COPADATA
do it your way

zenon guida

Interfacce di programmazione

v.7.11

©2013 Ing. Punzenberger COPA-DATA GmbH

Tutti i diritti riservati.

Tutti i diritti riservati la distribuzione e la copia - indifferentemente dal metodo - può essere consentita esclusivamente dalla dittaCOPA-DATA. I dati tecnici servono solo per la descrizione del prodotto e non rappresentano in alcun modo parti legali. Modifiche - anche sotto aspetti tecnici sono a noi riservate

Indice

1. Benvenuti nell'help COPA-DATA 6

2. Interfacce di programmazione 6

3. Process Control Engine (PCE) 9

3.1 L'editor PCE 9

3.1.1 Il task manager 10

3.1.2 Area di editazione 10

3.1.3 Finestra di emissione 11

3.1.4 I menù dell'editor PCE 11

3.1.5 La barra dei simboli dell'editor PCE 14

3.2 Procedura 15

3.2.1 Creazione di un task 15

3.2.2 Inserimento del codice 18

3.2.3 Funzione Show PCE 21

3.2.4 Esecuzione di task 21

3.3 VBScript - Introduzione 22

3.3.1 Tipi di dato 22

3.3.2 Variabile 24

3.3.3 Costanti 26

3.3.4 Operatori 27

3.3.5 Controllo dell'esecuzione del programma 29

3.3.6 Utilizzo di cicli per la ripetizione del codice. 32

3.3.7 Tipi di procedure 37

3.3.8 Convenzioni di scrittura del codice 39

4. Lista Macro 44

4.1 VBA: Barra simboli e Menu contestuale Visualizzazione dettagliata 46

4.2 VBA su sistemi a 64-Bit 50

4.3 Basi: 50

4.3.1 Proprietà degli oggetti - PROPERTIES 51

4.3.2 Metodi di oggetti - METHODS 51

4.3.3 Eventi di oggetti - EVENTS 52

4.3.4 VBA - modello oggetti in zenon 52
4.3.5 Come si utilizzano macro VBA? 54
4.3.6 Come si collega un elemento ein ActiveX in zenon? 56
4.3.7 Accesso mediante un programma esterno 58
4.3.8 Funzionalità di una variabile online 59
4.3.9 Lista bit di stato 61
4.3.10 Lasso per selezionare elementi dinamici nel Runtime. 64
4.4 Macro nell´editor 64
4.4.1 Barra degli strumenti "Lista macro" 65
4.4.2 Collegamento delle macro 67
4.5 Funzioni in zenon 68
4.5.1 Lanciare Macro-VBA 69
4.6 Sviluppo di wizard in VBA 71
4.6.1 Uso di un wizard 72
4.6.2 Struttura di nuovi wizard 73
4.6.3 Integrazione in VBA 73
4.6.4 Sviluppo di un wizard 73
4.6.5 Attualizzazione dei wizard. 79
4.7 Domande ricorrenti 80
4.7.1 Perché il pulsante rimane pigiato? 80
4.7.2 La macro non viene eseguita al primo clic. 80
4.7.3 Le macro non funzionano più nel RT? 81
4.7.4 Windows CE e VBA 81
4.8 Esempi 81
4.8.1 MouseEvents and ActiveX Control initialization 81
4.8.2 Visualizza informazioni variabili 82
4.8.3 Leggi e scrivi valori variabili 83
4.8.4 Leggi e scrivi variabili e implementa variabili online 84
4.8.5 Uso multiplo del dialogo 86
4.8.6 Alarm – Events and ActiveX Control handling 88
4.8.7 Accesso ad allarmi 90
4.8.8 Crea un interruttore (lavorare con variabili di processo) 92

5. VSTA 96

5.1 Basi: 97
5.1.1 Impostazione dell´ambiente VSTA 97

5.1.2 Accesso al modello oggetti in zenon 98
5.1.3 Funzioni in zenon 99
5.1.4 Debugging di un Add-in VSTA 100
5.1.5 Nuovi eventi in VSTA 101
5.1.6 Backup di progetti VSTA 101
5.2 Creazione di un progetto VSTA 102
5.2.1 Progetti VSTA nell´editor 102
5.2.2 Progetti VSTA nel runtime 103
5.2.3 Sviluppo di wizard in VSTA 104
5.3 Esempi 105
5.3.1 Creazione di variabili nell´editor di zenon 105
5.3.2 Scrittura di informazioni di progetto nella finestra di emissione di zenon 108
5.3.3 Leggi variabili in zenon mediante Regular Expressions 110

1. Benvenuti nell'help COPA-DATA

GUIDA GENERALE

Nel caso in cui non abbiate trovato delle informazioni che cercavate o se avete dei consigli relativi al completamento di questo capitolo dell'help, mandate una Mail a documentation@copadata.com (mailto:documentation@copadata.com).

SUPPORTO ALLA PROGETTAZIONE

Se avete delle domande concernenti progetti concreti, potete rivolgervi per E-Mail al support@copadata.com (mailto:support@copadata.com).

LICENZE E MODULI

Nel caso in cui doveste constatare che avete bisogno di altri moduli o licenze, rivolgetevi ai nostri dipendenti all'indirizzo sales@copadata.com (mailto:sales@copadata.com).

2. Interfacce di programmazione

In zenon sono a disposizione diverse interfacce per l´implementazione di programmi propri o per l´automatizzazione della progettazione:

- Process Control Engine (PCE) (A pagina: 9)
- Lista Macro (A pagina: 44)
- VSTA (A pagina: 96)

Informazioni sulla licenza

Contenuta nella licenza standard per Editor e Runtime

MENU

Punto di menù	Azione
Apri Editor PCE	Apre l'editor PCE
VBA-Visualizza Editor	Apre l'editor VBA
Apri Editor VSTA	Apre l'editor VSTA
Profilo editor	Apre la lista a cascata che serve alla selezione di un profilo editor.
Guida	Attiva la guida online

Informazioni su

Le informazioni relative alla creazione e all´implementazione di control (ActiveX, .NET, WPF) le trovate nella guida Controls.

Le informazioni concernenti la progettazione e l´uso dell´interfaccia SAP le trovate nel capitolo Interfaccia SAP.

APRI EDITOR VBA E VSTA.

EDITOR VBA

VBA avvia lo stesso ambiente di sviluppo per workspace e progetto. Per aprire l´editor VBA:

1. navigate per raggiungere il nodo**interfacce di programmazione**
2. cliccate poi con il tasto destro del mouse su Macro VBA.
3. selezionate nel menù contestuale **Apri VBA-Editor...**

Alternativa: premete la combinazione tasti Alt+F11

EDITOR VSTA

VSTA mette a disposizione ambienti di sviluppo separati per workspace e progetto. Se ne può usare solamente uno alla volta. Al momento dello start, ogni altro ambiente di sviluppo VSTA già aperto verrà chiuso.

Per aprire l´Editor VSTA per il workspace:

1. premete la combinazione tasti **Alt+F10**
2. verrà visualizzato il codice per il workspace e tutti i progetti caricati.

Per aprire l´Editor VSTA per il progetto attualmente caricato:

1. navigate per raggiungere il nodo**`interfacce di programmazione`**
2. cliccate poi con il tasto destro del mouse su VSTA.
3. selezionate nel menù contestuale **`Apri VSTA-Editor...`**
4. l´Editor verrà aperto per il progetto attualmente caricato.

API DALLA VERSIONE 7.10

Per l´uso delle interfacce zenon vale a partire da zenon 7.10 quanto segue:

- VSTA / .NET: **`.NET Framework 3.5 Service Pack 1`**deve essere installato.
- VBA: Se nel codice VBA si accede a Windows API o ad altre funzioni DLL importate, queste chiamate devono essere adattate a 64-bit. In generale vale quanto segue: un file VBA creato con una versione a 32 Bit non può essere usato in una versione di VBA a 64 Bit senza essere stato modificato.

> **Attenzione**
>
> Tenete presente che errori in applicazioni come ActiveX, PCE, VBA, VSTA, WPF e applicazioni esterne che accedono a zenon via API, possono condizionare anche la stabilità del Runtime.

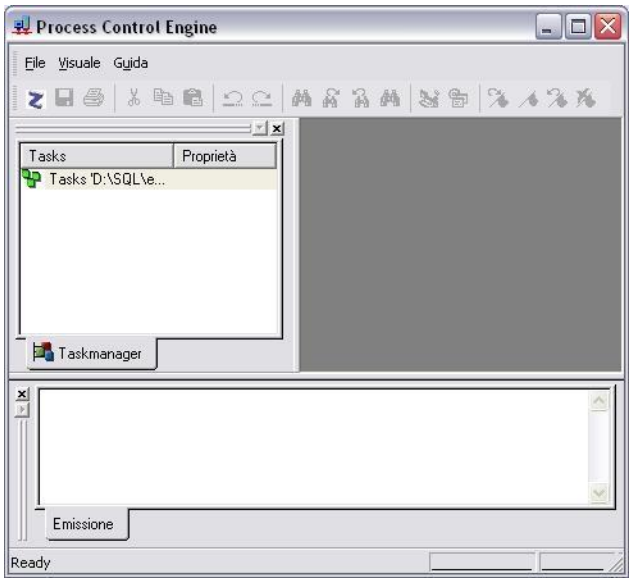
3. Process Control Engine (PCE)

Informazioni su

Il Process Control Engine (PCE) offre la possibilità di programmare dei processi ciclici in script VB o Java Script. La PCE si adatta particolarmente bene per funzioni dalla lunga durata, che sono attiva in backgroung (per es. funzioni di esportazione di dimensioni particolarmente grandi). Al contrario di VBA (A pagina: 44), la PCE supporta il Multithreading.

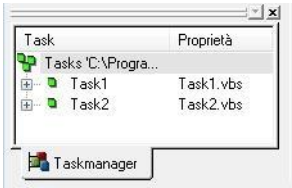
3.1 L'editor PCE

Il PCE si trova nel Manager di progetto alla voce '**`Scripting`**'. L'editor PCE si apre mediante il menu contestuale '**`Apri l'editor PCE`**'.

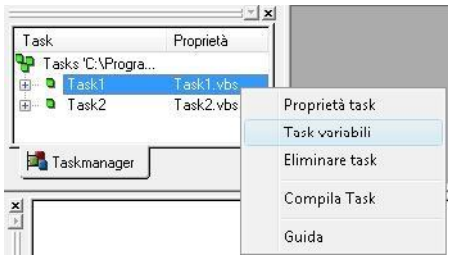


3.1.1 Il task manager

Il task manager dell'editor PCE mostra i task che sono stati creati e le variabili ad essi connesse.



Facendo clic due volte su un task lo si apre nell'editor. Con il tasto destro del mouse, invece, potete aprire il menu contestuale di un task.



Il menu contestuale di un task ha quattro voci:

Parametri	Descrizione
Proprietà task	Apre il dialogo "Proprietà" del task.
Task variabili	Apre il dialogo "Scelta variabili". In tal modo è possibile aggiungere ulteriori variabili al task.
Eliminare task	Cancela il task senza ulteriore interrogazione.
Compila Task	Compila il task.

3.1.2 Area di editazione

Nell´area di editazione dell´Editor PCE, si inserisce il codice dei task in VB Script o Java Script.

3.1.3 Finestra di emissione

3.1.4 I menù dell'editor PCE

Menu "File".

Il menu **File** contiene i seguenti comandi:

Parametri	Descrizione
Salva	Salva i task nuovi e quelli modificati.
Stampa	Stampa il task attivo al momento.
Chiudi	Chiude l'editor PCE.

Menu "Modifica".

Il menu **Modifica** contiene i seguenti comandi:

Parametri	Descrizione
Annulla	Annulla l'ultima azione eseguita.
Ripeti	Ripete l'ultima azione eseguita.
Taglia	Sposta un testo negli "appunti" di Windows.
Copiare	Copia un testo negli "appunti" di Windows.
Inserisci	Inserisce un testo preso dagli "appunti" di Windows.
Elimina	
Seleziona tutto	Seleziona l'intero testo del task.
Trova	Cerca un determinato testo nel task aperto.
Cercare successivo	Va al prossimo punto nel task in cui è stato rinvenuto l'elemento in questione.
Cercare precedente	Va al precedente punto nel task in cui è stato rinvenuto l'elemento in questione.
Sostituisci..	Sostituisce un testo del task con un altro.
Bookmark	Gestione di segnalibro nel codice del task.
Inserire bookmark	Inserisce nel punto segnato del codice un segnalibro.
Successivo bookmark	Si sposta nel codice al successivo segnalibro.
Precedente bookmark.	Si sposta nel codice al precedente segnalibro.
Elimina tutti i bookmark.	Elimina tutti i segnalibro dal codice.

Menu "Esegui".

Il menu **Esegui** contiene i seguenti comandi:

Salva tutti i task e rilancia	
Compila Task	Compila il task.

Menu "Visualizza"

Il menu **Visualizza** contiene i seguenti comandi:

Parametri	Descrizione
Impostazioni	Apre il dialogo "Impostazioni" dell'editor PCE.
Taskmanager	Apre/chiude il task manager
Finestra di emissione	Apre/chiude la finestra di emissione.
Status Bar	Apre/chiede la barra di stato

Menu "Finestra"

Il menu **Finestra** contiene i seguenti comandi:

Chiudi
Allinea
Dividi
Riordina simboli
Lista delle finestre aperte

Menu "Guida"

Il menu **Guida** contiene i seguenti comandi:

Funzionalità	Azione
Guida	Attiva la guida online
Informazioni su...	Aprite una finestra con informazioni su zenon: ▸ Numero di serie ▸ Numero di attivazione ▸ TAGs/IOs con licenza ▸ moduli con licenza Potete navigare all´interno della finestra-info servendovi di una barra di scorrimento. Cliccando all´interno della finestra o premendo il tasto **Esc** si esce dalla finestra-info.

3.1.5 La barra dei simboli dell'editor PCE

I più importanti comandi dell'editor PCE possono essere attivati cliccando sulle icone della barra dei simboli.

Sono disponibili i seguenti comandi:

Parametri	Descrizione
Chiudi	Chiude l'editor PCE.
Salva tutto	Salva i task nuovi e quelli modificati.
Stampare l´immagine attiva	Stampa il task attivo al momento.
Taglia	Sposta un testo negli "appunti" di Windows.
Copiare	Copia un testo negli "appunti" di Windows.
Inserisci	Inserisce un testo preso dagli "appunti" di Windows.
Annulla	Annulla l'ultima azione eseguita.
Ripeti	Ripete l'ultima azione eseguita.
Trova	Cerca un determinato testo nel task aperto.
Cercare successivo	Va al prossimo punto nel task in cui è stato rinvenuto l'elemento in questione.
Cercare precedente	Va al precedente punto nel task in cui è stato rinvenuto l'elemento in questione.
Sostituisci..	Sostituisce un testo del task con un altro.
Salva tutti i task e rilancia	
Avvia debugger	
Successivo bookmark	Si sposta nel codice al successivo segnalibro.
Inserisci boomark	Inserisce nel punto segnato del codice un segnalibro.
Precedente bookmark	Si sposta nel codice al precedente segnalibro.
Elimina tutti i bookmark.	Elimina tutti i segnalibro dal codice.

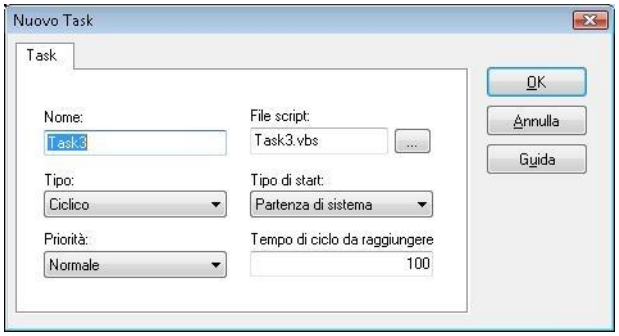
3.2 Procedura

3.2.1 Creazione di un task

Con il menù contestuale del taskmanager si può creare un nuovo task.

Proprietà del task

Dopo l'apertura del task si apre automaticamente il dialogo "Proprietà" dello stesso task.



Si possono impostare le seguenti proprietà:

Parametri	Descrizione
Nome	Denominazione univoca del task.
Tipo	I task possono essere eseguiti **ciclamente** o **una volta**. Ciclico: il task viene eseguito ciclicamente sulla base dell'intervallo impostato alla voce **Tempo di ciclo da raggiungere**. Unico: il task viene eseguito per una sola volta.
Priorità	Priorità di processo per il sistema operativo multithreading (**idle**,, **low**, **normal**, **high**,, **highest**,, **time critical)**.). Default: Normal Main process **:**: il task è attivo nello stesso thread del Runtime. Se il task finisce in un waiting loop o si blocca, ciò influenza anche il Runtime
File script	Scelta del file script: file VB (*.vbs) per VB Script oppure file JS (*.js) per Javascript. Quando si apre il task nell´area di editazione per la prima volta, vene creato il file corrispondente.
Tipo di start	System start : il task viene avviato automaticamente con il Runtime. (Questa è l´unica possibilità per usare PCE sotto Windows CE, visto che Windows CE non supporta VBA.) Gestito dagli eventi: il task viene avviato via una macro VBA e l´istruzione "thisProject.Tasks.Item("Taskname").Run".
Tempo di ciclo da raggiungere	Nel caso di task ciclici, il tempo di ciclo in millisecondi che deve essere raggiunto. Se ciò non accade, il task viene eseguito nel modo più veloce possibile.

Per apportare modifiche alle proprietà, si può aprire questo dialogo anche mediante il menù contestuale del task e il comando **Task-Proprietà...**.

Variabili del task

Dopo l´inserimento delle proprietà, si apre automaticamente il dialogo che serve alla selezione delle variabili. In questa sede vengono selezionate le variabili che devono essere editate nel task.

Qui devono essere collegate tutte le variabili che devono essere lette o scritte nel task. Esiste anche la possibilità di accedere alle variabili tramite l´oggetto variabili, ma solo le variabili collegate direttamente al task verranno lette automaticamente al momento dell´inizializzazione del task prima dell´esecuzione.

Le variabili devono avere la seguente sintassi:

```
Task.Value('Variable name')=123
```

Per apportare modifiche alla selezione variabili, si può aprire questo dialogo anche mediante il menù contestuale del task e il comando **`Task-Variabili...`**.

3.2.2 Inserimento del codice

Cliccando due volte su un task nel task manager, lo si apre nell'editor. Se il task viene aperto per la prima volta, viene creato in questo momento il corrispondente fil VBS o JS.

Vengono eseguite automaticamente quattro procedure:

Parametri	Descrizione
`Task_Init()`	Questa procedura viene eseguita automaticamente al momento dell'apertura del task.
`Task_Main()`	Questa procedura viene eseguita o una volta (tipo "unico") o ciclicamente (tipo "ciclico").
`Task_Exit()`	Questa procedura viene eseguita automaticamente prima della chiusura del task.
`Task_Timer(lTimerId)`	Questa procedura viene eseguita ciclicamente finché è in funzione il relativo timer. Il tempo di durata del ciclo viene trasmesso come parametro al momento dell'avvio del timer corrispondente.

Il PCE utilizza in linea di principio gli stessi modelli oggetti di VBA (vedi Tutorial VBA). Se si fa uso di oggetti VBA (a parte l'oggetto "task"), però, la capacità "multithreading" va perduta, visto che si può accedere a questi oggetti solo dal Hread principale.

Attenzione

Non tutte le funzioni dell´interfaccia COM sono compatibili con multithread e, dunque, possono essere usate solamente in un contesto main tread. Se viene settata nel task PCE una proprietà differente che nel "main process" , non è possibile alcun accesso da PCE al main thread. Nel caso ci sia ciononostante un accesso all´interfaccia COM, ciò può portare al verificarsi di sistemi di stato indefiniti, come, ad esempio, un Runtime freeze

Di grande importanza è, però, la collection Task e l'oggetto `Task`.

La Collection task

Count
Item
Parent

L'oggetto task

ActualCycleTime	Proprietà	Tempo ciclico del task effettivamente raggiunto
CountVariable	Proprietà	Numero delle variabili collegate al task.
CycleTime	Proprietà	Tempo impostato per il ciclo del task.
DynProperties	Proprietà	
ErrorNumber	Proprietà	
ErrorString	Proprietà	
Exit	Evento	
Init	Evento	
ItemVariable	Stile	
Main	Evento	
MemValue	Proprietà	Con Task.MemValue(Name)=value viene creata una variabile interna e le viene assegnato un valore. La variabile non deve essere dichiarata precedentemente. Questa variabili può essere letta anche da altri task. In tal modo essa permette di trasmettere un valore da un task ad un altro.
Nome	Proprietà	Nome del task attuale.
Parent	Proprietà	La Collection **Task**
Priority	Proprietà	Priorità del task attuale.
Run	Stile	Avvia un task
Sleep	Stile	Ferma un task.
StartTimer	Stile	Il metodo StartTimer avvia un timer del task.
Stato	Proprietà	
Stop	Stile	Arresta un task.
StopTimer	Stile	Il metodo StopTimer arresta un timer del task.
Timer	Evento	
Type	Proprietà	

`Value`	Proprietà	Con Task.Value(name of linked variable)=value si può assegnare un nuovo valore ad una variabile del progetto.

