COPADATA
do it your way

Manuale di zenon

Interfacce di programmazione

v.8.00

©2018 Ing. Punzenberger COPA-DATA GmbH

Tutti i diritti riservati.

La distribuzione e la copia di questo documento - indifferentemente dal metodo usato - è consentita solo con il permesso scritto della società COPA-DATA. I dati tecnici servono solo alla descrizione del prodotto e non rappresentano in alcun modo caratteristiche garantite in senso giuridico. Con riserva di modifiche - anche di tipo tecnico.

Indice

1. Benvenuti nell'help di COPA-DATA 7

2. Interfacce di programmazione 7

3. Add-Ins 10

3.1 Conoscenze preliminari 11

3.2 Terminologia 12

3.3 Restrizioni 14

3.4 Creare Add-In 14

3.4.1 Nozioni di base 15

3.4.2 Tipi di Extensions 19

3.4.3 Contenuto di Add-In Packages 21

3.4.4 Microsoft Visual Studio 22

3.4.5 SharpDevelop 29

3.4.6 Gestione del codice sorgente 32

3.4.7 Analisi di Add-In e Packaging Utility (AddInUtility) 32

3.4.8 Comportamento durante il reload 33

3.4.9 Isolamento 34

3.5 Utilizzo di Add-Ins nell´Editor e a Runtime 34

3.5.1 Come attivare Add-Ins in zenon 36

3.5.2 Comportamento durante l´installazione 36

3.5.3 Nodo “Add-Ins” nel manager di progetto 37

3.5.4 Utilizzo nell´Editor di zenon: 38

3.5.5 Utilizzo nel Runtime di zenon 43

3.5.6 Risoluzione dei problemi 46

3.6 Funzioni di zenon 46

3.6.1 Visualizza dialogo "Gestione dei Servizi Runtime" 47

3.6.2 Avvia Project Wizard Extension 50

3.7 Cambio/conversione di VSTA 52

4. Lista Macro 53

4.1 VBA: Barra degli strumenti e menù contestuale - Visualizzazione dettagli 54

4.2 VBA su sistemi a 64-Bit 58

4.3 Nozioni di base ... 58
4.3.1 Proprietà degli oggetti - PROPERTIES ... 58
4.3.2 Metodi ad oggetti - METHODS ... 59
4.3.3 Eventi di oggetti - EVENTS ... 59
4.3.4 Modello di oggetti VBA in zenon ... 60
4.3.5 Come utilizzare le macro VBA? ... 62
4.3.6 Come implementare un elemento ActiveX in zenon? ... 64
4.3.7 Accesso mediante un programma esterno ... 65
4.3.8 Funzionalità di una variabile online ... 66
4.3.9 Lista dei bit di stato ... 69
4.3.10 Lazo per selezionare elementi dinamici a Runtime ... 71
4.4 Macro nell´Editor ... 73
4.4.1 Barra degli strumenti "Lista macro" ... 73
4.4.2 Assegnazione di macro ... 75
4.5 Funzioni in zenon ... 77
4.5.1 Lanciare Macro-VBA ... 78
4.6 Sviluppare Wizard in VBA ... 80
4.6.1 Utilizzo di un Wizard ... 81
4.6.2 Struttura di un Wizard ... 81
4.6.3 Integrazione in VBA ... 82
4.6.4 Sviluppo di un Wizard ... 82
4.6.5 Aggiornare i Wizard. ... 88
4.7 Domande ricorrenti ... 88
4.7.1 Perché il pulsante rimane premuto? ... 88
4.7.2 La macro non viene eseguita al 1° clic ... 88
4.7.3 Le macro non funzionano più a RT? ... 89
4.7.4 Windows CE e VBA ... 89
4.8 Esempi ... 89
4.8.1 MouseEvents and ActiveX Control initialization ... 89
4.8.2 Visualizzazione di informazioni variabili ... 90
4.8.3 Lettura e scrittura di valori variabili ... 91
4.8.4 Lettura e scrittura di variabili e implementazione di variabili online ... 92
4.8.5 Uso multiplo della finestra di dialogo ... 94
4.8.6 Alarm – Events and ActiveX Control handling ... 96
4.8.7 Accesso ad allarmi ... 98
4.8.8 Impostare un interruttore (lavorare con variabili di processo) ... 100

5. VSTA 103

5.1 Nozioni di base 105

5.1.1 Configurazione dell'ambiente VSTA 105

5.1.2 Accesso al modello oggetti in zenon 106

5.1.3 Funzioni in zenon 107

5.1.4 Eseguire il debug di un Add-in VSTA 109

5.1.5 Events in VSTA 109

5.1.6 Backup di progetti VSTA 110

5.2 Creazione di un progetto VSTA 110

5.2.1 Progetti VSTA nell´Editor 111

5.2.2 Progetti VSTA a Runtime 112

5.2.3 Sviluppo di Wizard in VSTA 113

5.3 Esempi 113

5.3.1 Creazione di variabili nell´Editor di zenon 114

5.3.2 Scrittura di informazioni di progetto nella finestra di emissione di zenon 117

5.3.3 Lettura di variabili in zenon tramite Regular Expressions 119

5.3.4 Conversione dei colori tra.NET e zenon 122

6. Process Control Engine (PCE) 126

6.1 L'editor PCE 127

6.1.1 Il task manager 127

6.1.2 Area di editazione 128

6.1.3 Finestra di emissione 128

6.1.4 I menù dell'editor PCE 128

6.1.5 La barra dei simboli dell'editor PCE 131

6.2 Procedura 132

6.2.1 Creazione di un task 132

6.2.2 Inserimento del codice 134

6.2.3 Funzione Show PCE 135

6.2.4 Esecuzione di task 136

6.3 VBScript - Introduzione 137

6.3.1 Tipi di dato 137

6.3.2 Variabili 138

6.3.3 Costanti 141

6.3.4 Operatori 142

6.3.5 Controllo dell'esecuzione del programma 143

6.3.6 Utilizzo di cicli per la ripetizione del codice. 146
6.3.7 Tipi di procedure 151
6.3.8 Convenzioni di scrittura del codice 153

7. Object Model 158

1. Benvenuti nell'help di COPA-DATA

GUIDA GENERALE

Se non avete trovato le informazioni che cercavate o se avete dei consigli relativi al completamento di questo capitolo dell'help, potete scrivere una E-Mail all'indirizzo documentation@copadata.com.

