripter.DefineClassByRTTI(TIndicatorSignals);
Scripter.DefineClassByRTTI(TINegativeVolumeIndex);
Scripter.DefineClassByRTTI(TInetAddr);
Scripter.DefineClassByRTTI(TINetAddrs);
Scripter.DefineClassByRTTI(TINetUpdateGroup);
Scripter.DefineClassByRTTI(TINetUpdateGroups);
Scripter.DefineClassByRTTI(TINetVar);
Scripter.DefineClassByRTTI(TINetVars);
Scripter.DefineClassByRTTI(TINewHigh);
Scripter.DefineClassByRTTI(TINewLow);
Scripter.DefineClassByRTTI(TINNKorrelation);
Scripter.DefineClassByRTTI(TINNPrognose);
Scripter.DefineClassByRTTI(TInternetProperties);
Scripter.DefineClassByRTTI(TIntSignalIndicatorCache);
Scripter.DefineClassByRTTI(TIOnBalanceVolume);
Scripter.DefineClassByRTTI(TIPFE);
Scripter.DefineClassByRTTI(TIPositiveVolumeIndex);
Scripter.DefineClassByRTTI(TIPSAR);
Scripter.DefineClassByRTTI(TIPvt);
Scripter.DefineClassByRTTI(TIRAVI);
Scripter.DefineClassByRTTI(TIRMI);
Scripter.DefineClassByRTTI(TIRSI);
Scripter.DefineClassByRTTI(TIRSL);
Scripter.DefineClassByRTTI(TIRWI);
Scripter.DefineClassByRTTI(TISignalIndicator);
Scripter.DefineClassByRTTI(TISignalIndicatorCache);
Scripter.DefineClassByRTTI(TISignalIndicators);
Scripter.DefineClassByRTTI(TIStdDev);
```

```
Scripter.DefineClassByRTTI(TIStochastik);
Scripter.DefineClassByRTTI(TITrix);
Scripter.DefineClassByRTTI(TITRWinkel);
Scripter.DefineClassByRTTI(TITSF);
Scripter.DefineClassByRTTI(TIVHF);
Scripter.DefineClassByRTTI(TIVolumeNotis);
Scripter.DefineClassByRTTI(TIVolumePriceTrend);
Scripter.DefineClassByRTTI(TIWilderVol);
Scripter.DefineClassByRTTI(TParamEnhancedList);
Scripter.DefineClassByRTTI(TProfitStop);
Scripter.DefineClassByRTTI(TProgramSettings);
Scripter.DefineClassByRTTI(TProperties);
Scripter.DefineClassByRTTI(TSegment);
Scripter.DefineClassByRTTI(TSegments);
Scripter.DefineClassByRTTI(TSignalItem);
Scripter.DefineClassByRTTI(TSplit);
Scripter.DefineClassByRTTI(TSplits);
Scripter.DefineClassByRTTI(TStock);
Scripter.DefineClassByRTTI(TStockExchange);
Scripter.DefineClassByRTTI(TStockExchanges);
Scripter.DefineClassByRTTI(TStockNetValue);
Scripter.DefineClassByRTTI(TStockNetValues);
Scripter.DefineClassByRTTI(TStockNews);
Scripter.DefineClassByRTTI(TStockNewsList);
Scripter.DefineClassByRTTI(TStockProfile);
Scripter.DefineClassByRTTI(TStockProfiles);
Scripter.DefineClassByRTTI(TStocks);
Scripter.DefineClassByRTTI(TStockVariable);
Scripter.DefineClassByRTTI(TStockVariables);
Scripter.DefineClassByRTTI(TStopRate);
Scripter.DefineClassByRTTI(TStopRates);
Scripter.DefineClassByRTTI(TStrategy);
Scripter.DefineClassByRTTI(TStringParser);
Scripter.DefineClassByRTTI(TTaipanCatalogItem);
Scripter.DefineClassByRTTI(TTaipanCatalogItems);
Scripter.DefineClassByRTTI(TTradeMethod);
Scripter.DefineClassByRTTI(TTradeMethods);
Scripter.DefineClassByRTTI(TTradingSystem);
Scripter.DefineClassByRTTI(TTradingSystemMetricHelper);
Scripter.DefineClassByRTTI(TTradingSystemMetrics);
```