3.2.3 Funzione Show PCE

Con la funzione di zenon **`Show PCE`**, si può aprire l´Editor PCE dal Runtime.

3.2.4 Esecuzione di task

Esecuzione task con start di sistema

Se nella configurazione del task si è settata l´opzione `Tipo di start` su `Start di sistema` , il task stesso verrà eseguito automaticamente al momento dell´avvio del Runtime.

Questa è l´unica possibilità per usare PCE sotto Windows CE, visto che Windows CE non supporta VBA.

Esecuzione task gestita dagli eventi

Sul PC

Un task può essere avviato anche in modalità dipendente dagli eventi. In questo caso si deve settare il `Tipo di start` sull´opzione `Gestito da eventi` . Adesso il task non viene più avviato automaticamente con il Runtime.

Per poter avviare il task premendo un bottone, in seguito ad una violazione di valore limite, oppure in seguito ad un altro evento, si deve creare una macro VBA. L´istruzione VBA per lo start del task è la seguente:

```
thisProject.Tasks.Item(Taskname).Run
```

Questo task verrà avviato automaticamente in un thread proprio se nella configurazione non è stato indicata la **`Priorità Main Process`**.

Con la seguente istruzione VBA si può terminare il task in qualsiasi momento:

```
thisProject.Tasks.Item(Taskname).Stop
```

Sul terminale CE

Visto che Windows CE non supporta VBA, la procedura sopra descritta non è possibile su un terminale CE. C´è però la possibilità di richiamare anche qui delle task gestite dagli eventi.

A questo scopo si crea una task con il `Tipo di start System start` . Questo task, dunque, viene avviato automaticamente nel runtime. Questo task, inoltre, ottiene la **`Priorità Main Process`**, in modo tale che esso sia attivo nello stesso thread come il runtime. Con questo task vengono collegate solamente variabili bit preposte all´esecuzione gestita dagli eventi. L´istruzione per lo start di un altro task è la seguente:

Parent.Item(Taskname).Run

Questo task verrà avviato automaticamente in un thread proprio se nella configurazione non è stato indicata la **`Priorità Main Process`**.

Con la seguente istruzione si può terminare il task in qualsiasi momento:

Parent.Item(Taskname).Stop

3.3 VBScript - Introduzione

3.3.1 Tipi di dato

Variant

In VBScript esiste ufficialmente un solo tipo di variabile, denominato `Variant`. `Variant` è una forma particolare di tipo di dato che può contenere diversi tipi di informazioni a seconda del modo in cui viene utilizzata. Dato che `Variant` è l'unico tipo di dato disponibile in VBScript, è il tipo di dati restituito da tutte le funzioni di VBScript.

Nella forma più semplice, `Variant` può includere informazioni numeriche o stringhe. Una `Variant` è di tipo numerico se viene utilizzata in un contesto numerico, è di tipo stringa se utilizzata in un contesto stringa. Ciò significa che, se si lavora con dati simili a valori numerici, VBScript interpreterà tali dati come numeri e verranno eseguite le operazioni più appropriate. Naturalmente potete sempre fare in modo che i numeri si comportino come stringhe inserendoli fra virgolette (). In modo analogo, se si lavora con dati che possono essere solo interpretati come stringhe, essi verranno gestiti come dati stringa.

Sottotipi di variabili Variant

Oltre alla semplice classificazione di valore numerico o stringa, con le variabili `Variant` è possibile definire ulteriori distinzioni in base alla natura specifica delle informazioni numeriche Si possono, ad esempio, utilizzare informazioni numeriche cherappresentano una data o un orario. Insieme ad altri dati relativi alla data o all'orario, il risultato verrà sempre espresso come data o come orario. È tuttavia possibile utilizzare informazioni numeriche di vario tipo, dai valori booleani, ai numeri in virgola mobile di grandi dimensioni. Le varie categorie di informazioni che possono essere incluse in variabili `Variant` sono definite`sottotipi` . Per lo più basta assegnare ad una `Variant` i dati desiderati in modo tale che essa si comporti nel modo corrispondente.

Nella tabella seguente sono elencati i possibili sottotipi che una `Variant` può contenere.

Sottotipo	Descrizione
Empty	Variabile Variant non inizializzata. Il valore è 0 nel caso di variabili numeriche e una stringa di lunghezza zero () nel caso di variabili stringa.
Null	Variabile Variant che include dati non validi inseriti volutamente.
Boolean	Può assumere valore TRUE o FALSE.
Byte:	Contiene un intero compreso tra 0 e 255.
Integer	Contiene un intero compreso tra -32.768 e 32.767.
Currency	da -922,337,203,685,477.5808 a 922,337,203,685,477.5807.
Long	Contiene un intero compreso tra -2.147.483.648 e 2.147.483.647.
Single	Contiene un numero in virgola mobile e precisione singola compreso tra -3,402823E38 e -1,401298E-45 per valori negativie tra 1,401298E-45 e 3,402823E38 per valori positivi.
Double	Contiene un numero in virgola mobile e precisione doppiacompreso tra -1,79769313486232E308 e -4,94065645841247E-324 per valori negativi e tra 4,94065645841247E-324 e1,79769313486232E308 per valori positivi.
Date (Time)	Contiene un numero che rappresenta una data compresa tra l'1gennaio dell'anno 100 e il 31 dicembre del 9999.
String	Contiene una stringa di lunghezza variabile composta da unmassimo di circa 2 miliardi di caratteri.
Object	Contiene un oggetto.
Error	Contiene un numero di errore.

3.3.2 Variabile

Una variabile è un segnaposto che rimanda ad una posizione della memoria del computer in cui è possibile memorizzare informazioni di programma soggette a modifiche durante l'esecuzione degli script. Ad esempio, potete impostare una variabile dal nome KlickZaehler per memorizzare quante volte un utente clicca su un oggetto in una determinata Web-Seite. La posizione delle variabili in memoria non è importante dal punto di vista dell'utente. Per visualizzarne o modificarne il valore, è sufficiente farvi riferimento tramite il nome. In VBScript le variabili hanno sempre alla base il tipo di dato Variant.

Dichiarazione di variabili

Dichiarate le variabili esplicitamente nel vostro script usando le istruzioni Dim, quelle Public e quelle Private. Esempio:

```
Dim DegreesFahrenheit
```

Dichiarate più variabili separandone i nomi con una virgola. Esempio:

```
Dim Top, Bottom, Left, Right
```

Restrizioni della nomenclatura

I nomi variabili obbediscono alle regole standard della nomenclatura di elementi di lingua in VBScript. Per i nomi variabili vale quanto segue:

- Devono iniziare con un carattere dell´alfabeto.
- Non possono contenere un punto.
- Non possono essere piú lunghi di 255 caratteri.
- Devono essere univoci nell´ambito dell´area di validità per cui vengono dichiarati.

Ambito e durata di variabili.

Se dichiarate una variabile in una routine, solamente il codice all´interno di questa routine può accedere o modificare il valore di questa variabile. Il che vuol dire che essa ha un ambito locale ed è una variabile a livello di routine.

Se dichiarate una variabile al di fuori di una routine, la rendete visibile per tutte le routine del vostro script. Questa è una variabile a livello script e ha un ambito a livello di script.

Il tempo durante il quale una variabile conserva il proprio valore è noto come durata. La durata di una variabile a livello script va dal momento della dichiarazione a quello della chiusura dello script stesso. A livello di routine, una variabile esiste finché vi trovate nella routine. Quando quest´ultima viene terminata, la variabile viene distrutta.

Le variabili locali sono ideali spazi di salvataggio temporanei durante l´esecuzione di una routine. Potete avere variabili locali dello stesso nome in più routine differenti perché ognuna viene riconosciuta solamente dalla routine in cui è stata dichiarata.

Assegnazione di valori alle variabili

Per assegnare valori alle variabili, è necessario creare un'espressione indicando a sinistra la variabile e a destra il valore che si desidera assegnare. Esempio:

```
B = 200
```

Dati scalari e campi dato

Nella maggior parte dei casi alle variabili dichiarate viene assegnato un solo valore. Una variabile contenente un solo valore è definita variabile scalare. A volte, tuttavia, risulta utile assegnare a una singola variabile più valori. In questi casi, potete creare una variabile che può contenere una serie di valori. Una variabile contenente una serie di valori è definita variabile matrice. La dichiarazione di queste variabili è simile a quella delle variabili scalari. La sola differenza che, nelle dichiarazioni di variabili matrici, il nome della variabile è seguito da parentesi. Nell'esempio seguente, viene dichiarata una matrice a una soladimensione contenente 11 elementi:

```
Dim A(10)
```

Anche se tra parentesi è indicato il numero 10, la matrice include in effetti 11 elementi (numerati da 0 a 10). In VBScript infatti le matrici sono sempre in base zero edi conseguenza il numero degli elementi in esse contenuti corrisponde sempre al numero indicato tra parentesi più uno. Queste matrici sono definite a dimensione fissa.

Per assegnare dati a ciascun elemento di una matrice, è necessario utilizzare un indice. Nell'esempio seguente vengono assegnati dati numerici ai vari elementi della matrice, usando appunto gli indici:

```
A(0) = 256
A(1) = 324
A(2) = 100
. . .
```

```
A(10) = 55
```

Per recuperare dati da un elemento della matrice, è possibile procedere nello stesso modo, ovvero inserire un indice nell'elemento desiderato. Esempio:

```
. . .
AltraVariabile = A(8)
. . .
```

Le matrici non sono limitate a una sola dimensione. È infatti possibile specificare fino a 60 dimensioni, anche se nella maggior parte dei casi ne vengono specificate solo tre o quattro. Per dichiarare dimensioni multiple, è necessario separare con unavirgola i valori delle dimensioni indicati tra parentesi. Nell'esempio seguente la variabile MiaTabella è una matrice a due dimensioni composta da 6 righe e 11 colonne:

```
Dim MiaTabella(5, 10)
```

In una matrice a due dimensioni il primo numero corrisponde sempre al numero di righe, il secondo al numero di colonne.

È inoltre possibile dichiarare matrici le cui dimensioni vengono modificate durante l'esecuzione dello script. Questo tipo di matrici sono definite matrici dinamiche. Vengono inizialmente dichiarate in una routine utilizzando l'istruzione `Dim` come qualsiasi altra matrice, oppure l'istruzione `ReDim` . La differenza è che tra parentesi non viene indicata alcuna grandezza o numero di dimensioni. Esempio:

```
(Dim MioArray)
ReDimAltroArray)
```

Per utilizzare una matrice dinamica, è necessario specificare più istruzioni `ReDim` in modo da determinare il numero di dimensioni e la grandezza di ciascuna dimensione. Nell'esempio seguente l'istruzione `ReDim` imposta la grandezza iniziale della matrice dinamica su 25 Una successiva istruzione `ReDim` ridimensiona quindi la matrice impostandola su 30 e viene utilizzata la parola chiave `Preserve` per mantenere il contenuto della matrice durante l'operazione di ridimensionamento.

```
ReDim MioArray(25)
. . .
ReDim Preserve MioArray(30)
```

3.3.3 Costanti

Una costante è un nome significativo non soggetto a modifiche utilizzato in sostituzione di un numero o di una stringa. VBScript definisce un numero di costanti integrate. Potete ottenere ulteriori informazioni su queste costanti integrate nel VBScript Language Reference.

È anche possibile creare costanti personalizzate (o definite dall'utente) in VBScript mediante l'istruzione Const. In tal modo potete assegnare un nome significativo a costanti stringhe o costanti numeriche. In seguito potete assegnargli valori letterali e usarle nel vostro script. Esempio:

```
Const MyString = "This is a string."
Const MyAge = 49
```

Tenete presente che la stringa letterale è compresa fra virgolette (" "). Le virgolette sono il modo più chiaro per contraddistinguere i valori stringa dai valori numerici. È possibile rappresentare i valori letterali di data e ora racchiudendoli tra simboli di cancelletto (#); Esempio:

```
Const LaData = #6-1-97#
```

Visto che non esiste alcuna differenza funzionale fra costanti create in tal modo e variabili normali, è consigliabile adottare una convenzione di denominazione in modo da contraddistinguere le costanti dalle variabili. In tal modo si può evitare di assegnare un nuovo valore ad una costante durante l'esecuzione dello script. Potete usare, ad esempio, il prefisso vb oppure con per i vostri nomi costante, oppure potete scriverli a lettere maiuscole. In ogni caso, assicuratevi che sia possibile distinguere fra costanti e variabili. In tal modo eviterete problemi quando create script complessi.

3.3.4 Operatori

VBScript dispone di una molteplicità di operatori, fra gli altri operatori aritmetici, di confronto , di concatenazione e logici.

Precedenza tra operatori.

Quando in un'espressione si eseguono varie operazioni, ciascuna parte viene valutata e risolta secondo un ordine specifico definito precedenza tra operatori. Questa successione viene definita ordine di precedenza Per ignorare l'ordine di precedenza in modo che determinate parti di un'espressione vengano valutate prima di altre, è necessario utilizzare le parentesi. Le operazioni racchiuse tra parentesi vengono sempre valutate per prime. All'interno delle parentesi tuttavia viene mantenuta la normale precedenza tra operatori.

Nelle espressioni che includono operatori di categorie diverse, vengono innanzitutto valutati gli operatori aritmetici, quindi gli operatori di confronto e infine gli operatori logici. Gli operatori di confronto hanno la stessa precedenza, ovvero vengono valutati da sinistra a destra nello stesso ordine in cui sono stati specificati. Gli operatori aritmetici e logici vengono valutati in base all'ordine di precedenza indicato di seguito.

Operatori aritmetici

Descrizione	Simbolo
Elevamento a potenza	^
Operatore di negazione unario	-
Moltiplicazione	*
Divisione	/
Divisione intera	/
Divisione con modulo	Mod
Addizione	+
Sottrazione	-
Concatenazione di stringhe	

Operatori di confronto

Descrizione	Simbolo
Uguaglianza	=
Disuguaglianza	<>
Minore di	<
Maggiore di	>
Minore o uguale a	<=
Maggiore o uguale a	>=
Equivalenzatra oggetti	Is

Operatori logici

Quando in un'espressione si eseguono varie operazioni, ciascuna parte viene valutata e risolta secondo un ordine specifico definito precedenza tra operatori. Questa successione viene definita ordine di precedenza Per ignorare l'ordine di precedenza in modo che determinate parti di un'espressione vengano valutate prima di altre, è necessario utilizzare le parentesi. Le operazioni racchiuse tra parentesi vengono sempre valutate per prime. All'interno delle parentesi tuttavia viene mantenuta la normale precedenza tra operatori.

Nelle espressioni che includono operatori di categorie diverse, vengono innanzitutto valutati gli operatori aritmetici, quindi gli operatori di confronto e infine gli operatori logici. Gli operatori di confronto hanno la stessa precedenza, ovvero vengono valutati da sinistra a destra nello stesso ordine in cui sono stati specificati. Gli operatori aritmetici e logici vengono valutati in base all'ordine di precedenza indicato di seguito.

3.3.5 Controllo dell'esecuzione del programma

È possibile controllare il flusso di script tramite istruzioni condizionali e istruzioni di ciclo. Con le istruzioni condizionali è possibile scrivere codice VBScript che consente di prendere decisioni e ripetere azioni.

Prendere decisioni con l´aiuto di If...Then...Else

L'istruzione `If...Then...Else` consente di valutare se una condizione è `True` o `False` e, a seconda del risultato, di specificare una o più istruzioni da eseguire. In genere, la condizione corrisponde a un'espressione in cui due valori o due variabili vengono confrontati tramite un operatore di confronto. Le informazioni sugli operatori di confronto le trovate al capitolo "Operatori di confronto". Le istruzioni `If...Then...Else` possono essere nidificate in un numero qualsiasi di livelli di nidificazione.

Esecuzione di istruzioni se una condizione è true.

Per eseguire solo un´istruzione quando una condizione è `True` , usate la sintassi a linea singola per l´istruzione `If...Then...Else` . Quello che segue è un esempio di sintassi a linea singola. Tenete presente che in questo esempio la parola chiave `Else` viene omessa.

```
Sub FixDate()
Dim myDate
myDate = #2/13/95#
If myDate < Now Then myDate = Now
End Sub
```

Per eseguire più di una linea di codice, dovete usare la sintassi a linea multipla o block syntax. Questa sintassi contiene l´istruzione `End If` , come mostra il seguente esempio:

```
Sub AlertUser(value)
If value = 0 Then
AlertLabel.ForeColor = vbRed
AlertLabel.Font.Bold = True
AlertLabel.Font.Italic = True
End If
End Sub
```

Per eseguire quando una condizione è `True` , usate la sintassi a linea singola per l´istruzione `If...Then...Else` . Quello che segue è un esempio di sintassi a linea singola. Tenete presente che in questo esempio la parola chiave `Else` viene omessa.

```
Sub FixDate()
Dim myDate
myDate = #2/13/95#
If myDate < Now Then myDate = Now
End Sub
```

Per eseguire più di una linea di codice, dovete usare la sintassi a linea multipla o block syntax. Questa sintassi contiene l´istruzione `End If` , come mostra il seguente esempio:

```
Sub AlertUser(value)
If value = 0 Then
AlertLabel.ForeColor = vbRed
AlertLabel.Font.Bold = True
AlertLabel.Font.Italic = True
End If
End Sub
```

Esecuzione di determinate istruzioni se una condizione è true, ed esecuzione di altre istruzioni se una condizione è false.

Con l´istruzione `If...Then...Else` potete definire due blocchi con istruzioni eseguibili: un blocco viene eseguito se la condizione è `True` , l´altro se la condizione è `False` .

```
Sub AlertUser(value)
If value = 0 Then
AlertLabel.ForeColor = vbRed
AlertLabel.Font.Bold = True
AlertLabel.Font.Italic = True
Else
AlertLabel.Forecolor = vbBlack
AlertLabel.Font.Bold = False
AlertLabel.Font.Italic = False
```

```
End If
End Sub
```

Decidere fra più alternative

Una variazione dell´istruzione `If...Then...Else` vi consente di scegliere fra più alternative. Aggiungendo clausole `ElseIf` potete estendere la funzionalità dell´istruzione `If...Then...Else` in modo tale da poter controllare il corso del programma sulla base di diverse possibilità.

```
Esempio:
Sub ReportValue(value)
If value = 0 Then
MsgBox value
ElseIf value = 1 Then
MsgBox value
ElseIf value = 2 then
Msgbox value
Else
Msgbox Msgbox value outside the region!
End If
```

Prendere decisioni con Select Case

La struttura`Select Case` rappresenta un'alternativa all'istruzione `If...Then...ElseIf` per la selezione di un blocco di istruzioni specifico tra più blocchi diversi La funzione dell'istruzione `Select Case` è simile a quella dell'istruzione `If...Then...Else` , con ladifferenza tuttavia che il codice risulta più efficiente e spesso di più facile lettura.

Una struttura `Select Case` include una singola espressione di testo valutata una sola volta all'inizio della struttura stessa. Il risultato dell'espressione viene quindi confrontato con i valori di ciascun blocco `Case` della struttura. Se viene individuata una corrispondenza, il blocco di istruzioni associato al blocco `Case` specifico verràeseguitot:

```
Select Case Document.Form1.CardType.Options(SelectedIndex).Text
Case MasterCard
DisplayMCLogo
ValidateMCAccount
Case Visa
DisplayVisaLogo
ValidateVisaAccount
Case American Express
DisplayAMEXCOLogo
ValidateAMEXCOAccount
Case Else
```

```
DisplayUnknownImage
PromptAgain
End Select
```

3.3.6 Utilizzo di cicli per la ripetizione del codice.

L'esecuzione di cicli consente di eseguire ripetutamente un gruppo di istruzioni. In alcuni cicli le istruzioni vengono ripetute fino a quando una condizione risulta falsa (`False`), mentre in altri vengono ripetute fino a quando una condizione risulta vera (`True`). Esistono inoltre cicli in cui le istruzioni vengono ripetute un numero specifico di volte.

In VBScript sono disponibili le seguenti istruzioni per l'esecuzione di cicli:

Parametri	Descrizione
Utilizzo di cicli Do (A pagina: 32):	le istruzioni vengono ripetute fino a quando una condizione risultaTrue o rimane
Utilizzo di cicli While...Wend (A pagina: 35):	le istruzioni vengono ripetute fino a quando una condizioneè True.
Utilizzo di cicli For...Next (A pagina: 35):	le istruzioni vengono ripetute il numero di volte specificato da uncontatore.
Utilizzo di cicli For Each...Next (A pagina: 36):	un gruppo di istruzioni viene ripetuto per ciascun elemento in una lista o ogni ele

Usare cicli Do

E' possibile utilizzare le istruzioni `Do...Loop` per eseguire un blocco di istruzioni un numero indefinito di volte, senza che il numero delle ripetizioni sia fissato. Le istruzioni vengono ripetute mentre una condizione è `True` , oppure fino a quando risulta `True` .