SUPPORTO TECNICO ALLA PROGETTAZIONE

Per domande relative a progetti concreti ci si può rivolgere via E-Mail all'indirizzo support@copadata.com.

LICENZE E MODULI

Se si ha bisogno di nuovi moduli o licenze, rivolgersi ai dipendenti di COPA-DATA all'indirizzo sales@copadata.com.

2. Interfacce di programmazione

In zenon sono disponibili diverse interfacce per l'integrazione di programmi propri o per l'automazione della progettazione::

- **Lista macro (A pagina: 53) (**VBA**)**
- **VSTA (A pagina: 103)**
- **Add-In** (A pagina: 34) (Add-Ins**)**
- **Process Control Engine** (PCE
  PCE non viene più supportato a partire dalla versione 7.20 e non viene più visualizzato nella struttura ad albero dei moduli di zenon. Quando si convertono progetti che contengono task PCE da versioni precedenti alla 7.20, il nodo PCE viene visualizzato nuovamente per questi progetti. È stato sospeso il lavoro di sviluppo e di documentazione per PCE.

  Raccomandazione: utilizzare **zenon Logic** invece di PCE.

MENÙ CONTESTUALE

Voce di menù	Azione
Apri Editor-VBA	Apre l'Editor VBA.
Apri l´Editor VSTA con **Projectaddin**	Apre l'Editor VSTA.
Guida	Apre la guida online.

Info

Per informazioni sulla creazione e sull'implementazione dei controlli (ActiveX, .NET, WPF), fare riferimento al capitolo Controls.
Per informazioni sulla progettazione e sull'utilizzo dell'interfaccia SAP, vedere Interfaccia SAP.

APRIRE GLI EDITOR

ADD-IN

Per aprire l'Editor Add-In:

1. Nell´Editor di zenon, cliccare su **Extras**.
2. Selezionare la voce **Apri Editor Add-In**.

In alternativa: Premere la combinazione di tasti `Alt+F9`.

Per aprire l'Editor a Runtime:

- Nello **Startup Tool**, selezionare alla voce **Tools** il collegamento **SharpDevelop IDE**.

EDITOR VBA PER LISTA MACRO

VBA avvia lo stesso ambiente di sviluppo per **Workspace** e **progetto**.
Per aprire l´Editor di VBA:

1. Nell´Editor di zenon, passare al nodo **Scripting**.
2. Estendere questo nodo cliccando su `[+]`.
   Viene visualizzata la struttura del nodo.
3. Cliccare con il tasto destro del mouse su **Lista Macro**.
4. Nel menù contestuale, selezionare la voce **Apri Editor-VBA**.

In alternativa: Premere la combinazione di tasti `Alt+F11`

EDITOR VSTA

VSTA fornisce ambienti di sviluppo separati per **Workspace** e **progetto**. Solo uno di essi può essere utilizzato alla volta. All'avvio dell´Editor VSTA, qualsiasi altro ambiente di sviluppo VSTA già aperto verrà chiuso.

Per aprire l'Editor VSTA per il workspace:

1. Premere la combinazione di tasti `Alt+F10`.
   Viene visualizzata una finestra di dialogo che contiene un avviso.

   Viene visualizzato il codice del workspace e di tutti i progetti caricati.

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

1. Passare al nodo **Scripting**
2. Espandere la visualizzazione di questo nodo su `[+]`.
   Viene visualizzata la struttura del nodo.
3. Cliccare con il tasto destro del mouse su **VSTA**.
4. Nel menù contestuale, selezionare **Apri Editor VSTA con ProjectAddin**.
   Viene visualizzata una finestra di dialogo che contiene un avviso.

   Cliccare sul pulsante **OK** per proseguire con la progettazione VSTA.
   L'Editor VSTA si apre per il progetto attualmente caricato.

NOTA SULLE API A PARTIRE DALLA VERSIONE 7.10

Per l'utilizzo delle interfacce di programmazione di zenon, a partire da zenon 7.10 vale quanto segue:

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

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

3. Add-Ins

Add-Ins offrono la possibilità di estendere le funzionalità di zenon nell´Editor e a Runtime con diversi ambienti di sviluppo.
Per la creazione di Add-Ins sono disponibili:

- Microsoft Visual Studio (A pagina: 22)
- SharpDevelop (incluso nel pacchetto di installazione) (A pagina: 29)

Add-Ins possono essere creati con ogni linguaggio di programmazione .NET. Il supporto IDE è disponibile per i linguaggi di programmazione C# e Visual Basic.NET.

Nota: si determina quale ambiente di sviluppo viene utilizzato quando si fa clic su **Apri Editor Add-In** nelle impostazioni di zenon (menù "Extras") nella scheda **Add-Ins**.

Note:

- Tutte le liste e le interfacce supportano **IEnumerable**.
  Riferimento: https://msdn.microsoft.com/de-de/library/9eekhta0(v=vs.110).aspx (https://msdn.microsoft.com/de-de/library/9eekhta0(v=vs.110).aspx)
- **LINQ** (Language Integrated Query) può essere utilizzato.
  Riferimento: https://msdn.microsoft.com/en-us/library/mt693024.aspx (https://msdn.microsoft.com/en-us/library/mt693024.aspx)

> **Attenzione**
>
> *Agli Add-Ins si applicano le .NET Naming Guidelines.*
>
> *I nomi degli oggetti precedentemente usati per VSTA e VBA sono stati adattati. In VSTA e VBA i nomi usati fino ad ora rimangono immutati per motivi di compatibilità. Gli Add-Ins sono nominati in base alle .NET Naming Guidelines.*
>
> Una panoramica dei nomi degli oggetti si trova nella tabella **API_Naming_Conversion.xlsx**. Questa tabella si trova nel percorso di installazione di zenon, nella cartella **\HELP\AddIns**.