```
Scripter.DefineClassByRTTI(TTradingSystemThread);
Scripter.DefineClassByRTTI(TTradingSystemTrades);
Scripter.DefineClassByRTTI(TTrailingStop);
Scripter.DefineClassByRTTI(TWatchlist);
Scripter.DefineClassByRTTI(TWatchlistItem);
Scripter.DefineClassByRTTI(TWatchlistItems);
Scripter.DefineClassByRTTI(TWatchlists);
Scripter.DefineClassByRTTI(THistoryItem);
Scripter.DefineClassByRTTI(TKurse);
Scripter.AddObject('varStocks', Stocks);

Scripter.AddConstant('ctTrendfolger', ctTrendfolger);
Scripter.AddConstant('typMA', typMA);
Scripter.AddConstant('cRMI', cRMI);
Scripter.AddConstant('cCandlesticks', cCandlesticks);
Scripter.AddConstant('cMA', cMA);
Scripter.AddConstant('cBollinger', cBollinger);
Scripter.AddConstant('cPSAR', cPSAR);
Scripter.AddConstant('cGDUmsatz', cGDUmsatz);
Scripter.AddConstant('cMomentum', cMomentum);
Scripter.AddConstant('cRSI', cRSI);
Scripter.AddConstant('cStochastik', cStochastik);
Scripter.AddConstant('cChaikin', cChaikin);
Scripter.AddConstant('cDSStochastik', cDSStochastik);
Scripter.AddConstant('cMFI', cMFI);
Scripter.AddConstant('cCoppock', cCoppock);
Scripter.AddConstant('cRSL', cRSL);
Scripter.AddConstant('cMACD', cMACD);
Scripter.AddConstant('cTRIX', cTRIX);
Scripter.AddConstant('cCCI', cCCI);
Scripter.AddConstant('cRMI', cRMI);
Scripter.AddConstant('cPFE', cPFE);
Scripter.AddConstant('cTSF', cTSF);
Scripter.AddConstant('cPVT', cPVT);
Scripter.AddConstant('cNewHigh', cNewHigh);
Scripter.AddConstant('cNewLow', cNewLow);
Scripter.AddConstant('cStdDev', cStdDev);
Scripter.AddConstant('cHistVol', cHistVol);
Scripter.AddConstant('cVHF', cVHF);
Scripter.AddConstant('cWilderVOl', cWilderVOl);
```

```
Scripter.AddConstant('cADX', cADX);
Scripter.AddConstant('cRAVI', cRAVI);
Scripter.AddConstant('cTRWinkel', cTRWinkel);
Scripter.AddConstant('cRWI', cRWI);
Scripter.AddConstant('cAroon', cAroon);
Scripter.AddConstant('cNNKorrelation', cNNKorrelation);
Scripter.AddConstant('cNNPrognose', cNNPrognose);
Scripter.AddConstant('cForceIndex', cForceIndex);
Scripter.AddConstant('cOnBalanceVolume', cOnBalanceVolume);
Scripter.AddConstant('cVolumePriceTrend', cVolumePriceTrend);
Scripter.AddConstant('cNegativeVolumeIndex', cNegativeVolumeIndex);
Scripter.AddConstant('cPositivVolumeIndex', cPositivVolumeIndex);
Scripter.AddConstant('cVolumeNotisV', cVolumeNotisV);

Scripter.DefineClassByRTTI(TIndicator);
Scripter.DefineClassByRTTI(TChartindicator);
Scripter.DefineClassByRTTI(TISignalIndicator);
Scripter.DefineClassByRTTI(TIRMI);
Scripter.DefineClassByRTTI(TIndicatorGroups);
Scripter.DefineClassByRTTI(TIndicatorParams);
Scripter.DefineClassByRTTI(TIndicatorParam);
Scripter.DefineClassByRTTI(TIndicatorSignals);
Scripter.AddObject('varIndicators', IndicatorGroups);

Scripter.DefineClassByRTTI(TSegments);
Scripter.DefineClassByRTTI(TSegment);
Scripter.AddObject('varSegments', Segments);

Scripter.DefineClassByRTTI(TTransaction);
Scripter.DefineClassByRTTI(TTransactions);
Scripter.DefineClassByRTTI(TAssessments);
Scripter.AddObject('varTransactions', Transactions);