Ripetizioni di istruzioni mentre una condizione è True

È necessario utilizzare la parola chiave `While` per verificare una condizione in un'istruzione `Do...Loop` . La verifica può essere eseguita prima dell'inizio del ciclo, comeillustrato nell'esempio ChkPrimaWhile riportato di seguito, oppure dopo almeno una esecuzione del ciclo, come illustrato nell'esempio ChkDopoWhile: Nella routine ChkPrimaWhile, se myNum è impostato su 9 anziché su 20, le

istruzioni incluse nel ciclo non verranno mai eseguite. Nella routine ChkDopoWhile, le istruzioni incluse nel ciclo vengono eseguite una sola volta, in quanto la condizione ègià `False` .

```
Sub ChkPrimaWhile()
   Dim counter, mioNum
   counter = 0
   mioNum = 20
   Do While mioNum> 10
       mioNum = mioNum - 1
       counter = counter + 1
   Loop
   MsgBox  "Il ciclo ha eseguito "& counter & " ripetizioni.".
End Sub

SubChkDopoWhile()
   Dim counter, mioNum
   counter = 0
   mioNum = 9
   Do
       mioNum = mioNum - 1
       counter = counter + 1
   Loop While mioNum > 10
   MsgBox  "Il ciclo ha eseguito "& counter & " ripetizioni.".
End Sub
```

Ripetizione di un´istruzione finché una condizione non risulta True

È possibile utilizzare la parola chiave `Until` in due modi diversi per verificare una condizione in un'istruzione `Do...Loop` :. La verifica può essere eseguita prima dell'inizio del ciclo, come illustrato nell'esempio ChkPrimaUntil riportato di seguito, oppure dopo almeno una esecuzione del ciclo, come illustrato nell'esempio ChkDopoUntil. Il ciclo viene ripetuto finché la condizione risulta `False` .

```
Sub ChkPrimaUntil()
   Dim counter, mioNum
   counter = 0
   mioNum = 20
   Do Until mioNum = 10
       mioNum = mioNum - 1
       counter = counter + 1
```

```
    Loop
    MsgBox  "Il ciclo ha eseguito "& counter & " ripetizioni.".
End Sub

Sub ChkDopoUntil()
    Dim counter, mioNum
    counter = 0
    mioNum = 1
    Do
        mioNum = mioNum + 1
        counter = counter + 1
    Loop Until mioNum = 10
    MsgBox  "Il ciclo ha eseguito "& counter & " ripetizioni.".
End Sub
```

Uscire da un´istruzione Do...Loop dall´interno del Loop

Potete terminare un `Do...Loop` usando l´istruzione `Exit Do` . Visto che normalmente si desidera terminare l´esecuzione solamente in determinate situazioni (come, per esempio, per evitare un loop infinito), dovreste usare l´istruzione `Exit Do` nel blocco istruzioni `True` di un´istruzione `If...Then...Else` . Se la condizione è `False` , il loop continuerà ad essere eseguito in modo normale.

Nell´esempio seguente, a myNum è assegnato un valore che provoca un loop infinito. L´istruzione `If...Then...Else` controlla se si verifica questo stato e impedisce la ripetizione senza fine.

```
Sub ExitExample()
    Dim counter, mioNum
    counter = 0
    mioNum = 9
    Do Until mioNum = 10
        mioNum = mioNum - 1
        counter = counter + 1
        If myNum < 10 Then Exit Do
    Loop
    MsgBox  "Il ciclo ha eseguito "& counter & " ripetizioni.".
End Sub
```

Utilizzo di While...Wend

In VBScript, l´istruzione `While...Wend` viene messa a disposizione di sviluppatori che hanno familiarità con il suo uso. sono A causa della limitata flessibilità di `While...Wend` , si consiglia di usare al suo posto `Do...Loop` .

Uso di For...Next

Potete usare le istruzioni `For...Next` per definire che un blocco di istruzioni deve essere eseguito un numero specifico di volte. Nel caso di cicli, è necessario utilizzare una variabile contatore ilcui valore viene aumentato o diminuito a ogni ripetizione del ciclo.

Nella routine riportata di seguito, ad esempio, la routine MioCiclo viene eseguita 50 volte. L'istruzione `For` specifica la variabile contatore x e il corrispondente valore iniziale e finale L'istruzione `Next` consente di aumentare `la variabile contatore` con incrementi di una unità:

```
Sub DoMioCiclo50Volte()
   Dim x
   For x = 1 To 50
       MyProc
   Next
End Sub
```

La parola chiave `Step` consente di aumentare o diminuire la variabile contatore del valore specificato. Nell'esempio riportato di seguito la variabile contatore viene incrementata di due unità a ogni ripetizione del ciclo. Al completamento del ciclo, il totale corrisponde alla somma di 2, 4, 6, 8 e 10.

```
Sub DoMyProc50Times()
   Dim x
   For x = 1 To 50
       MyProc
   Next
End Sub
```

Per diminuire la variabile contatore, è necessario utilizzare un valore `Step` negativo. Dovete specificare un valore finale minore del valore iniziale.. Nell'esempio riportato di seguito, la variabile contatore myNum viene diminuita di due unità a ogni ripetizione del ciclo. Al completamento del ciclo, il totale corrisponde alla somma di 16, 14, 12, 10, 8, 6, 4 e 2:

```
Sub NuovoTotal()
   Dim mioNum, totale
   For mioNum = 16 To 2 Step -2
       totale = totale + mioNum
```

```
    Next
    MsgBox 'Il totale è" & total
End Sub
```

Uso dell´istruzione For Each...Next

Un ciclo `For Each...Next` è simile a quello `For...Next` . La sola differenza è che le istruzioni non vengono ripetute il numero di volte specificato, ma in un ciclo `For Each...Next` viene eseguito un gruppo istruzioni per ogni elemento di una lista di oggetti, oppure per ogni elemento in una matrice. Ciò risulta particolarmente utile quando non si conosce a priori il numero di elementi di un insieme.

Nell'esempio di codice HTML riportato di seguito, il contenuto di un oggetto `Dictionary` viene utilizzato per inserire testo in varie caselle di testo:

```
<HTML>
<HEAD><TITLE>Forms and Elements</TITLE></HEAD>
<SCRIPT LANGUAGE='VBScript'>
<!--
Sub cmdChange_OnClick
  Dim d                        'Crea una variabile
   Set d = CreateObject('Scripting.Dictionary')
  d.Add '0', 'Athen'        'Aggiunge alcune chiavi ed elementi
  d.Add '1', "Belgrado"
  d.Add '2', "Cairo"

  For Each I in d
      Document.frmForm.Elements(I).Value = D.Item(I)
  Next
End Sub
-->
</SCRIPT>
<BODY>
<CENTER>
<FORM NAME='frmForm'

<Input Type = 'Text'><p>
```

```
<Input Type = 'Text'><p>
<Input Type = 'Text'><p>
<Input Type = 'Text'><p>
<Input Type = 'Button' NAME='cmdChange' VALUE='Fare clic qui'><p>
</FORM>
</CENTER>
</BODY>
</HTML>
```

3.3.7 Tipi di procedure

Sub-routine

Una routine `Sub` è una successione di istruzioni VBScript che sono comprese fra le istruzioni `Sub` e `End Sub` e che eseguono azioni senza restituire alcun valore. In una routine `Sub` è possibile specificare argomenti, ovvero costanti, variabili o espressioni passate dallaroutine che sta eseguendo una chiamata. Se non si specifica alcun argomento nella routine `Sub` , nella corrispondente istruzione `Sub` è necessario includere parentesi vuote ()

Nella routine `Sub` riportata di seguito, vengono utilizzate le due funzioni VBScript intrinseche, ovvero incorporate, `MsgBox` e `InputBox` che consentono di richiedere informazioni all'utente. Viene quindi visualizzato il risultato di un calcolo basato sulleinformazioni fornite. Questo calcolo è eseguito in una routine `Function` creata in VBScript. Anche quest´ultima viene mostrata nel trattamento della successiva routine `Function` .

```
Sub ConvertTemp()
  temp = InputBox("Immettere la temperatura in gradi Fahrenheit.", 1)
  MsgBox "La temperatura è pari a "& Celsius(temp) & '"gradi C."
End Sub
```

Routine function

Una routine `Function` è costituita da una serie di istruzioni Visual Basic racchiuse tra le istruzioni `Function` e `End Function` . Una routine `Function` è simile ad una `Sotto` routine, ma può anche restituire un valore. Una routine `Function` può accettare argomenti, come costanti, variabili o espressioni, passati a essa dal codice chiamante. Se una routine `Function` non ha argomenti, la

sua istruzione Function deve contenere una coppia di parentesi vuota. Una routine Function restituisce valori assegnando un valore al proprio nome di funzione in una o più istruzioni della routine. Il tipo di valore restituito di un una routine Function è sempre Variant .

Nel seguente esempio, la funzione celsius calcola a partire da gradi Fahrenheit i corrispondenti gradi Celsius . Quando la funzione è chiamata dalla Sotto procedura ConvertTemp, una variabile contenente l´argomento value viene viene passata alla funzione. Il risultato del calcolo viene restituito alla routine chiamante e visualizzato in un apposito message box.

```
Sub ConvertTemp()
   temp = InputBox("Immettere la temperatura in gradi Fahrenheit.", 1)
   MsgBox "La temperatura è pari a "& Celsius(temp) & '"gradi C."
End Sub

Function Celsius(fDegrees)
   Celsius = (fDegrees - 32) * 5 / 9
End Function
```

Trasmissione di dati a o da routine

I dati vengono passati alla routine tramite argomenti. Essi fungono da segnaposto per i dati che vogliono passare alla loro routine. Quando si crea una routine utilizzando l'istruzione Sub - o Function , è necessario far seguire il nome della routine da parentesi. All'interno delle parentesi vengono inseriti gli argomenti separati da una virgola. Nell'esempio seguente, **fDegrees** è il segnaposto delvalore da convertire passato alla funzione Celsius:

```
Function Celsius(fDegrees)
  Celsius = (fDegrees - 32) * 5 / 9
End Function
```

Utilizzo di routine Sub e Function nel codice

Per usare una routine Function nel vostro codice, dovete usarla a destra di un´assegnazione variabile, oppure in un´espressione.

Esempio:

```
Temp = Celsius(GradF)
```

oppure

```
MsgBox 'The temperature is '' & Celsius(temp) & ' degrees C.'
```

Per richiamare una routine `Sub` da un'altra routine, è sufficiente specificarne il nome insieme ai valori degli argomenti obbligatori, separati da una virgola. L'istruzione `Call` non è obbligatoria. Se viene utilizzata, è necessario racchiudere gli eventuali argomenti tra parentesi.

Nell'esempio seguente vengono eseguite due chiamate alla routine MiaRoutine. In un caso l'istruzione `Call` è utilizzata nel codice, nell´altro no. Entrambe le chiamate eseguono la stessa operazione

```
Call MiaRoutine(primoarg, secondoarg)
MiaRoutine primoarg, secondoarg
```

3.3.8 Convenzioni di scrittura del codice

Le convenzioni di scrittura del codice sono proposte che consentono di facilitare la scrittura del codice con la Microsoft Visual Basic Scripting Edition.

Tali convenzioni per la programmazione comprendono, tra l´altro, le seguenti direttive:

Convezioni di denominazione per oggetti, variabili e routine
Convenzioni per i commenti
Indicazioni generali per la formattazione e il rientro del testo

Lo scopo principale delle convenzioni di scrittura del codice è uniformare la struttura e lo stile di uno script o gruppo di script in modo che il codice risulti semplice da leggere e comprensibile a tutti. Grazie alla definizione di buone convenzioni per la programmazione è possibile ottenere codice sorgente preciso, leggibile e chiaro, in conformità con il codice di altri linguaggi e quanto più possibile intuitivo.

Convenzioni di denominazione delle costanti.

Nelle versioni precedenti di VBScript non era disponibile alcun metodo di creazione di costanti definite dall'utente. Se utilizzate, le costanti venivano implementate come variabili e distinte dalle altre variabili con l'utilizzo solo di caratteri maiuscoli. Le parole che componevano il nome venivano separate con un carattere di sottolineatura (_).

Esempio:

```
USER_LIST_MAX
NEW_LINE
```

È ancora possibile utilizzare questo metodo per identificare le costanti, ma si consiglia di utilizzare uno schema di denominazione alternativo. Potete creare vere costanti con l'istruzione `Const` . Questa convenzione consente di utilizzare un formato concaratteri maiuscoli e minuscoli in cui i nomi delle costanti sono preceduti dal prefisso "con".

Ad esempio:

```
conCostantePersonale
```

Convenzioni di denominazione delle variabili

Per motivi di leggibilità e uniformità, in VBScript i nomi delle variabili devono essere descrittivi e preceduti dai prefissi elencati nella tabella seguente.

Sottot ipo	Prefisso	Esempio
Boolean	bln	blnFound
Byte:	byt	bytRasterData
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFirstName

Ambito di variabili

Le variabili dovrebbero essere sempre definite con l´ambito più limitato possibile. Le variabili VBScript possono avere uno dei seguenti ambiti.

Ambito	Dichiarazione in	Visibilità
Livello routine	Evento, funzione, o subroutine.	Visibile nella routine in cui è stata dichiarata.
Livello script	Sezione HEAD di una pagina HTML, al di fuori di ogni routine.	Visibile in ogni routine dello script.

Prefissi per l´area di validità delle variabili.

Con script di grandi dimensioni diventa particolarmente importante poter contraddistinguere l'area di validità delle variabili. A tale scopo è sufficiente aggiungere un prefisso di una lettera al prefisso del tipo di variabile in modo da evitare nomi eccessivamente lunghi.

Ambito	Prefisso	Esempio
A livello di routine	Nessuno	dblVelocity
A livello di script	s	sblnCalcInProgress

Nomenclatura valida di variabili e procedure

Nel corpo del nome variabile o procedura si dovrebbe utilizzare lettere maiuscole e minuscole. Il nome dovrebbe essere lungo abbastanza per descrivere lo scopo della variabile. Inoltre, i nomi procedura dovrebbero iniziare con un verbo, come, per es., InitNameArray o CloseDialog.

Per quello che concerne le espressioni di uso frequente o di lunghe dimensioni, si raccomanda l´uso di abbreviazioni standard, in modo tale da limitare il nome ad una lunghezza ragionevole. In generale si può dire che nomi variabile più lunghi di 32 caratteri sono difficili da leggere. Quando usate delle abbreviazioni, accertatevi che siano le stesse nell´intero script. L´alternanza casuale fra, per es., Cnt e Count all´interno di uno script o serie di script può generare confusione.

Convenzioni di denominazione degli oggetti.

Nella tabella seguente sono elencate le convezioni di denominazione di vari oggetti utilizzati nella programmazione in VBScript:

Tipo di oggetto	Prefisso	Esempio
Pannello 3D	pnl	pnlGroup
Pulsante animato	ani	aniMailBox
Casella di controllo	chk	chkReadOnly
Casella combinata (combo box) o di riepilogo a discesa (drop-down list box)	cbo	cboEnglish
Pulsante di comando	cmd	cmdExit
Finestra di dialogo comune	dlg	dlgFileOpen
Cornice	fra	fraLanguage
Barra di scorrimento orizzontale	hsb	hsbVolume
Immagine	img	imgIcon
Etichetta	lbl	lblHelpMessage
Linea	lin	linVertical
Casella di riepilogo (list box)	lst	lstPolicyCodes
Pulsante di selezione (Spin)	spn	spnPages
Casella di testo (text box)	txt	txtLastName
Barra di scorrimento verticale	vsb	vsbRate
Dispositivo di scorrimento (Slider)	sld	sldSkala

Convenzioni di scrittura dei commenti

Tutte le routine dovrebbero iniziare con un breve commento che ne descrive lo scopo. Tale commento non deve tuttavia includere informazioni dettagliate sulla modalità di implementazione. Tali informazioni, essendosoggette a modifiche, comporterebbero infatti un inutile lavoro di manutenzione o addirittura la presenza di commenti non corretti. La modalità di implementazione viene comunque descritta nel codice stesso e nei commenti inseriti sulla stessa riga delleistruzioni.

È necessario descrivere gli argomenti passati a routine la cui funzione non risulta evidente o che sono inclusi in un determinato intervallo. All'inizio delle routine è inoltre necessario includere una descrizione dei valori restituiti da funzioni e altre variabili che vengono modificate dalla routine, soprattutto se le modifiche vengonoapportate tramite argomenti di riferimento.

Nei commenti di intestazione delle routine è necessario includere le intestazioni di sezione indicate di seguito. Per alcuni esempi, vedere la sezione successiva "Formattazione del codice".

Intestazione di sezione	Commento
`Scopo`	Operazioni eseguite dalla routine, non la modalità di esecuzione.
`Presupposti`	Elenco di variabili esterne, controlli e altri elementi il cuistato ha effetto sulla routine.
`Effetti`	Elenco dell'effetto della routine su variabili esterne, controllie altri elementi.
`Input`	Descrizione degli argomenti con significato non evidente. Ciascun argomento deve essere incluso in una riga distintainsieme a un commento specifico sulla stessa riga.
`Valori restituiti`	Descrizione del valore restituito.

È importante tenere presente i seguenti punti:

le dichiarazioni di ogni variabile importante devono includere un commento sulla stessa riga in cui viene descritto l'utilizzo della variabile dichiarata;
i nomi di variabili, controlli e routine devono essere sufficientementedescrittivi in modo che sia necessario aggiungere commenti sulla stessa riga solo per la descrizione di implementazioni complesse;
all'inizio degli script è consigliabile includere cenni preliminari che descrivano brevemente lo script e in cui siano elencati gli oggetti, le routine, gli algoritmi, le finestre di dialogo e altre dipendenze di sistema. A volte può risultare utile aggiungere pseudocodice che descrivi l'algoritmo.

Formattazione del codice

Lo scopo è quello di risparmiare spazio sullo schermo, ma allo stesso tempo applicare al codice una formattazione che ne rifletta in modo chiaro la struttura logica e i livelli di nidificazione. Qui di seguito trovate alcuni consigli che possono esservi utili:

- i blocchi nidificati standard devono rientrare di quattro spazi.
- i commenti introduttivi di una procedura devono essere rientrati di uno spazio.
- le istruzioni di livello principale che seguono i commenti iniziali devono rientrare di quattro spazi. Ciascun blocco nidificato dovrebbe rientrare di altri quattro spazi.

Esempio:

Il seguente codice corrisponde alle convenzioni del VB Script coding.

- Scopo: cerca la prima comparsa dell´utente indicato nel campo dati UserList.
- Inputs: strUserList(): lista di utenti che deve essere esaminata.
- strTargetUser: il nome dell´utente di cui si è alla ricerca.
- Valori di risposta: Indice della prima apparizione di strTargetUser nel campo dati strUserList. se l´utente cercato non viene trovato: -1.

```
Function intSucheBenutzer (strBenutzerListe(), strZielBenutzer)
    Dim i                           ' Loop counter.
    Dim blnFound                 ' 'Target found' flag.
    intFindUser = -1
    i = 0                           ' Initialize loop counter
    Do While i <= Ubound(strUserList) and Not blnFound
        If strUserList(i) = strTargetUser Then
            blnFound = True       ' Set flag to True
            intFindUser= i    ' Set return value to loop count
        End If
        i = i + 1              ' Increment loop counter
    Loop
End Function
```

4. Lista Macro

Con VBA si può ampliare autonomamente lo spettro delle funzionalità di zenon. Viene descritto l´uso di macro con zenon.

MENU

Punto di menù	Azione
VBA-Visualizza Editor	Apre l'editor VBA.
Esporta tutti VBE...	Apre il dialogo che serve a selezionare il posto in cui salvare l´esportazione VBE.
Importa VBE	Apre il dialogo che serve alla selezione del file import VBE.
Profili editor	Apre la lista a cascata con i Profili editor predefiniti.
Guida	Attiva la guida online

Informazioni su

Se si modificano delle macro VBA nell´Editor:

- vengono compilati i file Runtime e trasmessi nel Runtime.
- il runtime viene riavviato.
- anche gli elementi VSTA vengono ricaricati, anche quando non c´è stata alcuna modifica in VSTA .