FUNZIONALITÀ

Add-Ins consentono, fra l´altro:

- Di realizzare strumenti di supporto per lo svolgimento del lavoro di progettazione nell´Editor (per es. Wizard).
- Di estendere il progetto a Runtime tramite l´aggiunta di ulteriori funzionalità.

Gli Add-Ins si trovano:

- Nell´Editor: nel menù **Extras**, alle voci:

- **GestioneAdd-Insper l´Editor...**
- **Avvia Wizard in Editor..**
- **Gestione servizi Editor...**

- A Runtime: nel nodo **Interfacce di programmazione- Add-Ins**

COMPATIBILITÀ

Gli Add-In di zenon sono compatibili verso l´alto e verso il basso.

COMPATIBILITÀ VERSO L'ALTO

Add-Ins sono compatibili anche con le versioni più recenti di zenon. Ad esempio, se un Add-In è stato compilato per la versione 7.60, è eseguibile anche nella versione 8.00.

COMPATIBILITÀ CON LE VERSIONI PRECEDENTI (RETROCOMPATIBILITÀ)

Add-Ins possono essere compilati per versioni precedenti di zenon. Ad esempio, è possibile creare un Add-In con zenon 8.00 per zenon 7.60.

TOOL E SUPPORTO TECNICO

Tool e supporto tecnico sono disponibili in:

- Visual Studio Developer Tools: marketplace.visualstudio.com (https://marketplace.visualstudio.com)
- COPA-DATA user forum: forum.copadata.com (https://forum.copadata.com/)

3.1 Conoscenze preliminari

Come per la creazione di altri componenti software, per lo sviluppo di Add-Ins per zenon sono necessarie alcune conoscenze pregresse.

Conoscenze preliminari necessarie

- Conoscenza dei linguaggi di programmazione C# o VB.NET (in funzione delle esigenze/complessità degli Add-Ins da sviluppare)
- Buona conoscenza di zenon.

Inoltre è raccomandabile

- Avere esperienza di lavoro con ambienti di sviluppo professionale (es. Microsoft Visual Studio).
- Utilizzo di un sistema di gestione delle versioni per gli sviluppi (TFS, SVN, GIT)

3.2 Terminologia

Termini tecnici e loro significato in relazione agli Add-Ins.

Termine	Significato
Add-In	Una raccolta di **Extensions**. È rappresentata dall´Add-in package. Un **Add-In** può contenere più **Extensions**.
Add-In Assembly	Libreria compilata (*.dll) contenente le **Extensions**.
Add-In Package	Un file ZIP (*.scadaAddIn) contenente metadati e librerie con le loro dipendenze. Un **Add-In Package** contiene una **Add-In Assembly**.
Extension	**Extensions** sono classi derivate da un'interfaccia con uno o più metodi. Sono punti di ingresso nell´**Add-In Package.** Ogni classe che rappresenta una **Extension** viene contrassegnata da un attributo .NET. Ci sono due tipi di **Extensions** (A pagina: 19), uno per l´Editor e uno per il Runtime. ▸ **Wizard Extensions** (Wizard) ▸ **Service Extensions** (Servizi)
IDE	Ambiente di sviluppo integrato. Abbreviazione per Integrated Development Environment.
Project Add-In Management	Componente per Editor e Runtime di zenon che gestisce i **Project Add-In Packages**.
Service Extension	Implementazione di una **Extension** che rappresenta un servizio (**Service**). **Servizi:** ▸ Vengono eseguiti in background. ▸ Vengono avviati automaticamente o manualmente. ▸ Rimangono in memoria finché non vengono terminati.
Wizard Extension	Implementazione di una **Extension** che rappresenta un Wizard. **Wizard:** ▸ Sono utilizzati a svolgere un determinato compito. ▸ Vengono scaricati dalla memoria di lavoro quando sono chiusi

Nota: i termini tecnici sono utilizzati solo in inglese nella documentazione relativa agli Add-Ins.

3.3 Restrizioni

Per gli Add-Ins valgono le seguenti limitazioni:

- zenon Web Client supporta l´esecuzione di Add-Ins, ma senza supporto IDE.

  Per questa ragione, il debug locale di Add-Ins non è quindi possibile in zenon Web Client; viene supportato invece un debug via strumenti di debug remoto.
- L´esecuzione di Add-Ins non è supportata in HTML Web Engine.

3.4 Creare Add-In

INFORMAZIONI GENERALI

Add-Ins vengono identificati tramite l´Add-In ID. Questo ID viene utilizzato per installazione e importazione.

L´Add-In ID consiste di un Namespace e di un ID locale: **namespace.localId**. Il Local ID identifica un Add-In, il Namespace assegna l´Add-In a un´organizzazione. Entrambe queste informazioni vengono definite tramite l´attributo .NET **Mono.AddIn.AddInAttribute**. Per garantire che il nome di un Add-In Package rimanga univoco al di là dei confini organizzativi, si utilizza il Namespace del progetto Add-In come parte dell´ID.
Esempio di un Add-In ID completo: **com.organisation.importwizard**.

L´assegnazione di Namespaces è facoltativa.
Raccomandazione: utilizzare Namespaces. Questo garantisce l'unicità globale al di là dei confini organizzativi. Si raccomanda di usare l'URL della propria organizzazione in ordine inverso.
Esempio: per **www.example.com** utilizzare **com.example** come Namespace.
I Namespaces possono essere ulteriormente suddivisi all´interno di un´organizzazione, per esempio per team, reparti o progetti.