Scripter.DefineClassByRTTI(TWatchlist);
Scripter.DefineClassByRTTI(TWatchlists);
Scripter.DefineClassByRTTI(TWatchlistItems);
Scripter.DefineClassByRTTI(TWatchlistItem);
Scripter.AddObject('varWatchlists', Watchlists);
```

```
  Scripter.DefineClassByRTTI(TCalculatedDepotItems);
  Scripter.DefineClassByRTTI(TCalculatedDepotItem);
  Scripter.AddObject('varDepot', CalculatedDepotItems);
  Scripter.DefineClassByRTTI(TStoprate);
  Scripter.AddObject('varStops', StopRates);
  Scripter.AddObject('varProgramSettings', ProgramSettings);

  Scripter.AddObject('varStockExchanges', StockExchanges);
  Scripter.AddObject('varINetVars', INetVars);
  Scripter.AddObject('varINetAddrs', INetAddrs);
  Scripter.AddObject('varAssets', Assets);
  Scripter.AddObject('varStrategy', Strategy);
  Scripter.AddObject('varNNetze', NNetze);
  Scripter.AddObject('varAccounts', Accounts);
  Scripter.AddObject('varImpFormate', ImpFormate);
  Scripter.AddObject('varTradeMethods', TradeMethods);
  Scripter.AddObject('varAssessments', Assessments);
  Scripter.AddObject('varSparplaene', Sparplaene);
  Scripter.AddObject('varINetUpdateGroups', INetUpdateGroups);
end;
```

Alle TAdv* Objekte sind genauer durch den Komponentenhersteller beschrieben und können aus Lizenzgründen auch von mir nicht genauer aufgegriffen werden.

Siehe hierzu unter: https://www.tmssoftware.com/site/tmspack.asp

Automatische Stammdaten-Korrektur-Script als Beispiel

Use-Case- Alle Titel entweder exklusiv oder zusätzlich dem Archiv zuordnen.

```
uses Common;

procedure doSetup
begin
  ...
end;

procedure doRun;
var litStock     : TStock;
var lArchiveSegment: TSegment;
var lArchiveSegmentBitMask: TBitmask;

begin
  (* Alle Werte in Archiv. *)
  lArchiveSegment:=varSegments.ItemsName['Archiv'];
  lArchiveSegmentBitMask:=lArchiveSegment.getNrAsBitMask;
  for idxStock:=1 to varStocks.Count do
  begin
    litStock:=varStocks.Items[idxStock-1];
    if (varSegments.getAndToInternalNrAsBool( litStock.StockSegments, lArchiveSegmentBitMask)) then
    begin
      (* Alle Werte zusätzlich dem Archiv zuordnen *)
      litStock.StockSegments:=varSegments.getORToInternalNr( litStock.StockSegments,
lArchiveSegmentBitMask);
      (* oder exklusiv d.h. alle Werte nur ins Archiv verschieben
         itStock.StockSegments:=lArchiveSegmentBitMask;
      *)
    end;
  end;
end;
```

```
begin
  doSetup();
  doRun();
end;
```

Ab der 21.6.x Version werden die Markt und Watchlisten-Zuordnungen über Bitmasken zugeordnet.

Diese sind vom Typ TBitmask:

```
type
  TBitmask = record
    BitFields: Array[1..8] of UInt64;

    const MaxFields = 8;
    const MaxBit = MaxFields*64-1;

    class operator Add(const v1, v2: TBitmask): TBitmask;
    class operator Equal(const v1, v2: TBitmask): Boolean;
    class operator NotEqual(const v1, v2: TBitmask): Boolean;

  end align 16;
```

Das komplette Zurücksetzen eines Titels kann damit mit folgender Konstante erfolgen:
const cZeroBitmask : TBitmask = (Bitfields: (0,0,0,0,0,0,0,0));

Beispielzuordnungen als Konstante sind z.B.