VBA avvia lo stesso ambiente di sviluppo per workspace e progetto. Per aprire l´editor VBA:

1. navigate per raggiungere il nodo**interfacce di programmazione**
2. cliccate poi con il tasto destro del mouse su Macro VBA.
3. selezionate nel menù contestuale **Apri VBA-Editor...**

Alternativa: premete la combinazione tasti Alt+F11

4.1 VBA: Barra simboli e Menu contestuale Visualizzazione dettagliata

BARRA SIMBOLI

Punto di menù	Azione
Nuova macro VBA	Crea una nuova macro e apre l´editor per le macro.
VBA-Visualizza Editor	Apre l'editor VBA.
Salva	Salva macro.
Rimuovi	Cancella l´elemento selezionato.
Esporta tutti VBE...	Apre il dialogo che serve a selezionare il posto in cui salvare l´esportazione VBE.
Importa VBE	Apre il dialogo che serve alla selezione del file import VBE.
Rinomina	Permette di rinominare le macro selezionate.
Guida	Attiva la guida online

MENÙ CONTESTUALE MODULI MACRO

Punto di menù	Azione
VBA-Visualizza Editor	Apre l'editor VBA.
Salva	Salva macro.
Esporta tutti VBE...	Apre il dialogo che serve a selezionare il posto in cui salvare l´esportazione VBE.
Importa VBE	Apre il dialogo che serve alla selezione del file import VBE.
Guida	Attiva la guida online

MENU CONTESTUALE MODULI

Punto di menù	Azione
Nuova macro VBA	Crea una nuova macro e apre l´editor VBA.
Guida	Attiva la guida online

MENU CONTESTUALE MACRO

Punto di menù	Azione
Modifica	Apre macro nell´Editor per potere effettuare delle modifiche.

	Alternativa: tasto `INVIO` oppure doppio clic.
Rimuovi	Cancella macro Alternativa: tasto `Canc`.
Rinomina	Apre campo nome per effettuare delle modifiche. Alternativa: tasto `F2`.
Guida	Attiva la guida online

BARRA DEI SIMBOLI EDITOR

Le macro che sono state realizzate usando VBA, possono essere gestite tramite la barra dei simboli **Lista macro**.

Simbolo	Funzione
(da sinistra a destra)	
Ricarica la lista delle macro VBA./VSTA	Carica tutte le macro**Public Sub Name ()** contenute in **myWorkspace** e nei moduli nella lista dropdown della barra simboli.
Cerca macro	Cerca macro tramite il campo di inserimento della combobox, oppure selezione dalla lista a cascata. La lista a cascata viene adattata all´elemento più ampio quando viene aperta.
Macro della lista a cascata	Contiene tutte le macro caricate e ne permette la selezione.
Esegui le macro selezionate.	Esegue la macro selezionata nella lista a cascata (dropdown).
Esegui la macro #<x> assegnata.	Esegue la macro assegnata al simbolo.
Assegna macro	Apre il dialogo che serve all´assegnazione di macro. Con i simboli possono essere collegate da 1 a 5 macro.
VBA	Filtra alla ricerca di macro VBA. Vengono visualizzate solamente macro VBA.
VSTA	Filtra alla ricerca di macro VSTA. Vengono visualizzate solamente macro VSTA.
ALL	Cancella il filtro attualmente in uso e vengono visualizzate tutte le macro.
AZ	Mette le macro in ordine ascendente, da 0 a 9 e dalla A alla Z.
ZA	Mette le macro in ordine discendente, dalla Z alla A e da 9 a 0.
Opzioni barra strumenti	Cliccando sulla freccia si apre il sotto-menu: Attivo: viene visualizzata la barra dei simboli. Se non appare la barra dei simboli, la si può rendere attiva facendo uso del menù **Opzioni -> Barre dei simboli**. **Nota:** per una barra degli strumenti posizionata liberamente (sganciata dall´Editor), le opzioni non vengono visualizzate. . La barra degli strumenti può essere chiusa cliccando sul pulsante X.

Informazioni su

Nel caso in cui nel dialogo di attribuzione non vengano offerte tutte la macro contenute in **myWorkspace**, eseguite nella barra dei simboli la funzione**Ricarica la lista delle macro VBA**.

4.2 VBA su sistemi a 64-Bit

A partire dalla sua versione 7.10, zenon supporta sistemi operativi a 64-Bit. Per quello che concerne VBA, si è passati alla versione VBA 7.1. Per questo motivo, VBA è disponibile anche in zenon 64-Bit. Se nel VBA code si accede a Windows API o ad altre funzioni DLL importate, queste aperture devono essere adattate a 64-bit. In generale vale quanto segue: un file VBA creato con una versione a 32-bit non può essere usato senza modifiche-a una versione a 64-bit di VBA.

Per scrivere un codice compatibile a 32 ed a 64 Bit, ci sono in VBA alcune funzioni define. Ad esempio:

```
#if Win64 then
   Declare PtrSafe Function MyMathFunc Lib "User32" (ByVal N As LongLong) As LongLong
#else
   Declare Function MyMathFunc Lib "User32" (ByVal N As Long) As Long
#end if
#if VBA7 then
   Declare PtrSafe Sub MessageBeep Lib "User32" (ByVal N AS Long)
#else
   Declare Sub MessageBeep Lib "User32" (ByVal N AS Long)
#end if
```

Potete ottenre alcune indicazioni utili sul porting di VBA 32-bit code a VBA 64-bit da Microsoft:

- Microsoft Office 2010, notes on porting:
  http://msdn.microsoft.com/en-us/library/ee691831.aspx
  (http://msdn.microsoft.com/en-us/library/ee691831.aspx)
- 32-Bit e 64-Bit Declares per API-Calls: http://www.jkp-ads.com/articles/apideclarations.asp
  (http://www.jkp-ads.com/articles/apideclarations.as)

4.3 Basi:

Descrive le basi del linguaggio di programmazione VBA - Visual Basic for Applications

4.3.1 Proprietà degli oggetti - PROPERTIES

Una proprietà degli oggetti (property) è un determinato attributo dell'oggetto. Nel caso di un oggetto-variabili, ciò può essere, per esempio, il valore, il nome o l'identificazione. Nel caso di un oggetto-cerchio, può essere, per esempio, la posizione o il colore del cerchio nell'immagine. Ogni oggetto possiede perlomeno una property (generalmente sono, però, più di una); ogni property ha un valore definito. Mentre il **`Property Name`** è un testo, **`Property Value`** ha un valore compreso fra 0 e, per es., 1000.

La particolarità delle properties è che esse, grazie alla modifica dei valori delle proprietà in un programma VBA, possono modificare il comportamento e l'aspetto dell'oggetto. Se ad esempio attribuite alla **`Property Value`** (valore) dell'oggetto variabili un altro valore, ottenete l'effetto che la variabile attualmente selezionata riceva il valore da Voi desiderato. Non potete però attribuire ad ogni property un nuovo valore. La **`Property Count`** (quantità) dell'oggetto variabili, ad esempio, è fissa, visto che la quantità delle variabili viene definita mediante le variabili create. Dando al count un valore superiore non aggiungete alcuna nuova variabile. Alcune properties sono, perciò, solo proprietà di lettura, il che significa che ne si può soltanto leggere il valore.

4.3.2 Metodi di oggetti - METHODS

Oltre alle properties, ogni oggetto può possedere dei metodi. Un metodo non è un attributo, ma una richiesta all'ggetto di fare qualcosa. Così una forma ha il Methode `Show`. Quale effetto ha questo metodo? Richiede alla forma di rappresentare se stessa sullo schermo. Allo stesso modo, la foma scompare con l'uso del comando `Unload`.

Il vantaggio di un metodo è quello che il programmatore non ha deve sapere nulla sulla struttura dell'oggetto e, soprattutto, non ha la possibilità di mettere le mani sui dati interni dell'oggetto stesso.

L'apertura del metodo Show oppure del comando unload avviene nel modo seguente:

`frmSollwert.Show` oppure `Unload frmSollwert`

Se volete, per esempio, aprire un'altra forma, il metodo rimane lo stesso; a cambiare è solo il nome della forma (nome oggetto).

`frmChange.Show` oppure `Unload frmChange`

Lo stesso metodo può essere, dunque, usato per diversi tipi di oggetto. Non tutti gli oggetti, però, devono possedere necessariamente dei metodi.

4.3.3 Eventi di oggetti - EVENTS

Anche se nel 90 per cento dei casi avete a che fare con properties e metodi quando lavorate con oggetti, c'è ancora una categoria di caratteristiche di cui può disporre un oggetto: Events. Alcuni oggetti della gerarchia di oggetti possono reagire ad eventi. Events si verificano da soli mentre si lavora con zenon.

Esempio

ogni volta che si apre un'immagine, si causa un open-event nell'oggetto "picture" corrispondente. Da programmatori, potete fissare nell'ambito della cosiddetta procedeura dell'evento (la procedure che viene richiamata al verificarsi dell'evento stesso) delle indicazioni volte a definire quello che deve accadere in un caso simile. Un esempio di ciò è la modifica di una variabile. Si potrebbe produrre un event che reagisce alla variazione di valore di una variabile.

4.3.4 VBA - modello oggetti in zenon

In linea di massima, nella struttura di progetto ci sono sempre una lista oggetti e oggetti.

Esempio:

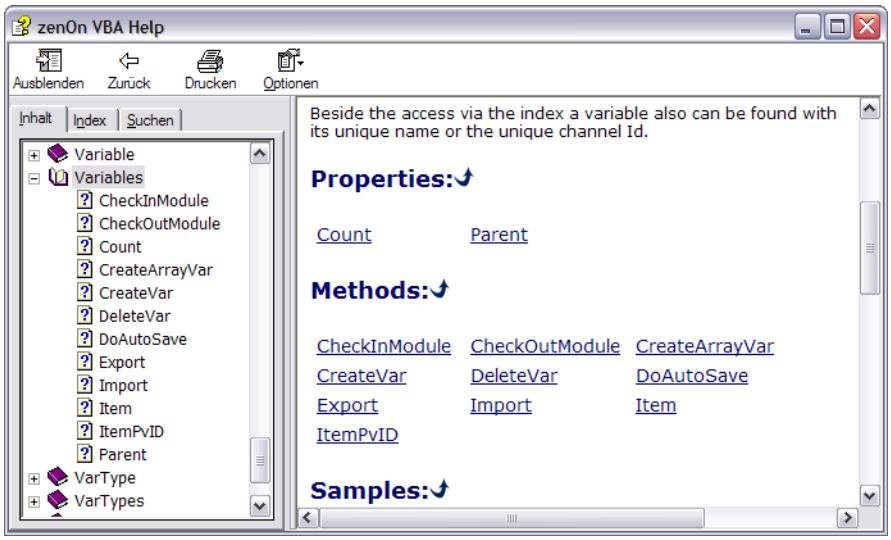
Projects – Project

Variables – Variable

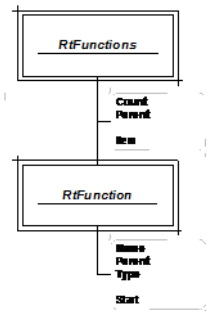
Elements – Elementi

Ulteriori informazioni sui modelli oggetto le trovate:

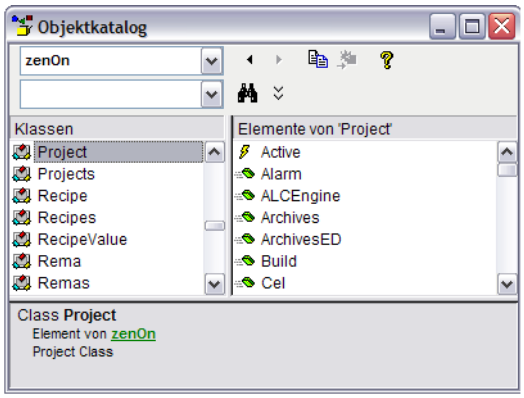
- nella guida in linea VBA



- nella visione generale grafica che ricevete in COPA-DATA

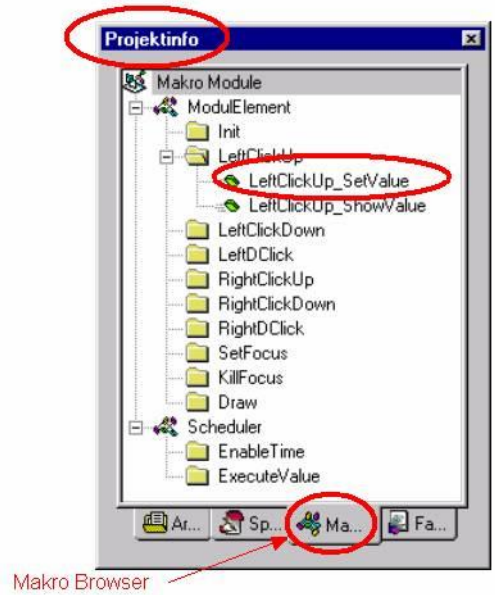


- nell´Object-Browser di VBA



4.3.5 Come si utilizzano macro VBA?

Per creare una nuova macro, si deve selezionare nella finestra **`Informazione progetto`** alla voce **`Makro Browser`** un evento desiderato al verificarsi del quale deve essere eseguita la nuova macro.



Cliccando con il tasto destro del mouse su questo evento, si apre un menu.

Selezionate il punto menu “Nuova macro” In tal modo viene generata una procedura da zenon:

```
Public Sub LeftClickUp_Sollwert(obElem As Element)
End Sub
```

Se esiste già una macro, questa può essere modificata, eliminata o ne se può cambiare il nome facendo clic con il tasto destro del mouse.

> ⚠ **Attenzione**
>
> *Nel caso in cui selezioniate il punto menu **`rinomina macro umbenennen`**, fate attenzione a non modificare il nome dell'evento, per es. LeftClickUp_..., del nome attuale. Altrimenti non viene eseguita la modifica del nome. Inoltre, quando modificate il nome di una macro, dovete modificare anche quello del sottoprogramma che deve essere eseguito by end nell'editor VBA.*

Dopo che si è inserito il codice sorgente da eseguire nella procedura generata da zenon, si deve collegare la macro create con un elemento.

Cliccando due volte su un elemento desiderato, appaiono sullo schermo le proprietà dell'elemento.

Alla voce "eventi" si può collegare la macro con l'elemento.

Cliccando con il tasto sinistro del mouse, viene provocato il verificarsi dell'evento LeftClickDown dell'elemento e viene eseguita la macro collegata.

Integrazione di macro già esistenti

Per integrare delle macro già esistenti in un altro progetto, procedete nel modo seguente:

1. Esportate tutti i formulari e moduli necessari nell'editor VBA e importateli nel progetto desiderato.

2. Le macro orientato ad eventi, nel `ModulElement.bas`, non vengono ancora mostrate nel browser macro. Perciò dovete creare nuovamente le macro nel browser macro.

   Il modo più semplice di far ciò è usare il nome della macro già esistente.

   per Esempio:
   LeftClickUp_ `DateSet2`
   LeftClickUp_ `DateSet4`
   LeftClickUp_ `TimeSet`
   Draw_ `Date2`
   Draw_ `Date4`
   Draw_ `Time`

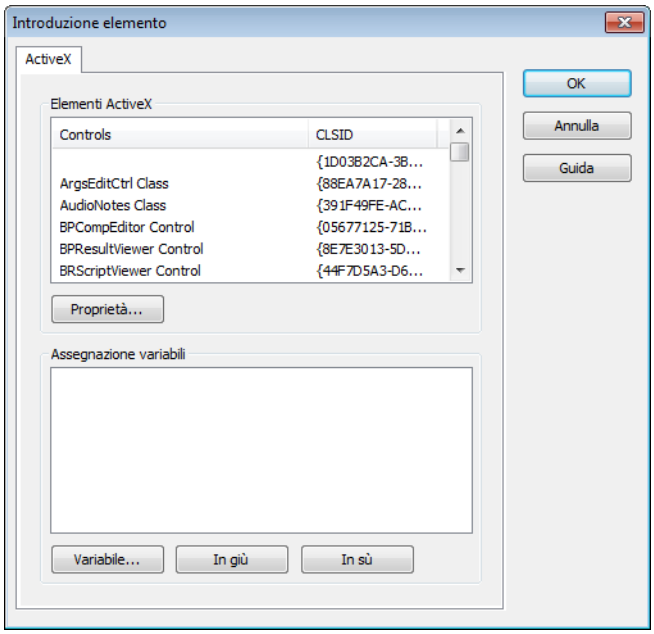
3. Al momento di creare delle macro, zenon genera procedure che hanno lo stesso nome delle macro esistenti. Queste procedure devono essere eliminate.

4. Collegate queste macro, come al solito, con un elemento dinamico.

> **Informazioni su**
>
> Se l'hardware non è collegato e utilizzatate il simulator-mode del driver SAIA, non dimenticare di fermare il driver.

4.3.6 Come si collega un elemento ein ActiveX in zenon?

Un elemento ActiveX viene portato in un'immagine come un elemento dinamico; si apre una finestra nella quale dovete scegliere un elemento ActiveX.



- Una volta scelto l'elemento nella lista rappresentata qui sopra, potete attribuirgli una variabile. Per far ciò, cliccate sul pulsante **variabile...** e selezionatene una, oppure createne una nuova.
- Date al Vostro elemento ActiveX un nome oggetto, attraverso il quale potrete poi accedervi in VBA.
- Nel nostro esempio usiamo il nome oggetto Slider6_DW18, visti che si tratta di un elemento ActiveX-Element Slider che è collegato con la variabile Datenwort 18.
- Adesso l'elemento slider deve essere attivato ed editato nell'editor VBA.
- A tal scopo, creiamo una nuova macro come descritto nel capitolo "Come si usano macro VBA? (A pagina: 54)".

La macro Init_slider trasmette l'elemento da inizializzare ad un sottoprogramma nell'oggetto thisProject; così viene definito un riferimento al progetto attuale.

```
Public Sub Init_Slider(obElem As Element)
thisProject . Init _ Slider obElem
End Sub
```

Come nella macro Init_Slider, anche in Draw_SliderValue l'elemento viene passato all'oggetto thisProject.

```
Public Sub Draw_SliderValue (obElem As Element, ByVal hdc As OLE _ HANDLE )
thisProject.Draw_Slider obElem
obElem.Draw hdc
End Sub
```

Nell'oggetto zenOn this Project viene inserito il codice sorgente indicato di seguito.

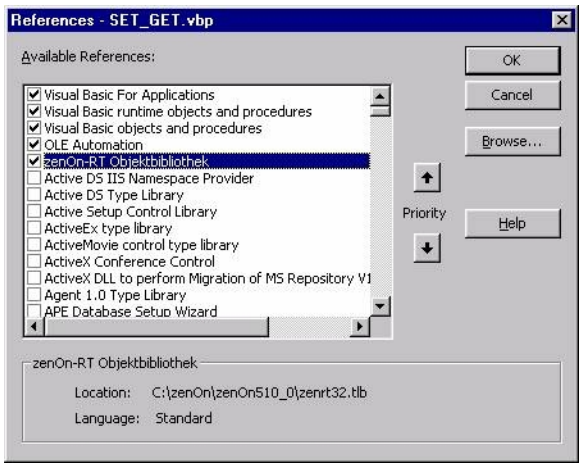
```
Public Declarations
Public WithEvents obSlider As Slider
Public obSliderPV As Variable
Public Sub Init_Slider (obElem As Element)
Set obSlider = obElem.ActiveX
'ActiveX exists
If obSchieberegler Is Nothing Then
Exit Sub
End If
Set obSliderPV = obElem . ItemVariable(0)
'variable exists
If obSliderPV Is Nothing Then
Exit Sub
End If
obSlider.Max = obSliderPV.RangeMax
obSlider.Min = obSliderPV.RangeMin
obSlider.TickFrequency = 1000
obSlider.LargeChange = 25
obSlider.SmallChange = 1
obSlider.Value = obSliderPV.Value
End Sub

Public Sub Draw _ Schieberegler ( obElem As Element )
Dim vVar As Variant
Dim obDynPic As DynPicture

Set obSliderPV = obElem.ItemVariable ( 0 )
Set obDynPic = thisProject.DynPictures. Item (BILD_1)
'variable exists
If obSliderPV Is Nothing Then
Exit Sub
End If
```

4.3.7 Accesso mediante un programma esterno

Per accedere mediante un programma esterno (ad esempio Visual Basic) a dati di zenon, si usa l'interfaccia COM. Questa interfaccia COM è la stessa ad essere usata anche da VBA. Ci sono dunque solo un paio di piccole differenze, di cui però si deve tenere conto.

Visual Basic 6

Per poter accedere all'interfaccia COM, quest'ultima deve essere implementata:



Mediante questa libreria tipi, potete accedere all'oggetto applicazione di zenon (il runtime).

Visto che qui non c'è un oggetto thisproject, lo si deve creare poter giungere ai dati.

```
Dim obProject As zenon.Project
Set obProject = zenon.Application.Projects.Item(PROJEKTNAME)
```

Se il progetto VB deve funzionare con tutti i progetti di zenon (deve essere, per così dire, indipendente da determinato nome progetto) può essere impostato anche in questo modo.

```
Set obProject = zenon.Application.Projects.Item(0)
```

Dopo che è stato creato l'oggetto progetto (thisProject), si può accedere, ad esempio, alle variabili ai fini di lettura e scrittura.