Per la creazione di Add-Ins sono disponibili:

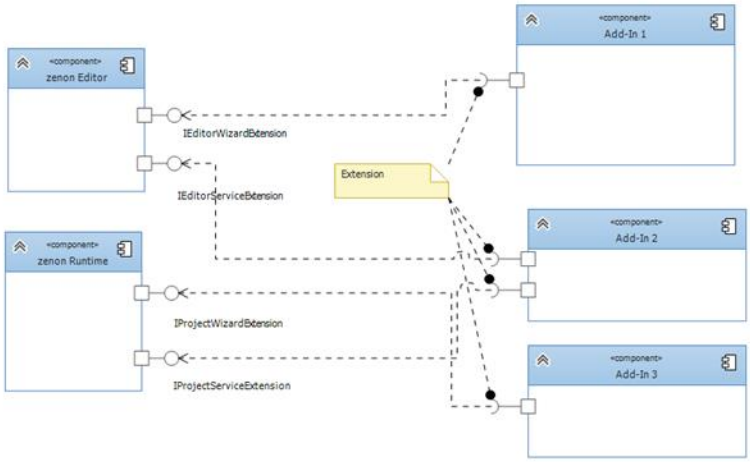
- Microsoft Visual Studio (A pagina: 22)
- SharpDevelop (incluso nel pacchetto di installazione) (A pagina: 29)

Add-Ins possono essere creati con ogni linguaggio di programmazione .NET. Il supporto IDE è disponibile per i linguaggi di programmazione C# e Visual Basic.NET.

Nota: si determina quale ambiente di sviluppo viene utilizzato quando si fa clic su **Apri Editor Add-In** nelle impostazioni di zenon (menù “Extras”) nella scheda **Add-Ins**.

3.4.1 Nozioni di base

Sono disponibili diverse Extensions per l´Editor di zenon e per il Runtime di zenon. A seconda dell´applicazione, è possibile selezionare una Extension appropriata.



Tutte le interfacce necessarie per lo sviluppo di Add-Ins sono definite in Assembly **Scada.AddIn.Contracts**.

Un Add-In Package può contenere una o più Extensions .

TIPI DI EXTENSIONS - INTERFACCE

Si distingue fra le seguenti Extensions :

EDITOR WIZARD EXTENSION - IEDITORWIZARDEXTENSION (A PAGINA: 19)

- L´esecuzione viene inizializzata dall´utente mediante la finestra di dialogo **Extras -> Avvia Wizards in Editor ...** nell´Editor di zenon.
- Implementa il metodo **Run** per eseguire una Wizard Extension nell´Editor di zenon.

EDITOR SERVICE EXTENSION - IEDITORSERVICEEXTENSION (A PAGINA: 19)

- L´esecuzione avviene automaticamente all´avvio dell´Editor, in dipendenza da **DefaultStartMode**.

  Nota: Per l´avvio automatico bisogna impostare nell´**Extension** l´attributo `DefaultStartMode=DefaultStartupModes.Auto`. La modalità di avvio può essere modificata in un secondo momento nella finestra di dialogo **Gestione servizi Editor** (A pagina: 41).
- la gestione avviene tramite la finestra di dialogo **Extras -> Gestione servizi Editor...** nell´Editor di zenon.

- Implementa i metodi **Start** e **Stop** per eseguire una Service Extension nell´Editor di zenon.

PROJECT WIZARD EXTENSION - IPROJECTWIZARDEXTENSION (A PAGINA: 20)

- L´esecuzione viene inizializzata dall´utente tramite la funzione di zenon **Avvia Project Wizard Extension**.
- Implementa il metodo **Run** per eseguire una Wizard Extension nel Runtime di zenon.

PROJECT SERVICE EXTENSION - IPROJECTSERVICEEXTENSION (A PAGINA: 21)

- L´esecuzione avviene automaticamente all´avvio del progetto a Runtime, in dipendenza da **DefaultStartMode**.

  Nota: Per l´avvio automatico bisogna impostare nell´**Extension** l´attributo `DefaultStartMode=DefaultStartupModes.Auto`. La modalità di avvio può essere modificata in un secondo momento nella finestra di dialogo **Gestione servizi Editor** (A pagina: 41).
- la gestione avviene tramite la funzione di zenon **Visualizza dialogo "Gestione dei Servizi Runtime"**.
- Implementa i metodi **Start** e **Stop** per eseguire una Service Extension nel Runtime di zenon.

ATTRIBUTI DELL'ESTENSIONE

I seguenti metadati sono definiti mediante l´attributo .NET chiamato **AddInExtension**.

Property	Descrizione	IEditorWizardExtension	IEditorServiceExtension	IProjectWizardExtension	IProjectServiceExtension
Name	Nome di visualizzazione in finestre di dialogo. Info: corrisponde al nome del Wizard o del servizio in zenon.	**X**	**X**	**X**	**X**
Description	Descrizione per la visualizzazione nelle finestre di dialogo. Info: corrisponde alla descrizione del Wizard o del servizio in zenon.	**X**	**X**	**X**	**X**
Category	Categoria del raggruppamento in finestre di dialogo. Info: è usata per la categorizzazione di Wizard nella finestra di dialogo **Avvia Wizards in Editor...** nell´Editor di zenon.	**X**	**--**	**X**	**--**
DefaultStartMode	Descrive il comportamento di avvio predefinito dei Service Add-In Extensions dopo l'installazione. ▸ `Manual`: Extension deve essere avviata manualmente dall´utente. ▸ `Auto`: viene avviato quando viene caricato il workspace nell´Editor o il progetto a Runtime. Nessuna indicazione: `Manual` viene utilizzato. Info: corrisponde alla modalità di avvio del servizio in zenon.	**--**	**X**	**--**	**X**

Legenda:

- **X**: vero
- **--**: non vero

ATTRIBUTE DELL´ADD-IN

Per un´Add-In Assembly sono necessari più attributi .NET, che vengono definiti nel file **AddInInfo.cs**.