```
      cEuroStoxx50: TBitmask = (Bitfields: (16,0,0,0,0,0,0,0));
      cNasdaq         : TBitmask = (Bitfields: (1 shl 12,0,0,0,0,0,0,0));
```

Folgende Methoden für alle von TEnhancedList abgegleiteten Klassen z.B. TSegments, TWatchlists, TStocks stehen zur Verfügung:

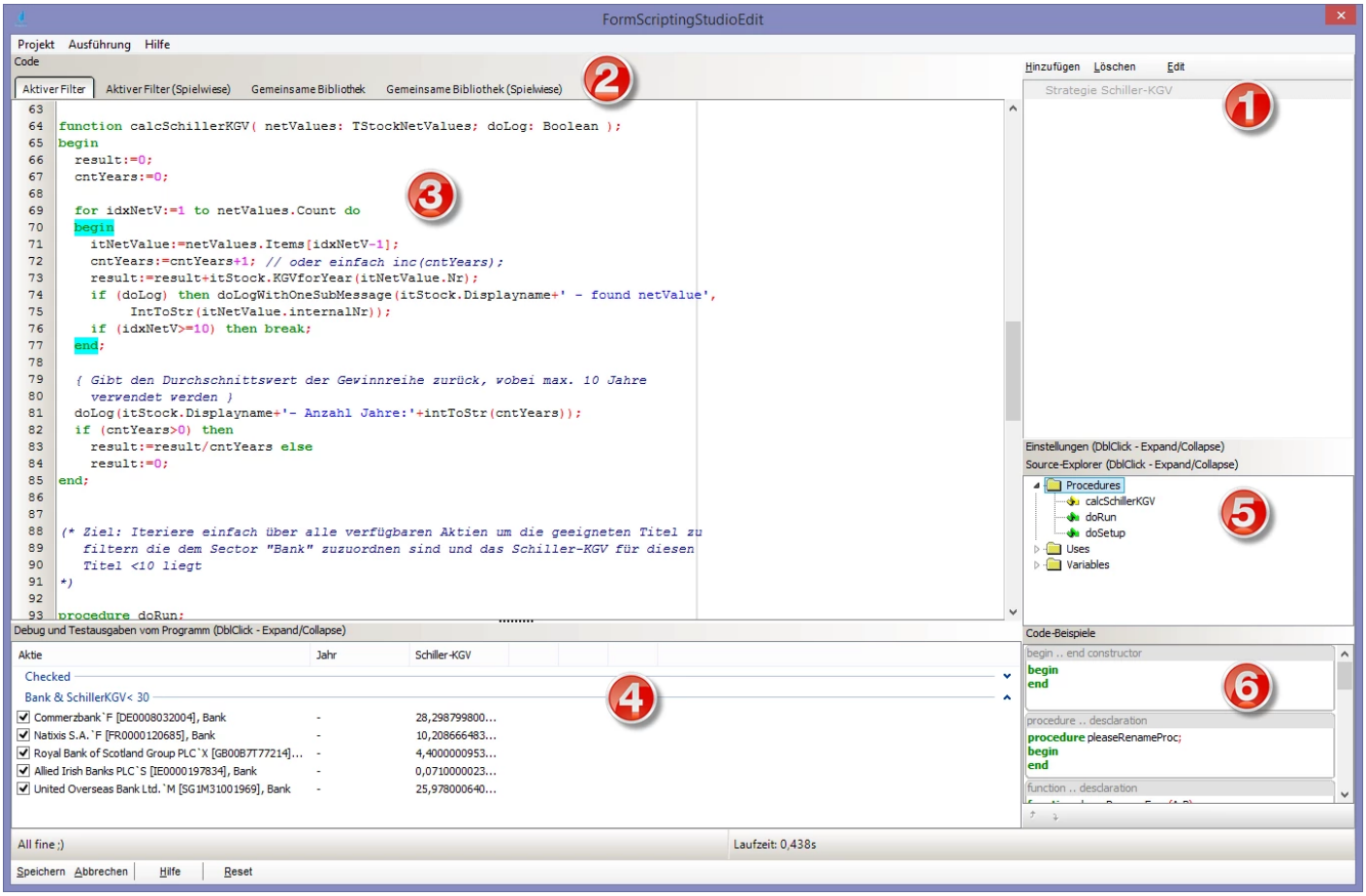
```
    class function getORToInternalNr(orValue, Nr: TBitmask): TBitmask;
    class function getORToNr(orValue, Nr: Int64): Int64;



    class function getXORToInternalNr(xorValue, Nr: TBitmask): TBitmask;
```

```
class function getAndToInternalNr(andValue, Nr: TBitmask): TBitmask;

class function getAndToInternalNrAsBool(andValue, Nr: TBitmask): Boolean;
class function getAndToNrAsBool(andValue, Nr: Int64): Boolean;

class function getAndNotToInternalNr(Nr, andNotValue: TBitmask): TBitmask;
```

An den Objektklassen selbst stehen zur Umrechnung immer zur Verfügung:

```
function PrimaryKeyID: Int64; virtual;
function getNrAsBitMask: TBitmask;
class function convertBitFieldToMask( Nr: Int64 ): TBitmask;
class function convertToMask( Nr: Int64 ): TBitmask;
```

Scripting-Studio-Editor und Bedienung



(1) Auswahl des aktiven Scriptes

Es werden beim Laden eines Filter immer gleichzeitig drei Skripte geladen, um das spätere Handling deutlich zu vereinfachen:

Speicherung unter	Titel	Zielstellung	Sichtbarkeit
\Daten\Scripts\<Name>.script	Aktiver Filter	Produktiver Filter	Ja (ab 2.6) in den Filtermasken im Frontend außerhalb des Scripting-Studios

Speicherung unter	Titel	Zielstellung	Sichtbarkeit
\Daten\Scripts\<Name>.script-dev	Aktiver Filter ("Spielwiese")	Test/Entwicklungsversion des produktiven Filters für Weiterentwicklungen, ohne den produktiven "Filter" kaputt zu entwickeln. Jeder Filter besitzt immer einen produktive und eine Entwicklungs(Sandbox)-Version. Eine Entwicklungsversion kann zur Produktiv-Version gemacht werden (wird hier kopiert) über das Hauptmenü "Projekte / Entwicklungsversion live nehmen".	Nein nur im Scripting-Studio
\Daten\Scripts\Common.script	Gemeinsame Bibliothek	Hier sollte gemeinsam genutzte Funktionen und Hilfsroutinen ausgelagert werden, um effektiv in allen Filter zu arbeiten und Redundanzen zu vermeiden. Da auch das gegenseitige Einbinden der Standard-Scripte erlaubt ist, dient diese Sonderbehandlung aber der Effizienzsteigerung, um hier gezielte wiederverwendbare Funktionen und Procedures zu verlagern, um die eigentlichen Filter schlank und effizient zu halten.	Nein nur im Scripting-Studio
\Daten\Scripts\Common.script-dev	Gemeinsame Bibliothek ("Spielwiese")	Test/Entwicklungsversion der gemeinsam genutzten Bibliothek	Nein nur im Scripting-Studio

Hinweise zu Datensicherung und -Ablage

Da in Scripten sehr viel Zeit und Aufwand gesteckt werden kann, ist die Datensicherung nicht zu unterschätzen. Aktuell umgesetzt ist daher die Erstellung einer automatisches Backup-Version mit <Name>-backup-YYYYMMDD-HHMM bei jedem Speichervorgang (<F2>). Zu einem späteren Zeitpunkt ist auch die Synchronisation mit eigenen Dropbox-Instanzen vorgesehen. Aktuell werden diese Sicherungsdateien zu einem Script wieder gelöscht, wenn eine Script-Version "Verifiziert" wird (siehe Kontextmenü zu einem Filtereintrag (r.Maustaste über einen Filternamen)).

(1a) Einstellungen

Unterhalb der Script-Auswahl findet sich ein kleines ein/ausklappbares Pannel "Einstellungen".

Für weitere Ausbaustufen wurde sofort ein Einstellungs-Grid angelegt, was praktisch unendlich fortgeführt werden kann durch seine Scrollfähigkeit. Beim Start sind nur wenige Einstellungen vorhanden:

- ◦ Codebeispiele ausführbar: Dies bezieht sich auf die unter (4) gezeigten Code-Beispiele, die so beeinflusst werden. Im Standard sind die Beispiele nur als Hinweise für die abzubildende Mindeststruktur gedacht. Für die Sandbox ist es aber wahrscheinlich interessant auch lauffähige Codebeispiele zu haben. Die Einstellung wird geändert durch Klick in die Wert-Spalte, wo sich dann eine entsprechende Drop-Down-Box öffnet
  - ◦ Code-Folding nutzen: Im Programm können dann alle begin..end - Blöcke ein- und ausgeklappt werden. Dies erleichtert das Handling bei längeren Skripten.

(2) Bibliotheks-Zugriff

Um die Entwicklung zu unterstützen werden zwei Grundkonzepte mit unterstützt:

- Unterscheidung zwischen Entwicklung (Sandbox) und aktuell produktivem Code
- Unterstützung von Bibliotheken insb. auch gemeinsam genutzter Bibliotheken zwischen den unterschiedlichen Scripten

Mit Wechsel zwischen den Reitern wird zwischen den verschiedenen dahinterliegenden Code-Fragmenten gewechselt.

(3) Code

Hier befindet sich der Quelltexteditor der analog einem Notepad funktioniert mit den vorhandenen Tastenkombinationen (Strg-C-Kopieren, Strg-V-Einfügen etc.). Die Besonderheit ist hier, dass mit dem vorhandenen Code ein automatische Code-Highlighting erfolgt d.h. Schlüsselwörter oder Strukturen werden automatisch hervorgehoben in Schriftart und Form. So werden Kommentare beispielsweise immer kursiv/blau dargestellt.

Für die Entwicklung sind besonders folgende Funktionen relevant:

- Strg + "." zeigt die aktuell gültigen Funktionen/Konstanten/Variablen/Methoden an
- ( ) nach einer Procedure oder Funktion zeigt alle gültigen Werte an

Übergreifend unabhängig vom Quell-Code kann mit:

- F8 - Der Quellcode überprüft werden. Fehler werden in der Statusleiste direkt angezeigt und der Cursor springt automatisch zur Fehlerstelle
- F9 - Der Quellcode wird überprüft und danach ausgeführt. Im Datenverzeichnis wird im Erfolgsfall unter Unitname.PSC der übersetzte Quellcode angelegt und ab diesem Zeitpunkt kann dieser Filter als Bibliothek in anderen benutzt werden über "uses <Unitname>"

(4) Ausgabe

Dies ist eigentlich eines der Highlights der Umsetzung und ist auch bisher nur eine erste Version, die spätere deutlich ausgebaut wird. Ziel ist es die Ergebnisse aus einem Filter-Lauf übersichtlich und performant und unmittelbar darzustellen.

Die Darstellung wird dabei durch den Code festgelegt d.h. welche Spalten, welcher Detaillierungsgrad, welche Sortierungen und welche Darstellungs-Gruppierungen genutzt werden sollen.

In der Ausgabe wird auf eine spezielle zugreifbare Komponente zugegriffen die folgende Eigenschaften unterstützt:

- Nutzung von Gruppen, um Ergebnisse im Filtervorgang zu gruppieren. Im Standard sind aktuell 3 Gruppen angelegt
- Ein/Ausklappen von Gruppen
- Nutzung von Checkbox-Markierungen
- Mehrspaltiges Layout, womit Informationen anders und strukturiert ausgegeben werden können

(5) Sourcecode-Explorer

Der Source-Explorer versucht zeitnah synchron zum Quellcode die Struktur des Skriptes wiederzugeben, wobei unterschieden wird nach

- procedures: Alle Funktionen und Proceduren im Code, wobei Funktionen mit gelb markiert werden und proceduren mit grün

- uses: Alle eingebundenen Fremdbibliotheken. Dies kann die Common-Bibliothek sein (siehe (1)) oder alle anderen vorhandenen Skripte
- variables: Alle nicht lokalen Variablen (innerhalb einer procedure oder function definiert) werden hier gezeigt.

Mit Doppelklick auf einen Eintrag springt der Cursor automatisch an die zugehörige Programmcode-Position.

(6) Code-Snippet-Beispiele zur Übernahme

Dies sind aktuell nicht erweiterbare Code-Beispiele (TODO: Pflege sollte außerhalb der IDE möglich sein und auch über Updates ermöglicht werden z.B. ScriptStudio.MOD).

Mit Doppelklick auf einen Eintrag wird das Codebeispiel übernommen. Aktuell existiert eine Einstellung unter (1), um die Beispiele entweder nur als Struktur oder lauffähig zu verwenden.