Leggere:

```
Wert = obProject.Variables.Item(Variablenname).Value
```

Scrivere:

```
obProject.Variables.Item(Variablenname).Value = valore
```

4.3.8 Funzionalità di una variabile online

Ci si può immaginare una variabile online VBA come un container che contiene delle variabili; quest'ultime devono essere naturalmente inserite nel container. Nel caso in cui ci sia una variazione del valore di una variabile contenuta nal container, ciò viene reso noto mediante un evento.

Funzionalità dell'evento

Quando viene attivato il container (`Container.Define`), tutte le variabili che vi si trovano vengono stimolate una volta per conoscere il valore loro valore attuale Viene eseguita la procedura „`Container_VariableChange`“ per ogni variabile che si trova nel container. Quando poi tutte le variabili sono state inizializzate, l'evento si verifica ogni volta che il valore di una variabile del container cambia.

In tal modo è escluso che venga letto un valore che non corrisponde al valore attuale della variabile.

Definire e impostare Container

Definizione:

```
Public WithEvents Container As OnlineVariable
```

Mediante questa riga di codice si definisce il container.

Creare

```
Set Container = thisProject . OnlineVariables. CreateOnlineVariables( Containername)
```

Mettere variabili in container

```
Container. Add Variablenname1
Container . Add Variablenname2
Container . Add Variablenname3
Container . Add Variablenname4
...
```

Questo procedimento deve essere ripetuto finché tutte le variabili desiderate non si trovano nel container.

Imposta evento

```
Private Sub Container_VariableChange(ByVal obVar As zenon.IVariable)
...
End Sub
```

Questo evento viene impostato automaticamente quando nel combobox di sinistra al margine superiore dell'editor VBA viene selezionato il container. Poi la procedura sopra illustrata viene inserita nel sourcecode. Grazie a „**obVar**“ viene data la variabile il cui valore è cambiato. Al verificarsi dell'evento, poi, si può leggere per es. il valore attuale (**ob-Var.Value**). Quali proprietà e valori delle variabili possono essere usati, lo si può vedere nella gerarchia degli oggetti nella documentazione VBA.

Attiva evento

```
Container.Define
```

Questa riga di comando attiva, per così dire, il controllo delle variabili nel container. Solo quando il comando "**Define**“ viene eseguito, il container è attivo.

Disattivazione dell'evento

```
Container.Undefine
```

Con questo comando, viene chiuso il monitoraggio in container; a partire da questo momento, l´evento (VariableChange) non viene più eseguito.

Elimina al momento di terminare

Per non lasciare veramente nulla nella memoria al momento di chiudere il runtime, il container deve essere eliminato al più tardi al momento di terminare il runtime.

```
thisProject. OnlineVariables. DeleteOnlineVariables ( Containername )
```

Solo dopo che questo container è stato cancellato, ne può essere impostato un altro con lo stesso nome.

4.3.9 Lista bit di stato

Numero Bit	Denominazione breve	Denominazione lunga	zenon Logic indicatore
0	M1	Stato utente 1	_VSB_ST_M1
1	M2	Stato utente 2	_VSB_ST_M2
2	M3	Stato utente 3	_VSB_ST_M3
3	M4	Stato utente 4	_VSB_ST_M4
4	M5	Stato utente 5	_VSB_ST_M5
5	M6	Stato utente 6	_VSB_ST_M6
6	M7	Stato utente 7	_VSB_ST_M7
7	M8	Stato utente 8	_VSB_ST_M8
8	NET_SEL	Seleziona nella rete	_VSB_SELEC
9	REVISION	Revisione	_VSB_REV
10	PROGRESS	Direzione	_VSB_DIREC
11	TIMEOUT	Runtime exceeded	_VSB_RTE
12	MAN_VAL	Manual value	_VSB_MVALUE
13	M14	Stato utente 14	_VSB_ST_14
14	M15	Stato utente 15	_VSB_ST_15
15	M16	Stato utente 16	_VSB_ST_16
16	GI	GI; General Interrogation	_VSB_GR
17	SPONT	Spontaneo	_VSB_SPONT
18	INVALID	Non valido	_VSB_I_BIT
19	T_CHG_A	Annuncio ora solare/ora legale	_VSB_SUWI
20	OFF	Spento	_VSB_N_UPD
21	T_EXTERN	External Real-Time	_VSB_RT_E
22	T_INTERN	Internal Real-Time	_VSB_RT_I
23	N_SORTAB	Non filtrati	_VSB_NSORT

24	FM_TR	Transformer	_VSB_DM_TR
25	RM_TR	Transformer anomaly	_VSB_RM_TR
26	INFO	Info della variabile	_VSB_INFO
27	ALT_VAL	Valore di riserva Se non è stato trasmesso ancora nessun valore, viene usato il valore di sostituzione definito; in caso contrario l´ultimo valore valido.	_VSB_AVALUE
28	RES28	Riservato per uso interno (lampeggio allarmi)	_VSB_RES28
29	N_UPDATE	Not updated	_VSB_ACTUAL
30	T_STD	Ora solare	_VSB_WINTER
31	RES31	Riservato per uso interno (lampeggio allarmi)	_VSB_RES31
32	COT0	Cause of Trasmissions Bit 1	_VSB_TCB0
33	COT1	Cause of Trasmissions Bit 2	_VSB_TCB1
34	COT2	Cause of Trasmissions Bit 3	_VSB_TCB2
35	COT3	Cause of Trasmissions Bit 4	_VSB_TCB3
36	COT4	Cause of Trasmissions Bit 5	_VSB_TCB4
37	COT5	Cause of Trasmissions Bit 6	_VSB_TCB5
38	N_CONF	Risposta negativa di SELECT dal dispositivo (IEC 60870 [P/N])	_VSB_PN_BIT
39	TEST	Test-Bit (IEC 60870 [T])	_VSB_T_BIT
40	WR_ACK	Conferma scrittura	_VSB_WR_ACK
41	WR_SUC	Scrittura avvenuta con successo	_VSB_WR_SUC
42	NORM	Stato normale	_VSB_NORM
43	N_NORM	Normal deviation	_VSB_ABNORM
44	BL_870	IEC 60870 Status: blocked	_VSB_BL_BIT
45	SB_870	IEC 60870 Status: substituted	_VSB_SP_BIT

46	NT_870	IEC 60870 Status: not topical	_VSB_NT_BIT
47	OV_870	IEC 60870 Status: overflow	_VSB_OV_BIT
48	SE_870	IEC 60870 Status: select	_VSB_SE_BIT
49	T_INVAL	Timestamp non valido	non definito
50	CB_TRIP	Intervento sezionatore rilevato	non definito
51	CB_TR_I	Rilevamento di intervento sezionatore disabilitato	non definito
52	RES52	riservato	non definito
53	RES53	riservato	non definito
54	RES54	riservato	non definito
55	RES55	riservato	non definito
56	RES56	riservato	non definito
57	RES57	riservato	non definito
58	RES58	riservato	non definito
59	RES59	riservato	non definito
60	RES60	riservato	non definito
61	RES61	riservato	non definito
62	RES62	riservato	non definito
63	RES63	riservato	non definito

Informazioni su

In formule sono disponibili tutti i bit di stato. Per altri usi la disponibilità può essere ridotta.

Dettagli per la modifica degli stati li potete trovare nel capitolo Modifica stati.

4.3.10 Lasso per selezionare elementi dinamici nel Runtime.

Gli elementi dinamici che sono collegati a una variabile oppure a una funzione, possono essere preselezionati nel Runtime con il lasso e, in tal modo, utilizzati, per eventi VBA.

Grazie al metodo **`SelElements`**, il sistema è in grado di riconoscere gli elementi dinamici scelti dall´utente nel Runtime come selezionati. Questi **`DynPicture.SelElements`** possono essere poi usati per eventi come Drag&Drop.

SELEZIONE MEDIANTE LASSO

Per selezionare nel Runtime degli elementi con il lasso, deve verificarsi quanto segue:

- deve essere attiva nelle proprietà di progetto, la proprietà `Impostazioni Runtime/Selezioni multiple in Runtime (Lazzo)`
- nelle proprietà dell´elemento dinamico deve essere attiva la proprietà `Runtime/selezionabile tramite lazzo`

Durante il Runtime ci sono diversi metodi a disposizione dell´utente per selezionare degli elementi:

- Seleziona elementi: cliccare con il tasto sinistro del mouse in una parte vuota dell´immagine e trascinare il lasso sugli elementi ad immagine tenendo premuto il tasto del mouse.
- Estendi selezione: `Ctrl+clic del mouse` su un elemento per selezionarlo o deselezionarlo.
- Aggiungi elementi: Aprendo il lasso, tenere premuto il tasto `Ctrl`: in tal modo si aggiungono nuovi elementi a quelli già selezionati.
- Cancella selezione: aprire un lasso, che non contiene alcun elemento.

4.4 Macro nell´editor

Le macro possono essere eseguite nell´editor per mezzo di una barra degli strumenti (A pagina: 65) configurabile. A tal scopo, delle macro vengono collegate a pulsanti nella barra degli strumenti **`VBA`** .

Inoltre le macro possono essere eseguite anche manualmente, oppure tramite l´editor VBA.

Facendo uso di wizard, potete far in modo che operazioni ricorrenti vengano eseguite automaticamente, oppure che degli interi progetti vengano realizzati semplicemente premendo un pulsante. Alcuni wizard

vengono forniti a titolo di esempio con zenon; essi possono essere ampliati e integrati a piacere: Essi aiutano a creare progetti, ad effettuare l´import e l´export, a creare delle variabili e a svolgere molti altri compiti. I dettagli relativi li potete trovare nel capitolo Wizards

EDITOR-EVENTS

Gli Eventi Editor sono una componente del workspace VBA e consentono di reagire ad eventi nella programmazione workspace,ad es. per Wizards oppure Trasporto Remoto. Ad esempio:

- OnElementCreated
- OnElementDeleted
- OnElementDoubleClicked
- OnObjectCreated
- ...

Tutti gli eventi e informazioni relative li trovate nella guida in linea concernente Object Model.

4.4.1 Barra degli strumenti "Lista macro"

Le macro che sono state realizzate usando VBA, possono essere gestite tramite la barra dei simboli **Lista macro**.

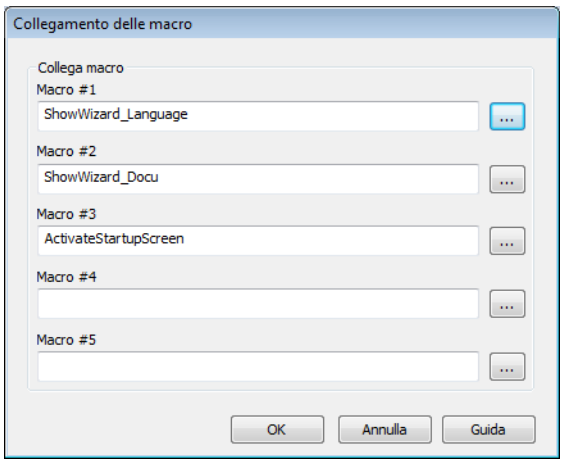
Simbolo	Funzione
(da sinistra a destra)	
Ricarica la lista delle macro VBA./VSTA	Carica tutte le macro**Public Sub Name ()** contenute in **myWorkspace** e nei moduli nella lista dropdown della barra simboli.
Cerca macro	Cerca macro tramite il campo di inserimento della combobox, oppure selezione dalla lista a cascata. La lista a cascata viene adattata all´elemento più ampio quando viene aperta.
Macro della lista a cascata	Contiene tutte le macro caricate e ne permette la selezione.
Esegui le macro selezionate.	Esegue la macro selezionata nella lista a cascata (dropdown).
Esegui la macro #<x> assegnata.	Esegue la macro assegnata al simbolo.
Assegna macro	Apre il dialogo che serve all´assegnazione di macro. Con i simboli possono essere collegate da 1 a 5 macro.
VBA	Filtra alla ricerca di macro VBA. Vengono visualizzate solamente macro VBA.
VSTA	Filtra alla ricerca di macro VSTA. Vengono visualizzate solamente macro VSTA.
ALL	Cancella il filtro attualmente in uso e vengono visualizzate tutte le macro.
AZ	Mette le macro in ordine ascendente, da 0 a 9 e dalla A alla Z.
ZA	Mette le macro in ordine discendente, dalla Z alla A e da 9 a 0.
Opzioni barra strumenti	Cliccando sulla freccia si apre il sotto-menu: Attivo: viene visualizzata la barra dei simboli. Se non appare la barra dei simboli, la si può rendere attiva facendo uso del menù **Opzioni -> Barre dei simboli**. **Nota:** per una barra degli strumenti posizionata liberamente (sganciata dall´Editor), le opzioni non vengono visualizzate. . La barra degli strumenti può essere chiusa cliccando sul pulsante X.

Informazioni su

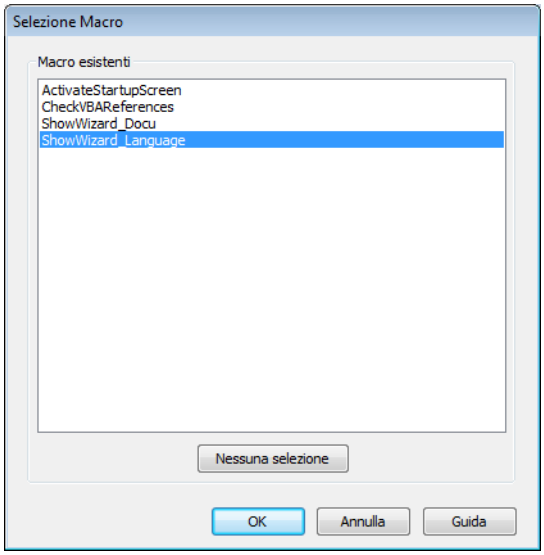
Nel caso in cui nel dialogo di attribuzione non vengano offerte tutte la macro contenute in **myWorkspace**, eseguite nella barra dei simboli la funzione**Ricarica la lista delle macro VBA**.

4.4.2 Collegamento delle macro

Le macro possono essere richiamate facendo uso di un pulsante nella barra dei simboli. Si possono collegare in questo modo al massimo cinque macro. Usando il pulsante **assegna** macro, si apre il dialogo che serve all´assegnazione delle macro.



Parametri	Descrizione
Makro #	Il numero macro corrisponde al numero del pulsante nella barra dei simboli. Cliccando sul pulsante . . . si apre il dialogo per selezionare le macro.
OK	Crea collegamenti con i pulsanti e chiude il dialogo.
Annulla	Annulla tutte le modifiche e chiude il dialogo.
Guida	Attiva la guida online



Parametri	Descrizione
Selezione esistente	Lista delle macro che possono essere collegate.
Nessuna selezione	Cancella l´assegnazione esistente per questo pulsante
OK	Assegna la macro selezionata al pulsante.
Annulla	Annulla tutte le modifiche e chiude il dialogo.
Guida	Attiva la guida online

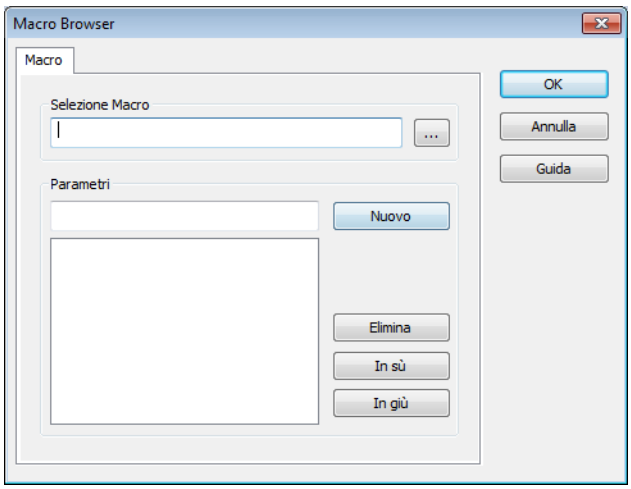
4.5 Funzioni in zenon

Nel dialogo **Scelta funzioni**, in corrispondenza dell´elemento **VBA** trovate le seguenti funzioni.

Funzione	Descrizione
Apri Editor PCE	Apre l'editor del modulo opzionale Process Control Engine (PCE).
VBA-Visualizza Editor	Apre l'editor VBA
Lanciare Macro-VBA	Esegue una macro VBA selezionata. **Attenzione:** L´evento VBA `project inactive` viene eseguito prima degli script `AUTO_END_xxx Skripte`. Per questo motivo, non viene più eseguita negli script la funzione di zenon `Esegui macro VBA`, dato che VBA non funziona più in questo momento.
VBA-Visualizza dialogo Macro	Apre il dialogo macro VBA

4.5.1 Lanciare Macro-VBA

Quando selezionate la funzione Esegui macro VBA, appare il seguente dialogo.



Sono disponibili le seguenti impostazioni:

Parametri	Descrizione
Selezione Macro	Apre il dialogo che serve a selezionare le macro (vediScelta macro (A pagina: 71)) Consiglio: Only lists VBA macros that match the number of parameters defined at the function Parameter (below).
Parametri	Inserite il valore desiderato (stringa) per un parametro.
Nuovo	Cliccate su questo pulsante per inserire il valore nella lista dei parametri disponibili.
Elimina	Cliccate su questo pulsante per cancellare l´inserimento selezionato dalla lista dei parametri disponibili. Potete cancellare solo un inserimento alla volta.
in sù	Cliccate su questo pulsante per spostare di una posizione verso l´alto l´inserimento selezionato. L´inserimento in questione verrà spostato di una posizione verso l´alto nella lista dei parametri.
In giù	Cliccate su questo pulsante per spostare di una posizione verso il basso l´inserimento selezionato. L´inserimento in questione verrà spostato di una posizione verso il basso nella lista dei parametri.

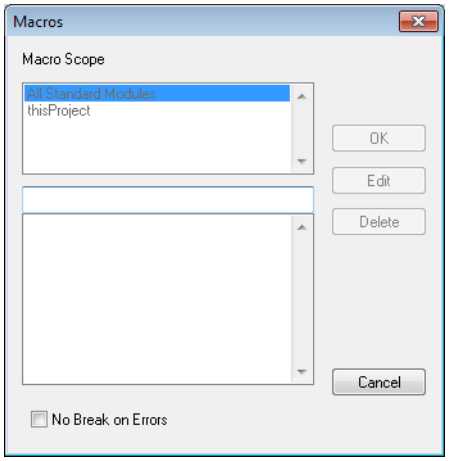
Avete la possibilità di depositare delle stringhe per macro che sono state create con parametri. Queste stringhe vengono trasmesse nel Runtime come singoli parametri quando viene eseguita la macro.

Informazioni su

Dovete sincerarvi che il numero dei parametri della macro collegata corrisponda a quello dei parametri impostati in questa sede.

Selezione Macro

Cliccando sul pulsante . . . appare il seguente dialogo.



Selezionate in questa sede la macro desiderata fra quelle disponibili e cliccate poi su OK.

4.6 Sviluppo di wizard in VBA

A partire dalla versione 6 è possibile automatizzare la fase di progettazione mediante l´uso di wizard. Le operazioni da ripetere spesso possono essere affidate ad un wizard, il quale provvede ad eseguirle: si può trattare, ad esempio, della creazione di un progetto, o della realizzazione di modelli ed immagini sulla base di uno standard definito.

Un´altra possibilità di utilizzo di wizard è l´effettuazione di modifiche automatiche in un progetto esistente, ad esempio, il cambiamento di proprietà degli elementi dinamici in tutte le immagini di uno dei progetti disponibili.

La base per la creazione di wizard sono Microsoft Visual Basic for Applications (VBA), nonché il modello oggetti di zenon.

Al momento sono disponibili i seguenti wizard:

- Wizard di progetto
- Wizard per l'importazione
- World view wizard
- Find VBA-Text Wizard

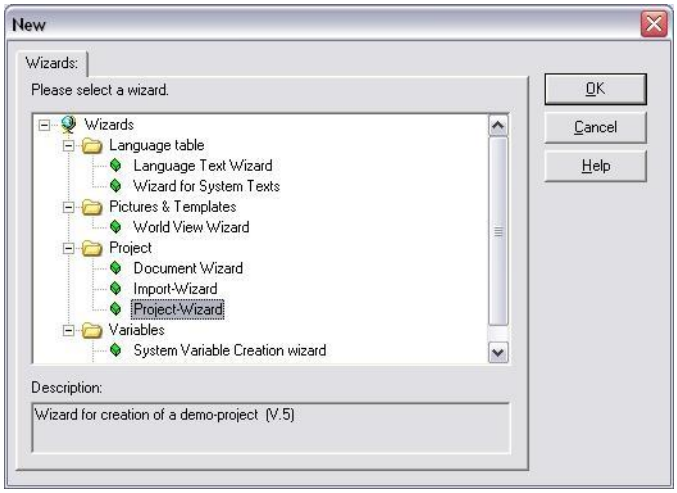
- Wizard for keyword-creation
- Wizard for keyword-creation
- Wizard for creating variables

I wizard sono disponibili nella forma di file codice sorgente VBA sul supporto di installazione. I nuovi wizard possono essere implementati tramite l´ambiente VBA.