I più importanti per l´utilizzo in zenon sono:

Nome attributo	Obbligatorio	Descrizione	Esempi
`Addin`	**X**	ID e versione dell´Add-In. Info: la versione corrisponde alla colonna **Versione** nella finestra di dialogo di gestione degli Add-In nell´Editor di zenon.	Esempio 1: [assembly: Addin("AddInIdentifier", "1.0", Namespace="com.example")] Esempio 2: [assembly: Addin("MultitouchAddIn", "1.0")] Esempio 3: [assembly: Addin("ProjectCreation", "1.0")]
`AddinDependency`	**X**	Definisce una dipendenza dal sistema zenon. Questo valore è costante e necessario perché zenon possa usare l´Add-In.	[assembly: AddinDependency("scada", "1.0")]
`AddinName`	--	Nome dell´Add-In. Info: corrisponde alla colonna **Nome** nella finestra di dialogo di gestione degli Add-In nell´Editor di zenon.	[assembly: AddinName("Editor Add-In Demo")]
`AddinDescription`	--	Descrizione dell´Add-In. Info: corrisponde alla colonna **Descrizione** nella finestra di dialogo di gestione degli Add-In nell´Editor di zenon.	[assembly: AddinDescription("Demonstrates an Add-In with a Wizard and a Service extension.")]

Legenda:

- **X**: vero
- **--**: non vero

3.4.2 Tipi di Extensions

Wizard nell´Editor - Editor Wizard Extension

Editor Wizard Extensions vengono utilizzate per realizzare Wizard o altre operazioni API gestite dall´utente nell´Editor.

- L´esecuzione viene inizializzata dall´utente mediante la finestra di dialogo **Extras -> Avvia Wizards in Editor ...** nell´Editor di zenon.
- Implementa il metodo **Run** per eseguire una Wizard Extension nell´Editor di zenon.

ESEMPIO DI CODICE (C#)

```
[AddInExtension("Wizard Extension Name", "Description of Wizard Extension", "Category of Wizard Extension")]

public class WizardExtension : IEditorWizardExtension

{

 public void Run(IEditorApplication context, IBehavior behavior)

 {

 // Implement feature here...

 }

}
```

Wizard nell´Editor - Editor Service Extension

Editor Service Extensions vengono utilizzate per realizzare operazioni API nell´Editor in modo automatico.

- L´esecuzione avviene automaticamente all´avvio dell´Editor, in dipendenza da **DefaultStartMode**.

  Nota: Per l´avvio automatico bisogna impostare nell´**Extension** l´attributo `DefaultStartMode=DefaultStartupModes.Auto`. La modalità di avvio può essere modificata in un secondo momento nella finestra di dialogo **Gestione servizi Editor** (A pagina: 41).
- la gestione avviene tramite la finestra di dialogo **Extras -> Gestione servizi Editor...** nell´Editor di zenon.
- Implementa i metodi **Start** e **Stop** per eseguire una Service Extension nell´Editor di zenon.

ESEMPIO DI CODICE (C#)

```
[AddInExtension("Service Extension Name", "Description of Service Extension", "Category of Service Extension", DefaultStartMode=DefaultStartupModes.Auto)]

public class ServiceExtension : IEditorServiceExtension

{

 public void Start(IEditorApplication context, IBehavior behavior)

 {

 // Startup code here

 }

 public void Stop()

 {

 // Stop code here

 }

}
```

Wizard a Runtime - Project Wizard Extension

Project Wizard Extensions vengono utilizzate per realizzare Wizard o altre operazioni API gestite dall´utente nel Runtime.

- L´esecuzione viene inizializzata dall´utente tramite la funzione di zenon **Avvia Project Wizard Extension**.
- Implementa il metodo **Run** per eseguire una Wizard Extension nel Runtime di zenon.

ESEMPIO DI CODICE (C#)

```
[AddInExtension("Wizard Extension Name", "Description of Wizard Extension", "Category of Wizard Extension")]

public class WizardExtension : IProjectWizardExtension

{

 public void Run(IProject context, IBehavior behavior)

 {

 // Implement feature here...

 }

}
```

Servizi a Runtime - Project Service Extension

Runtime Service Extensions vengono utilizzate per realizzare operazioni di background a Runtime.

- L´esecuzione avviene automaticamente all´avvio del progetto a Runtime, in dipendenza da **DefaultStartMode**.

  Nota: Per l´avvio automatico bisogna impostare nell´**Extension** l´attributo `DefaultStartMode=DefaultStartupModes.Auto`. La modalità di avvio può essere modificata in un secondo momento nella finestra di dialogo **Gestione servizi Editor** (A pagina: 41).
- la gestione avviene tramite la funzione di zenon **Visualizza dialogo "Gestione dei Servizi Runtime"**.
- Implementa i metodi **Start** e **Stop** per eseguire una Service Extension nel Runtime di zenon.

ESEMPIO DI CODICE (C#)

```
[AddInExtension("Service Extension Name", "Description of Service Extension", "Category of Service Extension", DefaultStartMode=DefaultStartupModes.Auto)]

public class ServiceExtension : IProjectServiceExtension

{

 public void Start(IProject context, IBehavior behavior)

 {

 // Startup code here

 }

 public void Stop()

 {

 // Stop code here

 }

}
```

3.4.3 Contenuto di Add-In Packages

Add-In Packages sono file ZIP con l´estensione di nome file **.scadaAddIn**. Contengono file binari e file di metadati. Ci devono essere file binari che vengono salvati come file ZIP a parte nell´Add-In Package.
I Packages possono essere creati con l´**Add-In Utility** (A pagina: 32). Con gli ambienti di sviluppo SharpDevelop e Microsoft Visual Studio, i Packages vengono creati automaticamente durante la compilazione.

Raccomandazione: non modificare i file manualmente, ma utilizzare sempre l´**Add-In Utility**.

CONTENUTO DELL´ADD-IN PACKAGE

L´Add-In Package contiene:

- **Content.xml**: file XML di metadati con contenuti del Package.
- **Binary.zip**: file ZIP con i file binari.