4.6.1 Uso di un wizard

Il comando **`Wizards`** del menu **`File`** apre la selezione dei wizard. In questo dialogo vengono visualizzati tutti i wizard disponibili in base alla categoria di appartenenza.

Nel caso in cui i wizard non dovessero contenere nessuna categoria, viene creato automaticamente un inserimento `Non collegato`. Sotto questo inserimento vengono indicati in una lista tutti gli assistenti non collegati.



Si attiva un wizard selezionandolo e cliccando su **OK**.

Informazioni su

I wizard supportano solamente single user project e non suportano un ambiente multi-user.

4.6.2 Struttura di nuovi wizard

Un wizard è una UserForm, che viene depositata nel nodo specifico dell´applicazione. La UserForm consiste di norma di un elemento multipagina, nel quale vengono rappresentati i singoli passi del wizard.

Cliccando su un pulsante **`Avanti`**, si passa alla prossima pagina dell´elemento multipagina. Tutti gli inserimenti debbono essere impostati temporaneamente; la creazione di oggetti, per esempio, modelli, immagini, .. può essere effettuata solamente con il **`completa`** del wizard.

Informazioni su

Le UserForm che devono servire da wizard, devono contenere alcuni public method che forniscoo le informazioni sul wizard al sistema di controllo. Nel caso in cui queste routine manchino in una UserForm, quest`ultima non verrà considerata un wizard.

4.6.3 Integrazione in VBA

I wizard sono contenuti nel nodo specifico dell´applicazione `ZWorkspace` . Questo oggetto rappresenta il workspace caricato nell´editor ed è a disposizione solo nell´editor di zenon.

Tutti gli oggettii in questo progetto VBA possono accedere all´attuale workspace mediante l´oggetto `MyWorkspace`. Quest´ultimo è collegato sempre al progetto attualmente attivo, al quale potete accedere mediante le proprietà di `ActiveDocument`.

I contenuti dell´oggetto ZWorkspace sono contenuti nel file `ZenWorkspace.vba`. Questo file viene copiato con la prima installazione della versione 6 nel percorso di installazione. Nei successivi update, questo file non viene sovrascritto. Alla fine di questo tutorial riceverete ulteriori informazioni per realizzare l´attualizzazione dei wizard.

4.6.4 Sviluppo di un wizard

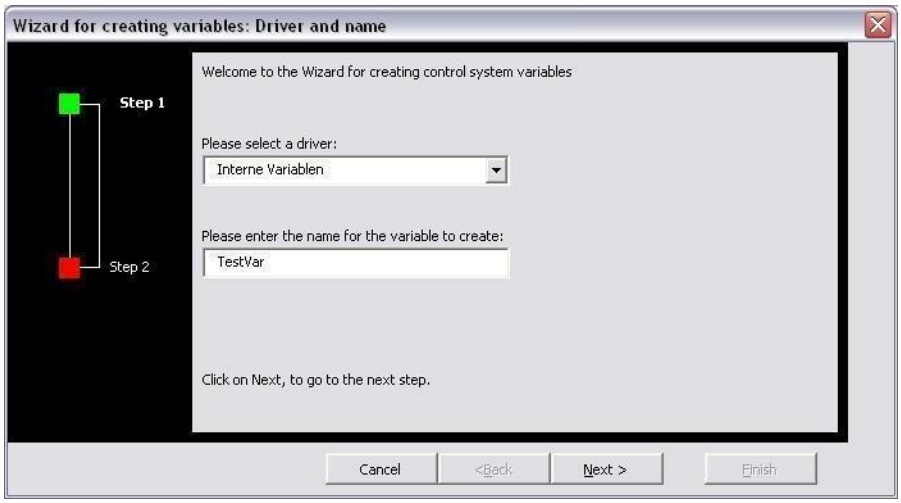
Questo tutorial crea un wizard per l'impostazione di variabili in un driver progettato.

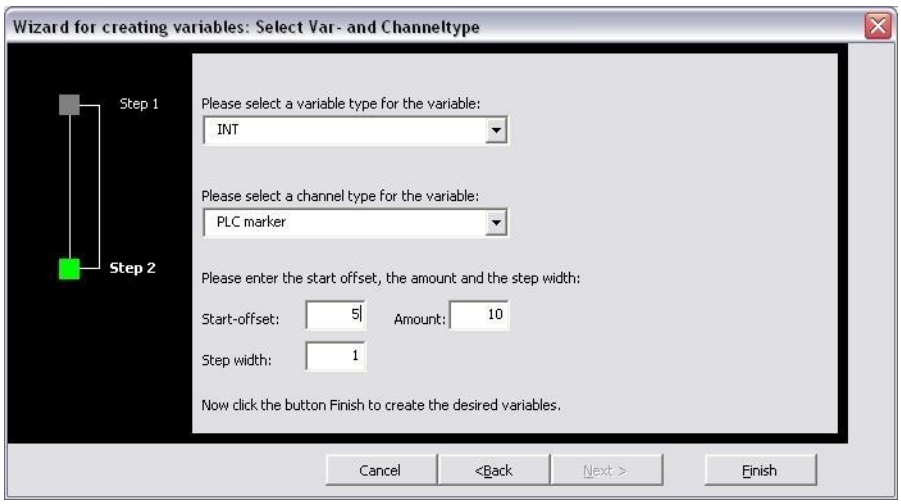
Aprite l´ambiente VBA dall´editor di zenon e passate alla cartella **`ZWorkspace/Formulare`**. Questo file contiene le basi per lo sviluppo di un wizard. Modificate il nome dello UserForm.

Nel caso in cui il wizard non dovesse essere disponibile nella cartella sopracitata, potete importarlo facendo uso del comando **`Importa file`**.

Informazioni su

Per sviluppare un wizard bisogna disporre di conoscenze relative al modello di oggetto di zenon e su VBA. In questo tutorial non si affrontano questi temi.





Create la superficie sopra rappresentata. Passate poi nel Code-Modul dello UserForm e scrollate fino alla fine del file. Lì troverete i seguenti metodi.

- `Public Function` GetWizardName () `As String`

  Fornisce il nome univoco del Wizard. Cambiatene il contenuto sostituendolo con `Wizard for creating variables`

- `Public Function` GetWizardInfo () `As String`

  Fornisce una breve descrizione che viene visualizzate nella scelta dei Wizard. Cambiatene il contenuto sostituendolo con `Wizard for creating variables to a selected driver`

- `Public Function` GetWizardCategory () `As String`

  Fornisce la categoria del Wizard. Nella selezione dei Wizard, essi vengono visualizzati a seconda della categoria in una struttura ad albero. Cambiatene il contenuto inserendo `Variables`.

- `Public Function` IsZenOnWizard () `As Boolean`

  Visualizzazione del Wizard nella selezione dei Wizard. Questo metodo fornisce "False", cosicché il Wizard non viene visualizzato, per es. perché non è ancora completato. Sostituite il Returntypen in `True`.

Questi metodi forniscono le informazioni sui Wizard che vengono richieste dal sistema di controllo. Bisogna tener presente che un Wizard viene visualizzato nella selezione Wizard solo nel caso in cui il metodo `IsZenOnWizard` fornisce `True`.

Passate a evento- `Initialize`della UserForm e cambiate i contenuti degli String-Arrays`m_strCaption`. Visto che il nostro Wizard consiste solo di due passi, potete cancellare le altre attribuzioni.

Inserite le seguenti definizioni nella parte alta del Code-Modul:

```
Private m_obDriver As Driver
Private m_obVarType As VarType
Private m_nChannelType As Integer
```

Create un metodo per l'inizializzazione del Combobox dei driver. Il compito della routine è rappresentare tutti i driver progettati del progetto attale nel ComboBox.

```
cbDriver.Clear
Dim nIndex As Long
For nIndex = 0 To MyWorkspace.ActiveDocument.Drivers.Count - 1
   Dim obDriver As Driver
   Set obDriver = MyWorkspace.ActiveDocument.Drivers.Item(nIndex)
   If (Not obDriver Is Nothing) Then
      cbDriver.AddItem
      obDriver.Name
   End If
Next nIndex
If (cbDriver.ListCount > 0) Then
   cbDriver.ListIndex = 0
End If
```

Avete poi bisogno di una Routine che visualizzi in un ComboBox tutti i tipi di variabili progettati del progetto.

```
If (Not m_obDriver Is Nothing) Then
cbVarType.Clear
Dim nIndex As Long , nSelect As Integer
For nIndex = 0 To MyWorkspace.ActiveDocument.VarTypes.Count - 1
Dim obVarType As VarType
Set obVarType = MyWorkspace.ActiveDocument.VarTypes.Item(nIndex)
If ( Not obVarType Is Nothing And obVarType.IsSimple = True) Then
cbVarType.AddItem
obVarType.Name
If (obVarType.Name = INT) Then
   nSelect = nIndex
End If
End If
Next nIndex
cbVarType.ListIndex = nSelect
End If
```

Al momento di aprire il Wizard verrano visualizzate le variabili già progettate per cercare un Start-Offset libero per le variabili che debbono essere create. Ciò accade grazie al seguente metodo.

```
Private Function FindHighestOffsetVar() As Long
On Error GoTo Error
Dim nIndex As Long , nOffset As Long
For nIndex = 0 To MyWorkspace.ActiveDocument.Variables.Count - 1
Dim obVar As Variable
   Set obVar = MyWorkspace.ActiveDocument.Variables.Item(nIndex)
   If ( Not obVar Is Nothing) Then
   If (obVar.Offset > nOffset) Then
         nOffset = obVar.Offset
      End If
   End If
Next nIndex
FindHighestOffsetVar = nOffset
Exit Function
Error : MsgBox
Error occurs: + Err.Description + Source + Err.Source
End Function
```

Passate in "Evento-Initialize" dello UserForm e ampliate questo metodo con le righe seguenti:

```
txtStart.Value = CStr(FindHighestOffsetVar + 1)
InitializeDriver
```

L'attribuzione a `txtStart` setta lo Start-Offset proposto per le variabili da creare. Mediante il metodo `InitializeDriver` viene riempito il ComboBox con i driver progettati.

Create un evento "change" per il Combobox dei driver e inserite il seguente codice. Dopo che è stato selezionato un driver, vengono individuati i tipi di variabili. L'oggetto driver selezionato viene scritto nella variabile `m_obDriver` per essere usato in seguito.

```
Private Sub cbDriver_Change()
cmdNext.Enabled = True
Set m_obDriver = MyWorkspace.ActiveDocument.Drivers.Item(cbDriver.Value)
If ( Not m_obDriver Is Nothing) Then
   InitializeVarType
End If
End Sub
```

Create un trattamento evento `Change` per VarType-Combobox e inserite il seguente codice. Il tipo di variabile selezionato viene scritto nella variabile m_obVarType per essere usato in seguito.

```
Private Sub cbVarType_Change()
Set m_obVarType = MyWorkspace.ActiveDocument.VarTypes.Item(cbVarType.Value)
End Sub
```

Adesso non Vi resta che creare la routine evento che deve servire alla creazione delle variabili con le impostazioni che sono state definite in precedenza. Ciò accade mediante il pulsante "Finish".

```
Private Sub cmdFinish_Click()
On Error GoTo Error
If (cbVarType.ListIndex = -1) Then
MsgBox 'Please select a variable type'
cbVarType.SetFocus
Exit Sub
End If
If (txtStart.Value = Or txtCount.Value = Or txtStep.Value = ) Then
MsgBox 'Please enter Start-Offset', 'count of creating variables and the step'
txtStart.SetFocus
End If
If (m_obVarType Is Nothing) Then
MsgBox 'Variable type + cbVarType.Name + doesnt exist!'
Exit Sub
End If
Dim nPrvMousePtr As Integer
nPrvMousePtr = MousePointer
MousePointer = fmMousePointerHourGlass
DoEvents
Dim strName As String
Dim nIndex As Long , nVarIndex As Integer
Dim nStartOff As Long , nStep As Integer
nVarIndex = 1
nStartOff = CLng (txtStart.Value)
nStep = CLng (txtStep.Value)
For nIndex = 0 To CLng (txtCount.Value - 1)
Dim obVar As Variable
strName = txtName.Value + _ + CStr (nIndex + 1)
'*** Guaranteeing uniqueness of the variable name
Dim bResult As Boolean
bResult = False
Do
Set obVar = MyWorkspace.ActiveDocument.Variables.Item(strName)
If (obVar Is Nothing) Then
bResult = True
Else
nVarIndex = nVarIndex + 1
strName = txtName.Value + _ + CStr (nVarIndex)
End If
```

```
Loop While
bResult = False
'*** Create variable
Set obVar = MyWorkspace.ActiveDocument.Variables.CreateVar (strName, m_obDriver,
tpSPSMerker, m_obVarType)
If ( Not obVar Is Nothing) Then
obVar.Offset = nStartOff
nStartOff = nStartOff + nStep
End If
Next nIndex
MousePointer = nPrvMousePtr
Unload Me
Exit Sub
Error :
MousePointer = nPrvMousePtr
MsgBox Error occurs: + Err.Description + Source + Err.Source
End Sub
```

Nel momento in cui viene completata la realizzazione del Wizard, il sistema controlla se le impostazioni che sono state definite sono valide o meno. Nel caso in cui non dovesse essere così, viene generata una segnalazione di errore e l'utente viene invitato a correggere gli inserimenti errati.

Nel caso in cui, invece, le impostazioni sono corrette, il sistema procede alla creazione delle variabili. Tali variabili ricevono un nome ed un indice. Nel caso in cui nel progetto dovesse esistere già una variabile con lo stesso nome, il sistema ricerca il prossimo indice libero. Nel nostro esempio di codice viene sempre creata una variabile del tipo Merker-PLC. Ad ogni nuovo passaggio, l'Offset per la variabile viene aumentato.

4.6.5 Attualizzazione dei wizard.

Per aggiornare i wizard:

1. scegliere dal menu **File** l´inserimento **Update Wizard...**
2. si apre un dialogo che serve ad effettuare l´update dei wizard già disponibili
3. selezionate il wizard desiderato
4. avviate l´update cliccando su **avvia update**

Se un wizard o una classe sono già esistenti nel workspace, sarà visualizzato un messaggio corrispondente

Attenzione

I wizard già esistenti vengono sovrascritti quando il sistema effettua l´update. Le modifiche individuali effettuate nel wizard andranno un questo caso perdute

4.7 Domande ricorrenti

In questo capitolo forniamo delle risposte ad alcune delle domande più frequenti. Ulteriori risposte a problemi ricorrenti le trovate anche online nel COPA-DATA Forum utenti (http://www.copadata.com/forums/).

4.7.1 Perché il pulsante rimane pigiato?

Se un pulsante è collegato per es. con un evento - LeftClickUp, alla fine della procedura deve essere eseguito il LeftClickUp.

```
Public Sub LeftClickUp_Schalter(obElem As Element)
frmSchalter.Show
obElem.LeftClickUp
End Sub
```

4.7.2 La macro non viene eseguita al primo clic.

La soluzione corrisponde a quella della domanda: Perché il pulsante rimane pigiato? (A pagina: 80):

Se un pulsante è collegato per es. con un evento - LeftClickUp, alla fine della procedura deve essere eseguito il LeftClickUp.

```
Public Sub LeftClickUp_Schalter(obElem As Element)
frmSchalter.Show
obElem.LeftClickUp
End Sub
```

4.7.3 Le macro non funzionano più nel RT?

Questo effetto può verificarsi se è aperto nel runtime l'editor VBA e poi si pigia su stop/start per terminare o avviare VBA. In questo caso gli oggetto (variabili online, oggetti d'immagine,....) non sono più validi, dato che perdono il riferimento se ha luogo una nuova inizializzazione.

4.7.4 Windows CE e VBA

VBA è a disposizione nell'Editor CE per la realizzazione di Wizard. VBA non viene supportato nel runtime. Dettagli sull'editor li trovate nel capitolo Crea progetti per CE.

4.8 Esempi

Qui di seguito trovate alcuni esempi per VBA

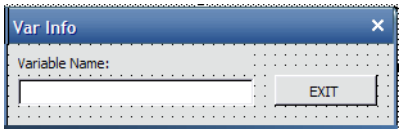
4.8.1 MouseEvents and ActiveX Control initialization

```
Option Explicit


Public Sub Init_ActiveX(obElem As Element)
    'Initializing ActiveX...
    thisProject.Init_MSChart_AX obElem
End Sub

Public Sub LeftClickUp_Sample1(obElem As Element)
    'Initializing Userform...
    frmSample1.InitForm obElem
    'Show Userform
    frmSample1.Show
End Sub

Public Sub LeftClickUp_Sample2(obElem As Element)
```

```
    'Initializing Userform...
    frmSample2.InitForm obElem
    'Show Userform
    frmSample2.Show
End Sub

Public Sub LeftClickUp_Sample3(obElem As Element)
    'Initializing Userform...
    frmSample3.InitForm obElem
    'Show Userform
    frmSample3.Show
End Sub

Public Sub LeftClickUp_Sample4(obElem As Element)
    Dim NewForm As New frmSample4
    'Initializing NEW Userform...
    NewForm.InitForm obElem
    'Show NEW Userform
    NewForm.Show (0)
End Sub
```

4.8.2 Visualizza informazioni variabili

Visualizza nome variabile per elemento su cui si è cliccato:

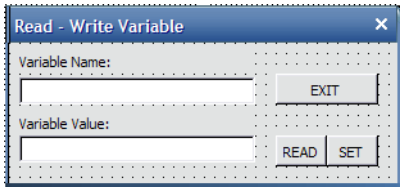



```
Option Explicit

Dim obVar As Variable

'User defined Public Procedure for initializing Objects

Public Sub InitForm(obElem As Element)
    'set the variable object like the linked variable of the element
    Set obVar = obElem.ItemVariable(0)
    'write variable name into the textbox
```

```
    txtVarName.Text = obVar.Name
End Sub

Private Sub cmdExit_Click()
    'close Userform
    Unload Me
End Sub
```

4.8.3 Leggi e scrivi valori variabili

Leggi valori di una variabile e effettua backup:

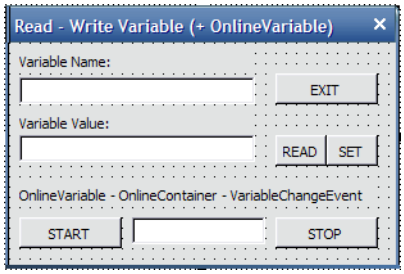



```
Option Explicit

Dim obVar As Variable

'User defined Public Procedure for initializing Objects

Public Sub InitForm(obElem As Element)
    'set the variable object like the linked variable of the element
    Set obVar = obElem.ItemVariable(0)
    'write variable name into the textbox
    txtVarName.Text = obVar.Name
End Sub

Private Sub cmdExit_Click()
    'close Userform
    Unload Me
End Sub

Private Sub cmdRead_Click()
    'read value from variable and write into textbox
    txtValue.Text = obVar.Value
```

```
End Sub

Private Sub cmdWrite_Click()
    'write text as value to variable
    obVar.Value = txtValue.Text
    'or changing text to value before writing...
    'obVar.Value = Val(txtValue.Text)
End Sub
```

4.8.4 Leggi e scrivi variabili e implementa variabili online

Leggi informazioni variabili, scrivi valori e implementa variabili online:



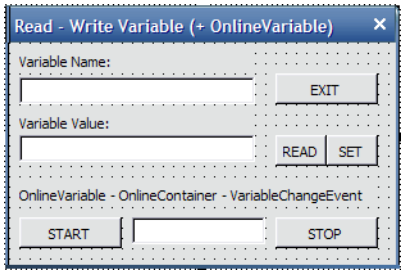

```
Option Explicit

Dim obVar As Variable
Dim WithEvents zOnlineVariable As OnlineVariable

'User defined Public Procedure for initializing Objects

Public Sub InitForm(obElem As Element)
    'set the variable object like the linked variable of the element
    Set obVar = obElem.ItemVariable(0)
    'write variable name into the textbox
    txtVarName.Text = obVar.Name
    'create an OnlineVariable container
    Set zOnlineVariable = thisProject.OnlineVariables.CreateOnlineVariables("OLV")
    'add variables to the container (by name of the variable)
    zOnlineVariable.Add obVar.Name
End Sub
```

```
Private Sub cmdExit_Click()
    'close Userform
    Unload Me
End Sub

Private Sub cmdRead_Click()
    'read value from variable and write into textbox
    txtValue.Text = obVar.Value
End Sub

Private Sub cmdWrite_Click()
    'write text as value to variable
    obVar.Value = txtValue.Text
    'or changing text to value before writing...
    'obVar.Value = Val(txtValue.Text)
End Sub

Private Sub cmdOLV_Start_Click()
    'start the OnlineVariable - Define
    'the VariableChange Event will be executed
    zOnlineVariable.Define
End Sub

Private Sub cmdOLV_Stop_Click()
    'stop the OnlineVariable - UnDefine
    'the VariableChange Event will be stopped
    zOnlineVariable.Undefine
End Sub

Private Sub zOnlineVariable_VariableChange(ByVal obVar As IVariable)
    'write actual value into textbox
    txtOLV.Text = obVar.Value
End Sub

Private Sub UserForm_Terminate()
    'the VariableChange Event will be stopped if running
    zOnlineVariable.Undefine
    'delete OnlineVariable container
    thisProject.OnlineVariables.DeleteOnlineVariables ("OLV")
```

```
End Sub
```

4.8.5 Uso multiplo del dialogo

Userforms possono essere usati più volte:




```
Option Explicit
Dim obVar As Variable
Dim WithEvents zOnlineVariable As OnlineVariable
Dim strOLVName As String

Public Sub InitForm(obElem As Element)
    'set the variable object like the linked variable of the element
    Set obVar = obElem.ItemVariable(0)
    'write variable name into the textbox
    txtVarName.Text = obVar.Name
    'create name for Online Container
    strOLVName = "OLV_" & obElem.Name
    'get existing online container
    Set zOnlineVariable = thisProject.OnlineVariables.Item(strOLVName)
    'check if online container exists
    If zOnlineVariable Is Nothing Then
        'create an OnlineVariable container
        Set zOnlineVariable =
thisProject.OnlineVariables.CreateOnlineVariables(strOLVName)
        'add variables to the container (by name of the variable)
        zOnlineVariable.Add obVar.Name
    End If
End Sub

Private Sub cmdExit_Click()
```

```
    Unload Me       'close Userform
End Sub

Private Sub cmdRead_Click()
    'read value from variable and write into textbox
    txtValue.Text = obVar.Value
End Sub


Private Sub cmdWrite_Click()
    'write text as value to variable
    obVar.Value = txtValue.Text
    'or changing text to value before writing...
    'obVar.Value = Val(txtValue.Text)
End Sub

Private Sub cmdOLV_Start_Click()
    'the VariableChange Event will be executed
    zOnlineVariable.Define
End Sub

Private Sub cmdOLV_Stop_Click()
    'the VariableChange Event will be stopped
    zOnlineVariable.Undefine
End Sub

Private Sub zOnlineVariable_VariableChange(ByVal obVar As IVariable)
    'write actual value into textbox
    txtOLV.Text = obVar.Value
End Sub

Private Sub UserForm_Terminate()
    'the VariableChange Event will be stopped if running
    zOnlineVariable.Undefine
    'delete OnlineVariable container
    thisProject.OnlineVariables.DeleteOnlineVariables (strOLVName)
End Sub
```

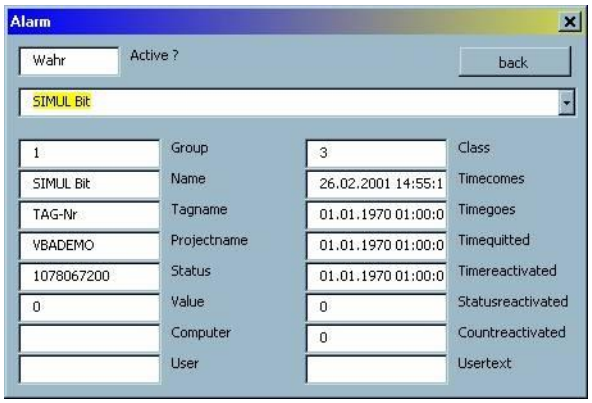
4.8.6 Alarm – Events and ActiveX Control handling

```
Option Explicit

Dim WithEvents obChart As MSChart
Dim WithEvents zOLV As OnlineVariable
Dim WithEvents zAlarm As Alarm

'procedure is executed on startup of the zenon runtime

Private Sub Project_Active()
    'init the alarm object for events
    Set zAlarm = thisProject.Alarm
End Sub

'procedure is executed when an Alarm comes
Private Sub zAlarm_AlarmComes(ByVal obItem As IAlarmItem)
    Dim strInfo As String
    'write specific information about the alarm into a StringVariable
    strInfo = obItem.Text & " - " & obItem.Name
    thisProject.Variables.Item("Var_Comes").Value = strInfo
End Sub

'procedure is executed when an Alarm has gone
Private Sub zAlarm_AlarmGoes(ByVal obItem As IAlarmItem)
    Dim strInfo As String
    'write specific information about the alarm into a StringVariable
    strInfo = obItem.Text & " - " & obItem.Name
    thisProject.Variables.Item("Var_Goes").Value = strInfo
End Sub

'procedure is executed when an Alarm was acknowledged by a user
Private Sub zAlarm_AlarmAcknowledged(ByVal obItem As IAlarmItem)
    Dim strInfo As String
    'write specific information about the alarm into a StringVariable
    strInfo = obItem.Text & " - " & obItem.Name
    thisProject.Variables.Item("Var_Acknowledged").Value = strInfo
End Sub
```

```
'procedure is executed on terminating the zenon Runtime
Private Sub Project_Inactive()
    'free the alarm object
    Set zAlarm = Nothing
    'delete OnlineVariable for Chart actualization...
    thisProject.OnlineVariables.DeleteOnlineVariables "CHART"
End Sub

'procedure for MSChart ActiveX initialization...
Public Sub Init_MSChart_AX(YourAX As Element)
    Set obChart = YourAX.AktiveX
    obChart.RowCount = 3
    obChart.ColumnCount = 1
    Set zOLV = thisProject.OnlineVariables.Item("CHART")
    'does existing OnlineVariable exist?
    If zOLV Is Nothing Then
        'if not, create it...
        Set zOLV = thisProject.OnlineVariables.CreateOnlineVariables("CHART")
        zOLV.Add "Internal_UINT_001"
        zOLV.Add "Internal_UINT_002"
        zOLV.Add "Internal_UINT_003"
    End If
    zOLV.Undefine 'if not stopped, refreshing not possible
    'START watching variables...
    zOLV.Define
End Sub

'event on Variable change - refresh chart...
Private Sub zOLV_VariableChange(ByVal obVar As IVariable)
    'setting values to display in chart control
    Select Case obVar.Name
        Case "Internal_UINT_001"
            obChart.Row = 1
            obChart.RowLabel = "Var1"
            obChart.Data = obVar.Value
        Case "Internal_UINT_002"
            obChart.Row = 2
            obChart.RowLabel = "Var2"
            obChart.Data = obVar.Value
```

```
        Case "Internal_UINT_003"
            obChart.Row = 3
            obChart.RowLabel = "Var3"
            obChart.Data = obVar.Value
    End Select
End Sub

'event of the Chart AX...
Private Sub obChart_DblClick()
    MsgBox "You have DoubleClicked the ActiveX!"
End Sub
```

4.8.7 Accesso ad allarmi



DESCRIZIONE:

Nel formulario frmAlarm si può selezionare in un combobox un allarme che si trova nella memoria. Dopo avere effettuato la selezione, tutti i dati riguardanti l'allarme vengono inseriti nei box di testo sottostanti (gruppo, calsse, variabile ...)

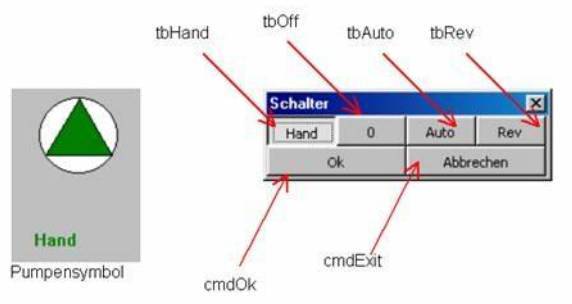
Per visualizzare frmAlarm, utilizziamo un evento macro indipendente visto che non stabiliamo un collegamente ad un elemento nel presente esempio.

```
Sub Alarm ()
frmAlarm.Show
End Sub
'La macro viene eseguita grazie alla funzione "esegui macro.."
```

```
'Con l´apertura della forma questa viene inizializzata, ed eseguita la seguente routine.
Da essa dipende il fatto che la Combobox impostata nella forma riceve tutti gli allarmi
che si trovano al momento in memoria.

Private Sub UserForm _ Initialize ()

'fill combobox with all AlarmItems
Dim i As Integer
Dim obAlarmItems As AlarmItems
Dim obAlarm As Alarm

Set obAlarm = thisProject.Alarm
Set obAlarmItems = obAlarm.AlarmItems (*)

If obAlarmItems.Count = 0 Then
MsgBox (# Alarms = 0 )
Exit Sub
End If

For i = 0 To obAlarmItems.Count - 1
cmbAlarmItems.AddItem obAlarmItems.Item ( i ). Name
Next i

txtAktiv.Text = obAlarm.Aktiv
cmbAlarmItems.Text = cmbAlarmItems.List ( 0 )
End Sub


'Se viene selezionato solo un allarme del Combobox, reagisce l'evento Change del Combobox.
Private Sub cmbAlarmItems _ Change ()

'put actual properties from AlarmItem in textboxes
Dim obAlarmItems As AlarmItems
Dim obAlar As Alarm

Set obAlarm = thisProject.Alarm
 Set obAlarmItems = obAlarm.AlarmItems (*)
txtComputer.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Computer
txtCountreactivated.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Countreactivated
txtName.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ). Name
txtProjectname.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Projectname
txtStatus.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Status
txtStatusreactivated.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex
).Statusreactivated
txtTagname.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Tagname
txtTimecomes.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Timecomes
txtTimegoes.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Timegoes
txtTimequitted.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Timequitted
```

```
txtTimereactivated.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Timereactivated
txtUser.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).User
txtUsertext.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Usertext
txtValue.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).Value
tbGroup.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).AlarmGroup
tbClass.Text = obAlarmItems.Item ( cmbAlarmItems.ListIndex ).AlarmClass
End Sub
```

4.8.8 Crea un interruttore (lavorare con variabili di processo)



In questo esempio si disegna una pompa che consiste di un cerchio e di un triangolo. Quest'ultimo lo definite Voi come simbolo proprio. Sopra ponete un elemento multibin e collegatelo con tre variabili bit merker.

Si deve inoltre stabilire di quale colore deve essere il triangolo al centro in caso di una modifica del valore delle variabili.

In primo luogo si collega l'elemento multibin con una macro che fa apparire una forma frminterruttore.

Mediante la forma frminterruttore si può modificare il valore delle tre variabili bit merker.

Informazioni su

Solo una delle tre variabili può contenere il valore 1. (ciò significa che se una variabile viene settata sul valore 1, le altre due devono ricevere il valore 0)

Per utilizzare questa macro più volte in un progetto in riferimento a diverse variabili, dovete fare attenzione di collegare con l'elemento multibin solo variabili bit merker che contengono già nel nome della variabile quale stato della pompa gestiscono.

Per es.:
Variable_Auto
Variable_Hand
Variable_Revi

Informazioni su

Le desinenze _Auto, _Hand e _Revi sono definite in modo fisso nel codice sorgente dell'esempio.

Mediante queste desinenze di cinque segni del nome della variabile, si può definire ora quale variabile viene settata sul valore 1 cliccando su un pulsante toggle e quale sul valore 0.

Nella macro LeftClickUp_interruttore viene richiamato un sottoprogramma FindVariable nella formai frminterruttore, cui viene trasmesso l'elemento selezionato obElem.

```
Public Sub LeftClickUp_Schalter (obElem As Element)
frmSchalter.FindVariable obElem
position (pixel to points = (pixel * 0.75))
frmSchalter.Top = obElem.Bottom * 0.75
frmSchalter.Left = obElem.Left * 0.75
frmSchalter.Show
obElem.LeftClickUp
End Sub
```

Dichiarazione moduli globali variabili:

```
Dim cmdLast As ToggleSchaltfläche
Dim strHand As String
Dim strAuto As String
Dim strRevi As String
Nel sottoprogramma FindVariable vengono controllote anzitutto tutte le variabili che sono
offerte dall'elemento trasmesso.
```

A seconda della desinenza (_Auto, _Hand o _Revi), i nomi delle variabili vengono attribuiti alle variabili string sopra dichiarate.

Inoltre il sistema rileva quale stato ha la variabile e, a seconda del valore (1 o 0), viene azionato il corrispondente pulsante toggle o no.

Al momento dell'apertura della forma frminterruttore, il nome del pulsante Toggle al momento premuto viene scritto in una variabile string. Nel caso in cui l'utente decida di interrompere la procedura di modifica del valore, vengono ripristinati i valori originali.

```
Public Sub FindVariable (obElem As Element)
Dim i As Integer
Dim obVariable As Variable

For i = 0 To obElem . CountVariable - 1
Select Case Right $( obElem . ItemVariable ( i ). Name , 5 )
Case _ Auto
strAuto = obElem . ItemVariable ( i ). Name

Case _ Hand
strHand = obElem . ItemVariable ( i ). Name

Case _ Revi
strRevi = obElem . ItemVariable ( i ). Name
End Select
Next i

Set obVariable = thisProject . Variables . Item ( strHand )
If obVariable . Value = 1 Then
tbHand . Value = True
Set cmdLast = tbHand
End If

Set obVariable = thisProject . Variables . Item ( strAuto )
If obVariable . Value = 1 Then
tbHand . Value = True
Set cmdLast = tbAuto
End If

Set obVariable = thisProject . Variables . Item ( strRevi )
If obVariable . Value = 1 Then
tbRev . Value = True
Set cmdLast = tbRev
End If

If tbHand . Value = False And tbAuto . Value = False And tbRev . Value = False Then
tbOff . Value = True
Set cmdLast = tbOff
End If
End Sub
```

La Function VarExists generata autonomamente controlla solamente se le variabili collegate esistono; se ciò non dovesse corrispondere alla realtà, segue una segnalazione di errore. Variable doesn't exist.

```
Function VarExists ()

Dim obVariable As Variable
Set obVariable = thisProject . Variables . Item (strHand)

If obVariable Is Nothing Then
MsgBox (Variable doesnt extist)
VarExitsts = False
Exit Function
End If

Set obVariable = thisProject . Variables . Item (strAuto)
If obVariable Is Nothing Then
MsgBox ( Variable doesnt extist )
VarExitsts = False
Exit Function
End If

Set obVariable = thisProject . Variables . Item (strRev)
If obVariable Is Nothing Then
MsgBox ( Variable doesnt extist )
VarExitsts = False
Exit Function
End If

VarExists = True
End Function
```

Se l'utente clicca su "Annula", la modifica del valore viene annullata e viene ripristinato lo stato originario.

```
Private Sub cmdExit _ Click ()
cmdLast.Value = True
Unload Me
End Sub

Private Sub cmdOk _ Click ()
Unload Me
End Sub
```

Se un pulsante toggle è premuto, non ne deve essere attivo nessun altro.

```
Private Sub tbAuto_Change ()
If tbAuto . Value = False And tbHand.Value = False And tbRev . Value = False Then
tbOff . Value = True
End Sub
```

In "eventi click" di ogni pulsante toggle si controlla se questo non è premuto e se la variabile esiste. Nel caso in cui entrambe le cose corrispondano alla realtà, vengono trasmessi i valori delle variabili collegate.

5. VSTA

Con Visual Studio Tools for Applications - VSTA, la funzionalità del runtime e dell´editor di zenon può essere ampliata con l´aggiunta della programmazione .NET.

VSTA è disponibile, seppure con limitazioni, anche per zenon Web Server e per zenon Web Client.

Informazioni su

Se si modificano delle macro VBA nell´Editor:

- vengono compilati i file Runtime e trasmessi nel Runtime.
- il runtime viene riavviato.
- anche gli elementi VSTA vengono ricaricati, anche quando non c´è stata alcuna modifica in VSTA .

VSTA mette a disposizione ambienti di sviluppo separati per workspace e progetto. Se ne può usare solamente uno alla volta. Al momento dello start, ogni altro ambiente di sviluppo VSTA già aperto verrà chiuso.

Per aprire l´Editor VSTA per il workspace:

1. premete la combinazione tasti **Alt+F10**
2. verrà visualizzato il codice per il workspace e tutti i progetti caricati.

Per aprire l´Editor VSTA per il progetto attualmente caricato:

1. navigate per raggiungere il nodo**`interfacce di programmazione`**
2. cliccate poi con il tasto destro del mouse su VSTA.
3. selezionate nel menù contestuale **`Apri VSTA-Editor...`**

l´Editor verrà aperto per il progetto attualmente caricato.

5.1 Basi:

VSTA è uno sviluppo della ditta Microsoft che si basa sulla tecnologia .NET. Per potere comprendere il programma, sono necessarie delle conoscenze di base concernenti la programmazione orientata agli oggetti, .NET e C#/Visual Basic.NET.

5.1.1 Impostazione dell´ambiente VSTA

Di default, il supporto per VSTA in zenon è già attivato. Disattivando il supporto VBA, anche l´ambiente VSTA non è più disponibile.

Mediante il seguente inserimento in C:\Users\All Users\COPA-DATA\System\zenon6.ini, l´ambiente VSTA può essere attivato o disattivato manualmente:

attivare VSTA	disattivare VSTA
[VSTA] ON=1	[VSTA] ON=0

Il supporto per VBA viene attivato o disattivato nel modo seguente:

attivare VBA	disattivare VBA
[VBA] EIN=1	[VBA] EIN=0

In seguito, l´ambiente di sviluppo per VSTA sarà disponibilie in zenon.

Informazioni su

VSTA consente lo sviluppo di progetti nelle lingue di programmazione C# e Visual Basic.NET. Come linguaggio standard per progetti VSTA nell´Editor è prevista C#. Mediante il seguente inserimento, il linguaggio può essere cambiato passando a Visual Basic.NET:

[VSTA]

CSHARP=0

5.1.2 Accesso al modello oggetti in zenon

In VSTA si può accedere al modello oggetti di zenon che viene usato anche in VBA. Il modello oggetto rimane in entrambi gli ambienti di sviluppo la stessa funzionalità.

Attenzione

A causa di limitazioni nella denominazione di modelli oggetti in VSTA, sono state realizzate alcune modifiche del modello ogetto. Le trovate nella tabella qui sotto

Property alt	Property neu
IDriver.Name	IDriver.Identification
IDriver.Driver	IDriver.Name
Event alt	**Event neu**
IApplication.Close	IApplication.OnClose
IZenWorkspace.Startup	IZenWorkspace.OnStartup
IZenWorkspace.Exit	IZenWorkspace.OnExit

L´accesso si verifica in VSTA mediante l´oggetto `this` e sostituisce l´oggetto VBA `MyWorkspace`. I metodi e gli oggetti successivi sono identici. Nei seguenti metodi vene creato un modello con il nome "TemplateName" in zenon.

```
public void Macro1()
{
   this.ActiveDocument.Templates().Create("TemplateName", true);
}
```

Informazioni su

Diversamente da quanto accade in VBA, in VSTA si dà importanza alla scrittura maiuscola e minuscola, nonché a parentesi dopo il nome della funzione.

Per aprire i metodi in zenon, il progetto deve essere salvato e compilato seguendo i seguenti passi:

1. Cliccando su *File -> Save MyWorkspace.cs*, il progetto viene salvato.
2. Cliccando su *Build -> Build WorkspaceAddin*, il progetto viene compilato.

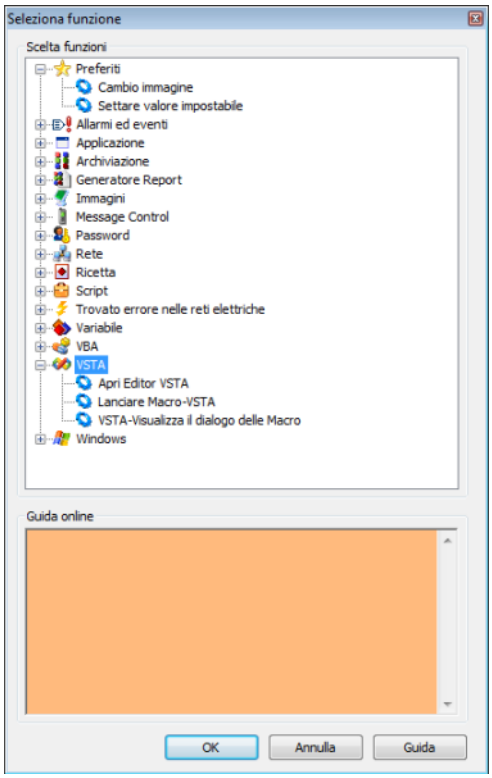
Il metodo è poi a Vostra disposizione come macro nella barra degli strumentidelle macro VBA che si trova nell´editor di zenon. Nel caso in cui il dialogo per l'assegnazione delle macro non dovesse offrire tutte le macro contenute in MyWorkspace, la funzione "Ricarica lista delle macro VBA" dovrebbe essere eseguita nella barra degli strumenti.