FILE XML

Il contenuto di un Add-In Package è elencato in un proprio file XML (**Content.xml**).

FILE BINARI

I file binari vengono salvati in un proprio file ZIP. Questo file deve contenere tutti i componenti necessari per eseguire l´Add-In su un altro sistema. Sono necessari riferimenti diretti e indiretti a.NET Assemblies. Devono essere incluse solo Assemblies esterne. Sono da considerare Assemblies esterne quelle che non sono parte della .NET Framework Class Library , o che vengono installate nel corso della installazione di zenon.

Attenzione: assicurarsi di avere tutte le licenze necessarie.

Nota: .NET Assemblies caricate dinamicamente non sono inclusi nei metadati di .NET Assemblies.

Info

COPA-DATA non si assume alcuna responsabilità e non fornisce supporto tecnico per Libraries esterne.

3.4.4 Microsoft Visual Studio

Questa sezione fornisce informazioni sulla creazione e la gestione di Add-Ins con Microsoft Visual Studio. Gli Add-In possono essere creati mediante:

- MS Visual Studio 2015 (Community, Professional, Enterprise)
- MS Visual Studio 2017 (Community, Professional, Enterprise)

COPA-DATA Developer Tools

COPA-DATA Developer Tools forniscono gli strumenti per creare Add-Ins per l´Editor e il Runtime di zenon, come Wizard o servizi.

ZENON ADD-INS

COPA-DATA **Developer Tools** consentono agli utenti di creare e eseguire il debug di zenon Add-Ins in modo rapido e semplice mediante **Microsoft Visual Studio**. Forniscono template di progetto per l´Editor e il Runtime di zenon. Per entrambe le applicazioni sono disponibili template per servizi e Wizard (extensions).

COPA-DATA **Developer Tools** supportano:

- **C#**
- **VB.NET**

Con COPA-DATA **Developer Tools** vengono creati `Add-In packages` durante la creazione di Add-Ins. Il deployment e il debug dei Add-In packages sono integrati. Questo significa: Quando si avvia il debug, vengono distribuiti gli Add-Ins e il Debugging Tool viene automaticamente collegato all´Editor o al Runtime di zenon.

REQUISITI

Per utilizzare COPA-DATA **Developer Tools**, è necessario::

- **Visual Studio 2015 (Community, Professional or Enterprise)**o superiore
- **zenon 7.60**

NOVITÀ NELLA VERSIONE 08:00

Novità e modifiche relative ai **COPA-DATA Developer Tools** in zenon 8.00:

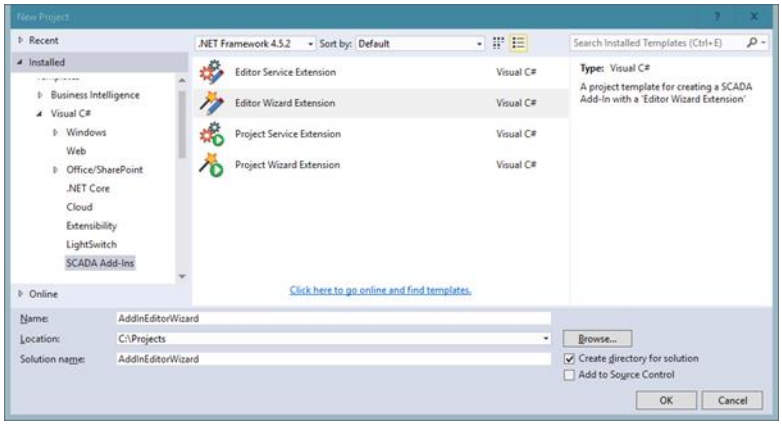
- È possibile creare Add-Ins per zenon a partire dalla versione 7.60.
- Supporto di **Microsfot Visual Studio 2017** (Community, Professional, Enterprise).
- Fornisce template di progetto per Wizard WPF per zenon Runtime e zenon Editor.

PRIMI PASSI CON COPA-DATA DEVELOPER TOOLS

Sarà facile creare, eseguire il debug e distribuire Wizard e servizi utilizzando COPA-DATA Developer Tools.

CREARE UN PROGETTO

Per creare un nuovo progetto, utilizzare la finestra di dialogo **New Project**.

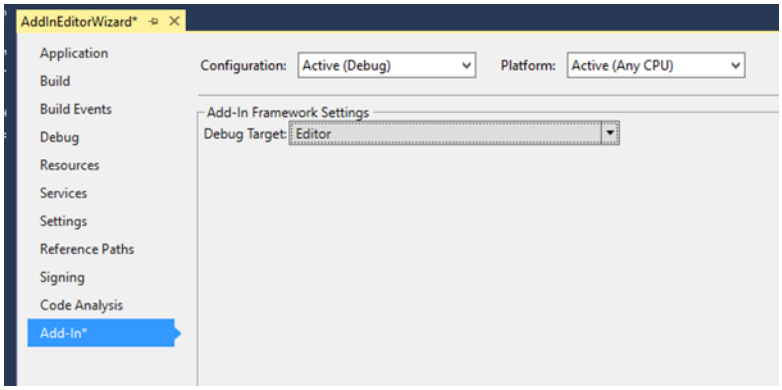


Dopo aver creato il progetto, viene creata una classe con Extension vuota:

```
/// <summary>
/// Description of Editor Wizard Extension.
/// </summary>
[AddInExtension("Your Editor Wizard Extension Name", "Your Editor Wizard Extension Description", "Your Editor Wizard Extension Category")]
~references
public class EditorWizardExtension : IEditorWizardExtension
{
  #region IEditorWizardExtension implementation

  ~references
  public void Run(IEditorApplication context, IBehavior behavior)
  {
    // enter your code for which should be executed at the SCADA Editor Wizard Dialog

  }

  #endregion
}
```

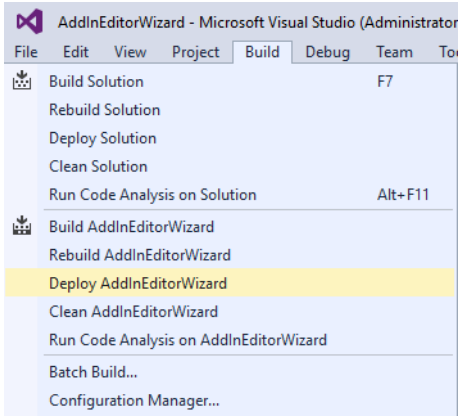
DEBUG

Per eseguire il debug, selezionare una destinazione nelle proprietà del progetto:



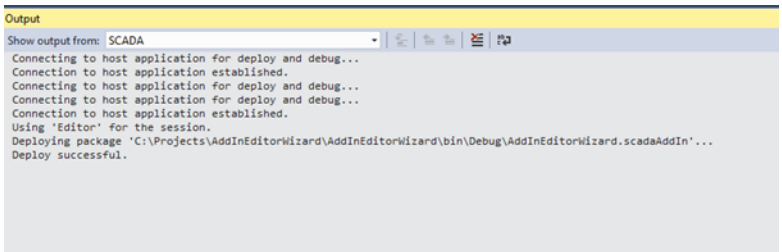
DEPLOYMENT

Per eseguire la distribuzione, utilizzare il comando **Deploy** nel menù **Build**:



Nota: se il debug viene avviato direttamente, anche la distribuzione viene avviata automaticamente.

Durante distribuzione e debug, la categoria **SCADA** visualizza informazioni dettagliate nella finestra di emissione.



DOCUMENTAZIONE E SUPPORTO TECNICO

La documentazione per COPA-DATA **Developer Tools** è disponibile in Visual Studio come guida integrata. È sufficiente premere il `tasto F1` all'interno dell'applicazione per ottenere una guida sensibile al contesto. È possibile ottenere supporto anche sul sito web di COPA-DATA:

- Per sapere le ultime notizie e per ottenere supporto da parte di altri utenti, utilizzare il forum: https://forum.copadata.com/forumdisplay.php?45-Programming-Interface-API (https://forum.copadata.com/forumdisplay.php?45-Programming-Interface-API)
- Per ottenere supporto da parte di COPA-DATA, si prega di contattare il supporto tecnico: https://www.copadata.com/en/support-services/ (https://www.copadata.com/en/support-services/)

Creare un Add-In Package con Visual Studio

Per creare un Add-In Package con Visual Studio:

1. Avviare anzitutto Visual Studio.

   Oppure: Nell`Editor di zenon, selezionare il menù **Extras ->Apri Editor Add-In....**
   Suggerimento: nel menù Extras => Impostazioni=> Add-Ins, verificare che Visual Studio sia l'ambiente di sviluppo predefinito.

2. Installare i Visual Studio Developer Tools.
   Nota: li si può scaricare da Visual Studio Marketplace https://marketplace.visualstudio.com/.

3. Selezionare **File -> New Project.**

4. Alla voce **Templates -> Visual C# -> SCADA Add-Ins** o **Templates -> Visual Basic -> SCADA Add-Ins** selezionare un template appropriato.

5. Creare un nuovo progetto basato su questo:

   - Editor Service Extension (A pagina: 19)
   - Editor Wizard Extension (A pagina: 19)
   - Project Service Extension (A pagina: 21)
   - Project Wizard Extension (A pagina: 20)

   Nota: tutti gli esempi utilizzati in questo capitolo sono basati su C#.

6. Aggiungere il proprio codice, i propri riferimenti ecc.

7. Creare il progetto.

8. L´Add-In Package con l´estensione ***.scadaAddIn** viene creato automaticamente nella cartella di output del progetto (a seconda della configurazione: `...bin\Debug\` o `...bin\Release\`).

9. Importare l´Add-In Package nell´Editor; a questo proposito, si vedano i capitoli:

   - Installare e gestire Add-Ins per l´Editor (A pagina: 39)
   - Installare e gestire Add-Ins per il Runtime (A pagina: 44)

IMPOSTARE LA VERSIONE DI ZENON

Per ogni Add-In si deve definire per quale versione di zenon deve essere compilato.
Per farlo:

1. Nell'ambiente di programmazione, selezionare **Add-In Framework Settings**.

2. Aprire la proprietà **Minimal Scada Version**.

3. Selezionare la versione desiderata nel menù a tendina.

   `Latest Version`: l´Add-In viene compilato per la versione di zenon attualmente registrata.

Per verificare nell´Editor di zenon per quale versione si sta compilando:

1. Cliccare su Add-In.

2. Aprire il gruppo di proprietà **Generale**.

3. Verificare la proprietà **Versione minima**.

Eseguire il deploy e il debug di Add-Ins

Add-Ins Si può eseguire il debug degli Add-Ins sia nell´Editor che a Runtime.

Nota: quando si interrompe il debug, è possibile:

- Separare il Debugger dal processo (Detach): l'Editor o il Runtime continuano a funzionare.
- Terminare il processo: l'Editor o il Runtime vengono terminati.

ESEGUIRE IL DEBUG NELL´EDITOR

Procedura per eseguire il debug di un Add-In nell´Editor:

1. Aprire l´Editor di zenon
2. Avviare il processo di debug in Visual Studio premendo il tasto `F5`, oppure selezionando **Debug -> Start Debugging**.
3. Il progetto Add-In viene compilato e ne viene eseguito il deploy nell´Editor.

   Nota: l´Add-In viene copiato nella cartella `%ProgramData%\COPA-DATA\zenon800\EditorAddInCache`.

   Per rimuoverlo dall´Editor, procedere come descritto nel capitolo Installare e gestire Add-Ins per l´Editor (A pagina: 39).
4. Eseguire il debug dell´Add-In.