Informazioni su

Macro VSTA con parametro, per es. `Public void MacroWithParam(string mString)`*, non vengono supportate e non vengono neppure offerte all´utente nella barra degli strumenti delle macro.*

5.1.3 Funzioni in zenon

In zenon sono state sviluppate le seguenti nuove funzioni per VSTA. Esse si trovano nel gruppo di funzioni "VSTA".



In parallelo alle funzioni VBA esistenti, sono state implementate delle funzioni simili per VSTA:

Nome funzione	Descrizione
Apri Editor VSTA	apre l´editor di VSTA nel runtime
Lanciare Macro-VSTA	Nell´editor può essere selezionata una macro VSTA che viene avviata al momento dell´esecuzione della funzione nel runtime. **Nota:** Macro VSTA con parametro, per es. `Public void MacroWithParam(string mString)`, non vengono supportate. Esse non vengono offerte all´utente né al momento della progettazione nell´Editor, né allo start della funzione nel Runtime.
VSTA-Visualizza il dialogo delle Macro	Nel runtime viene visualizzato un dialogo in cui possono essere visualizzate, selezionate ed eseguite le macro a disposizione.

5.1.4 Debugging di un Add-in VSTA

Mediante l´editor di VSTA, è possibile realizzare il debugging di AddIn scritti individualmente. Bisogna tenere presente che si può realizzare il debugging di Project-AddIns solamente nel runtime di zenon, di Workspace-AddIns solo nell´editor di zenon.

Una sessione di debugging viene avviata mediante il menu *Debug - Start Debugging*. Dei breakpoint possono essere fissati come nell´editor di VBA, cioè cliccando con il tasto sinistro del mouse nella barra di breakpoint di color grigio che si trova sul bordo sinistro accanto alla riga corrispondente.

Informazioni su

Nell´effettuare il debugging di Runtime Add-in, tenete presente quanto segue:

I file Runtime modificati in zenon devono essere generati di nuovo prima dell´inizio dell´operazione di debugging.

5.1.5 Nuovi eventi in VSTA

Visto che accade una terminazione di un AddIn al momento della compilazione di un codice modificato, al momento dell´avvio di una sessione di debugging o al momento della chiusura della stessa, sono stati implementati in VSTA nuovi eventi. Essi permettono, ad esempio, la valutazione e il rilascio di una referenza oggetto e il salvataggio dei dati esistenti.

Per ogni terminazione esistono due events. Il primo di essi viene avviato poco prima della terminazione, il secondo dopo l´avvio di una nuova sessione AddIn.

Event	Descrizione
`OnPreVSTADebugStart`	Vene effettuato il trigger prima che una sessione di debugging venga avviata. Al momento dello start, viene eliminato un AddIn attivo, delle referenze devono essere rese disponibili e i dati disponibili devono essere salvati.
`OnVSTADebugStart`	Vene effettuato il trigger dopo che una sessione di debugging è stata avviata.
`OnPreVSTADebugStop`	Vene effettuato il trigger prima che una sessione di debugging venga arrestata. Al momento dell´interruzione di una sessione di debugging, viene eliminato un AddIn attivo, delle referenze devono essere rese disponibili e i dati disponibili devono essere salvati.
`OnVSTADebugStopped`	Vene effettuato il trigger dopo che una sessione di debugging è stata arrestata.
`OnPreVSTAUpdate`	Viene effettuato il trigger prima dell´eliminazione dell´AddIn, se è stata generata con successo una nuova versione dell´AddIn.
`OnPostVSTAUpdate`	Vene effettuato il trigger quando viene caricata una nuova versione dell´AddIn.

5.1.6 Backup di progetti VSTA

I progetti VSTA nel runtime vengono zippati automaticamente al momento della generazione dei file runtime e sono contenuti in Workspace backup.

Per i progetti VSTA nell´edtor, invece, si deve effettuare un backup manuale. Nella cartella C:/ProgramData/COPA-DATA/*version*/VSTAWorkspace/, si trovano i progetti VSTA dell´editor.

5.2 Creazione di un progetto VSTA

In modo simile a quanto accade per VBA, esiste anche in VSTA la possibilità di creare dei progetti sia per l´editor che per il runtime. In linea di principio, i progetti nell´editor vengono relizzati nel linguaggio di programmazione C#. Per quello che concerne il runtime, invece, sono a Vostra disposizione sia C# che Visual Basic.NET.

> **Informazioni su**
>
> *Nell´ambito dell´editor di VSTA, può essere visualizzato solamente un progetto alla volta. Inoltre, può essere attiva solamente un´istanza alla volta dell´editor. Al momento dell´avvio dell´editor di VSTA, viene chiusa un´istanza già eventualmente aperta.*

5.2.1 Progetti VSTA nell´editor

Al momento della creazione di un progetto per l´editor di zenon, un VSTA-AddIn viene caricato per il workspace. Per modificare l´Add-In, bisogna aprire l´editor di VSTA mediante *File - Aprire editor VSTA...*. L´interfaccia utente dell´editor VSTA è identica all´ambiente di sviluppo di Visual Studio della ditta Microsoft.

> **Informazioni su**
>
> *Mediante il menu Help / Contents , si può aprire la guida dell´editor VSTA. Questa guida fornisce una visione d´insieme sulle funzioni dell´editor, le features del framework .NET e la programmazione in Visual Basic.NET e C#.*

Il VSTA-Add-In consiste essenzialmente della classe `MyWorkspace`. Questa classe può essere ampliata solamente con l´aggiunta di metodi propri. La classe ospita di default i seguenti due metodi:

Funzione	Descrizione
`MyWorkspace_Startup`	viene eseguito automaticamente al momento dell´avvio di zenon, dopo la creazione di un build e al momento dell´avvio di una sessione di debugging.
`MyWorkspace_Shutdown`	viene eseguito automaticamente al momento dellla chiusura di zenon, dopo la creazione di un build e al

	momento della chiusura di una sessione di debugging.

Attenzione

I nomi metodi possono iniziare esclusivamente con `Macro` *(per es. Macro1, MacroVSTA) , non devono contenere nessun parametro e devono essere definiti come* `Public` *. Inoltre, il nome classe e altri metodi e event creati da VSTA non possono essere modificati.*

Per aprire i metodi in zenon, il progetto deve essere salvato e compilato seguendo i seguenti passi:

1. Cliccando su *File -> Save MyWorkspace.cs*, il progetto viene salvato.
2. Cliccando su *Build -> Build WorkspaceAddin*, il progetto viene compilato.

Il metodo è poi a Vostra disposizione come macro nella barra degli strumentidelle macro VBA che si trova nell´editor di zenon. Nel caso in cui il dialogo per l'assegnazione delle macro non dovesse offrire tutte le macro contenute in MyWorkspace, la funzione "Ricarica lista delle macro VBA" dovrebbe essere eseguita nella barra degli strumenti.

5.2.2 Progetti VSTA nel runtime

Per creare un progetto VSTA per il runtime, si deve anzitutto aprire l´ambiente VSTA. A tal scopo, procedete nel modo seguente:

1. *aprite nel manager di progetto il punto Interfacce di programmazione.*
2. *Aprite il menu contestuale di VSTA.*
3. *Cliccate su Aprire editor VSTA....*

Selezionate adesso nel dialogo che comparirà il linguaggio che desiderate. A questo punto verrà impostato un progetto nel linguaggio che avete selezionato.

Informazioni su

Il linguaggio selezionato non può essere modificato in un secondo momento; per questo motivo, questo dialogo viene visualizzato solo la prima volta.

In questo progetto viene impostata da zenon una classe dal nome `ThisProject` , che ospita i seguenti due metodi:

Funzione	Descrizione
`ThisProject_Startup`	viene eseguito automaticamente al momento dell´avvio del runtime.
`ThisProject_Shutdown`	viene eseguito automaticamente al momento della chiusura del runtime.

Questa classe può essere ampliata solamente con l´aggiunta di metodi propri.

Attenzione

I nomi metodi possono iniziare esclusivamente con `Macro` *(per es. Macro1, MacroVSTA) , non devono contenere nessun parametro e devono essere definiti come* `Public` *. Inoltre, il nome classe e altri metodi e event creati da VSTA non possono essere modificati.*

Esiste la possibilità di accedere tramite il modello oggetti di zenon a tutte le funzionalità runtime. Le funzioni specifiche dell´editor non possono essere usate in VBA.

Il runtime di zenon viene avviato automaticamente al momento dell´avvio del debugger. Trovate ulteriori informazioni nel capitolo Debugging di un Add-In VSTA- (A pagina: 100).

Informazioni su

Mediante il menu Help / Contents , si può aprire la guida dell´editor VSTA. Questa guida fornisce una visione d´insieme sulle funzioni dell´editor, le features del framework .NET e la programmazione in Visual Basic.NET e C#.

5.2.3 Sviluppo di wizard in VSTA

L´ambiente VSTA offre come VBA (A pagina: 71) la possibilità di sviluppare propri wizard.

Per poter accedere in un modulo all´oggetto modello zenon, deve essere trasmessa una referenza al modulo stesso. A questo scopo, si crea un metodo nella classe `MyWorkspace` . Nell´esempio seguente, viene instanziato un modulo dal nome `Wizard` e viene richiamato il metodo `ZenonInstance` con una referenza all´oggetto modello zenon come parametro. Mediante `ShowDialog()`, viene visualizzato il modello Wizard.

```
public void Macro1()
{
```

```
        Form1 Wizard = new Form1();
        Wizard.ZenonInstance(this.Application);
        Wizard.ShowDialog();
    }
```

Nel codice del modello deve essere impostata una variabile member, che riprende l´oggetto modello zenon.

```
    public zenOn.IApplication m_Zenon=null;
```

Per ultimo, viene impostato il metodo **ZenonInstance** . Questo metodo riprende la referenza all´oggetto modello e la pone sull´oggetto **m_Zenon**.

```
    public void ZenonInstance(zenOn.IApplication app)
    {
        m_Zenon = app;
    }
```

Adesso, si può procedere allo sviluppo di classi e metodi propri nel modello che accedono all´oggetto modello. Tutti i metodi, gli oggetti e gli attributi sono a disposizione tramite l´oggetto **m_Zenon**.

5.3 Esempi

Qui di seguito trovate alcuni esempi dell´uso di VSTA sia nel runtime che nell´editor.

5.3.1 Creazione di variabili nell´editor di zenon

Nel seguente esempio, un file di testo viene aperto e il contenuto viene utilizzato per impostare delle variabili nell´editor di zenon. Il file di testo contiene un numero qualsiasi di righe. Ogni riga contiene il nome e il tipo di dato di una variabile, separati da una virgola (esempio: Variable1,BOOL).

Il metodo **Macro1** cerca anzitutto il driver interno nell´editor di zenon. Poi, il sistema visualizza a beneficio dell´utente un dialogo che gli propone una scelta di file; l´utente deve selezionare il file di testo. In seguito, il metodo legge il file di testo e crea le variabili. Il metodo **GetDataType** serve a individuare e ad assegnare al momento della creazione delle variabili il tipo di dato corrispondente.

```
using System;
using System.Windows.Forms;
using System.IO;
namespace WorkspaceAddin
{
```

```
[System.AddIn.AddIn("MyWorkspace", Version = "1.0", Publisher = "", Description = "")]
public partial class MyWorkspace
{
    private void MyWorkspace_Startup(object sender, EventArgs e)
    {
    }
    private void MyWorkspace_Shutdown(object sender, EventArgs e)
    {
    }
    public zenOn.IVarType GetDataType(zenOn.IVarType vType, string strVType)
    {
        //gets the corresponding vartypes for bool, int, real and strings
        switch (strVType)
        {
            case "BOOL":
                vType = this.ActiveDocument.VarTypes().Item("BOOL");
                break;
            case "INT":
                vType = this.ActiveDocument.VarTypes().Item("INT");
                break;
            case "REAL":
                vType = this.ActiveDocument.VarTypes().Item("REAL");
                break;
            case "STRING":
                vType = this.ActiveDocument.VarTypes().Item("STRING");
                break;
            default:
                vType = this.ActiveDocument.VarTypes().Item("INT");
                break;
        }
        return vType;
    }
    public void Macro1()
    {

        //create objects that will take the intern driver and the variable type
        zenOn.IDriver zInternDriver = null;
        zenOn.IVarType vType = null;
        //search for the Intern driver and throw exception if no driver was found
```

```
            try
            {
                for (int nDriverCount = 0; nDriverCount <
this.ActiveDocument.Drivers().Count; nDriverCount++)
                {
                    if (this.ActiveDocument.Drivers().Item(nDriverCount).Name ==
"Intern")
                    {
                        zInternDriver =
this.ActiveDocument.Drivers().Item(nDriverCount);
                    }
                }
            }
            catch (Exception xDrv)
            {
                MessageBox.Show("Unable to find zenon 'Intern' driver. Error: " +
xDrv.Message);
                throw;
            }
            this.ActiveDocument.Variables().DoAutoSave(false);
            try
            {
                OpenFileDialog VarFileSelect = new OpenFileDialog();
                String[] strVarLine = new String[2];

                //show file dialog
                if (VarFileSelect.ShowDialog() == DialogResult.OK)
                {
                    string strLine = string.Empty;
                    //open new stream reader with selected file
                    StreamReader ImportStream = new StreamReader(VarFileSelect.FileName,
System.Text.Encoding.Default);

                    //read in line by line, split the lines when a ',' occurs and create
variables
                    while ((strLine = ImportStream.ReadLine()) != null)
                    {
                        strVarLine = strLine.Split(new Char[] { ',' });
                        this.ActiveDocument.Variables().CreateVar(strVarLine[0],
zInternDriver, zenOn.tpKanaltypes.tpSystemVariable, GetDataType(vType,strVarLine[1]));
```

```
                    }
                    ImportStream.Close();
                }
            }
            catch (Exception xFileRd)
            {
                MessageBox.Show("An error occured while opening the file: " +
xFileRd.Message);
                throw;
            }
            this.ActiveDocument.Variables().DoAutoSave(true);
        }
        #region VSTA generated code
        private void InternalStartup()
        {
            this.Startup += new System.EventHandler(MyWorkspace_Startup);
            this.Shutdown += new System.EventHandler(MyWorkspace_Shutdown);
        }
        #endregion
    }
}
```

5.3.2 Scrittura di informazioni di progetto nella finestra di emissione di zenon

Nell´esempio seguente si vuole mostrare come, tramite VSTA, si possa accedere alla finestra di emissione dell´editor zenon. Il metodo dal nome `Macro1` legge nel progetto le immagini di processo impostate, individua il modello corrispondente e tutti i driver disponibili, nonché la loro denominazione.

```
using System;
namespace WorkspaceAddin
{
    [System.AddIn.AddIn("MyWorkspace", Version = "1.0", Publisher = "", Description = "")]
    public partial class MyWorkspace
    {
        private void MyWorkspace_Startup(object sender, EventArgs e)
        {
        }
        private void MyWorkspace_Shutdown(object sender, EventArgs e)
        {
```

```
        }
        public void Macro1()
        {
            string strPicName = string.Empty;
            string strCorTemp = string.Empty;
            string strDriverName = string.Empty;
            string strDrvDesc = string.Empty;
            //print start string into output window
            this.Application.DebugPrint(" --------------START--------------",
zenOn.tpDebugPrintStyle.tpMsg);
            //go through all pictures and print name and used template into output window
            for (int i = 0; i < this.ActiveDocument.DynPictures().Count; i++)
            {
                strPicName = this.ActiveDocument.DynPictures().Item(i).Name;
                strCorTemp =
this.ActiveDocument.DynPictures().Item(i).get_DynProperties("Template").ToString();
                this.Application.DebugPrint(" Picture '" + strPicName + "' uses Template
'" + strCorTemp + "'",zenOn.tpDebugPrintStyle.tpMsg);
            }
            //print seperator string into output window
            this.Application.DebugPrint("
------------------------------",zenOn.tpDebugPrintStyle.tpMsg);
            //go through all drivers and print name and description into output window
            for (int i = 0; i < this.ActiveDocument.Drivers().Count; i++)
            {
                strDriverName = this.ActiveDocument.Drivers().Item(i).Name;
                strDrvDesc = this.ActiveDocument.Drivers().Item(i).Identification;
                this.Application.DebugPrint(" Driver '" + strDriverName + "' has
description '" + strDrvDesc + "'", zenOn.tpDebugPrintStyle.tpMsg);
            }
            //print end string into output window
            this.Application.DebugPrint(" ---------------END---------------",
zenOn.tpDebugPrintStyle.tpMsg);
        }
        #region VSTA generated code
        private void InternalStartup()
        {
            this.Startup += new System.EventHandler(MyWorkspace_Startup);
            this.Shutdown += new System.EventHandler(MyWorkspace_Shutdown);
        }
```

```
        #endregion
    }
}
```

5.3.3 Leggi variabili in zenon mediante Regular Expressions

Nel seguente esempio, delle variabili zenon vengono lette in un progetto runtime e salvate in un file di testo locale.

Grazie a Regular Expressions, vengono lette le variabili, i cui nomi iniziano con 3 numeri e un trattino un basso (per es. "001_var" oppure "234_xyz"). Il sistema chiede poi all´utente di scegliere una cartella. In questa cartella viene impostato un file di testo con un nome dipendente dal tempo. In questo file vengono salvati il nome, l´identificazione e il valore attuale di tutte le variabili in questione separate da un trattino.

Informazioni su

E´ possibile che si debbano aggiungere manualmente delle referenze per eseguire l´esempio nel runtime di zenon. A questo scopo, aprite nel Project Explorer il menu contestuale e cliccate su Add Reference... Le referenze necessarie in questo esempio sono le seuenti:

- Microsoft.VisualStudio.Tools.Applications.Runtime.v9.0
- Sistema
- System.AddIn
- System.Data
- System.Windows.Forms
- System.Xml
- zenonVSTAProxy6500

```
using System;
using System.Text.RegularExpressions;
using System.IO;
using System.Windows.Forms;
namespace ProjectAddin
```

```
{
    [System.AddIn.AddIn("ThisProject", Version = "1.0", Publisher = "", Description = "")]
    public partial class ThisProject
    {
        private void ThisProject_Startup(object sender, EventArgs e)
        {
        }
        private void ThisProject_Shutdown(object sender, EventArgs e)
        {
        }
        public void Macro1()
        {
            string sFilename = string.Empty;
            string strName = string.Empty;
            string strDescription = string.Empty;
            string strValue = string.Empty;

            //define regular expression pattern
            Regex rexMatch = new Regex("^([0-9]){3}[_]");
            try
            {
                sFilename = FolderSelection("Select place to store the variable
information");

                //create stream writer to the .txt file
                StreamWriter MatchedVariables = new StreamWriter(sFilename, true);
                //run through all variables in zenon
                for (int i = 0; i < this.Variables().Count; i++)
                {
                    //if name of the variable matches the pattern, get name, tag name and
current value
                    if (rexMatch.IsMatch(this.Variables().Item(i).Name))
                    {
                        strName = this.Variables().Item(i).Name;
                        strDescription = this.Variables().Item(i).Tagname;
                        strValue = this.Variables().Item(i).get_Value(0).ToString();
                        //write information to the .txt file
                        MatchedVariables.WriteLine(strName + ";" + strDescription + ";"
+ strValue);
```

```
                }
            }
            //close stream
            MatchedVariables.Close();
        }
        catch (Exception xFS)
        {
            MessageBox.Show ("An error occured -> " + xFS.Message);
            throw;
        }
    }
    private string FolderSelection(String strCaption)
    {
        string strSelectedPath = string.Empty;
        //create a dialog for selecting the output folder
        FolderBrowserDialog FolderSelectionDialog = new FolderBrowserDialog();
        FolderSelectionDialog.Description = strCaption;
        try
        {
            if (FolderSelectionDialog.ShowDialog() == DialogResult.OK)
            {
                //if selection was valid, get the current date, put it to file date
format
              //then create a txt file with the name "zenonVar" and the corresponding
date
                DateTime currentTime = DateTime.Now;
                strSelectedPath = FolderSelectionDialog.SelectedPath + "\\zenonVar"
+ currentTime.ToFileTime() + ".txt";
            }
        }
        catch (Exception xFD)
        {
            MessageBox.Show("An error occured: "+xFD.Message);
            throw;
        }
        return strSelectedPath;
    }
    #region VSTA generated code
    private void InternalStartup()
```

```
        {
            this.Startup += new System.EventHandler(ThisProject_Startup);
            this.Shutdown += new System.EventHandler(ThisProject_Shutdown);
        }
        #endregion
    }
}
```