> **Info**
>
> *In alternativa al metodo descritto sopra, è anche possibile eseguire il debug degli Add-Ins mediante **Debug -> Attach to Process**. In questo caso, connettersi dal progetto Add-In in Visual Studio al processo nell´Editor di zenon.*
>
> *Tipo di Debugger raccomandato (**Debugger Type**): `Managed mit CLR 4.0`*

ESEGUIRE IL DEBUG A RUNTIME

Procedura per eseguire il debug di un Add-In a Runtime:

1. Compilare l´Add-In e importarlo come descritto nel capitolo Installare e gestire Add-Ins per il Runtime (A pagina: 44).
2. Avviare il Runtime di zenon.
3. In Visual Studio, alla voce **Debug Target**, selezionare il progetto.
   L'istanza corrispondente deve essere in esecuzione.
4. Avviare il processo di debug premendo il tasto `F5`, oppure selezionando **Debug -> Run**.

5. Viene eseguito il debug e deploy del progetto Add-In.
   Assicurarsi che il Wizard o il servizio non siano in esecuzione in caso di deploy ripetuti.
6. Eseguire il debug dell´Add-In.

> **Info**
>
> *In alternativa ai metodi descritti sopra, è possibile eseguire il debug degli Add-Ins mediante **Debug -> Attach to Process**. Selezionare il processo zenon Runtime (**Zenrt32.exe**) e cliccare su **Attach**.*

Funzione di ricerca e accesso alla documentazione di zenon API

Da Visual Studio, è possibile utilizzare il **tasto F1** per passare direttamente alla documentazione di zenon quando il cursore si trova su contenuti di Add-In Framework.

Se il cursore si trova su un'altra keyword, si apre la documentazione MSDN.

L'accesso alla documentazione di zenon API è supportato nelle seguenti versioni di zenon:

- 7.60
- 8.00

E per le seguenti versioni di Visual Studio:

- 2015
- 2017

Quando si apre la documentazione API, viene presa in considerazione la versione aperta nella Visual Studio Solution.

Se non viene rilevata nessuna versione di zenon, viene automaticamente aperta la documentazione zenon API più recente.

Per la rispettiva versione di zenon sono disponibili le seguenti opzioni di immissione:

- Documentazione API **Add-In**
- Documentazione API **Com**

È anche possibile cercare determinati contenuti della documentazione di zenon API in Visual Studio:

- Quando si immette un termine di ricerca nell´apposita barra dell'interfaccia di Visual Studio, viene avviata una ricerca del contenuto nella documentazione di Add-In Framework e nella documentazione delle API Com:
- Quando si inserisce **@zenoncom** e il termine di ricerca nell´apposita barra dell'interfaccia di Visual Studio, viene avviata una ricerca del contenuto nella documentazione Com.

  Esempio: **@zenoncom** Allarme

Immettere il termine desiderato nella barra di ricerca e confermare la selezione con il **tasto Invio**.

Si può tornare direttamente dal risultato della ricerca alla documentazione di zenon API.

3.4.5 SharpDevelop

SharpDevelop (anche #develop) è un ambiente di sviluppo open source basato sulla piattaforma Microsoft .NET. **SharpDevelop** è incluso nel pacchetto di zenon e viene installato durante l´installazione di zenon.
Nota: la versione più recente e la documentazione ufficiale di **SharpDevelop** sono disponibili online al seguente indirizzo: www.icsharpcode.net/opensource/sd/ (http://www.icsharpcode.net/opensource/sd/).

Creare un Add-In Package con SharpDevelop

Per creare un Add-In Package con SharpDevelop:

1. Avviare **SharpDevelop**; per farlo:
   - Nell`Editor di zenon, selezionare il menù **Extras -> Apri Editor Add-In...**, oppure
   - Nello **Startup Tool**, selezionare alla voce **Tools** il collegamento **SharpDevelop IDE**.

   Suggerimento: nel menù Extras => Impostazioni=> Add-Ins, verificare che SharpDevelop sia l'ambiente di sviluppo predefinito.
2. Selezionare **File -> New Solution.**
3. Alla voce **C# -> SCADA Add-Ins** o **VB -> SCADA Add-Ins** selezionare un template appropriato.
4. Creare un nuovo progetto basato su questo:
   - Editor Service Extension (A pagina: 19)
   - Editor Wizard Extension (A pagina: 19)
   - Project Service Extension (A pagina: 21)
   - Project Wizard Extension (A pagina: 20)

   Nota: tutti gli esempi di questo capitolo della guida sono basati su C#.
5. Aggiungere il proprio codice, i propri riferimenti ecc.
6. Compilare il progetto.
   Verrà creato l´Add-In Package con il suffisso file ***.scadaAddIn** nella cartella di output del progetto.
   Il nome di questa cartella è (a seconda di quanto definito in fase di configurazione) `...bin\Debug\` o `...bin\Release\`.
7. Importare l´Add-In Package nell´Editor; a questo proposito, si vedano i capitoli:

- Installare e gestire Add-Ins per l´Editor (A pagina: 39)
- Installare e gestire Add-Ins per il Runtime (A pagina: 44)

IMPOSTARE LA VERSIONE DI ZENON

Per ogni Add-In si deve definire per quale versione di zenon deve essere compilato.
Per farlo:

1. Nell'ambiente di programmazione, selezionare **Add-In Framework Settings**.
2. Aprire la proprietà **Minimal Scada Version**.
3. Selezionare la versione desiderata nel menù a tendina.

   `Latest Version`: l´Add-In viene compilato per la versione di zenon attualmente registrata.

Per verificare nell´Editor di zenon per quale versione si sta compilando:

1. Cliccare su Add-In.
2. Aprire il gruppo di proprietà **Generale**.

Verificare la proprietà **Versione minima**.

Eseguire il deploy e il debug di Add-Ins

Add-Ins Si può eseguire il debug degli Add-Ins sia nell´Editor che a Runtime.

Nota: quando si interrompe il debug, è possibile:

- Separare il Debugger dal processo (Detach): l'Editor o il Runtime continuano a funzionare.
- Terminare il processo: l'Editor o il Runtime vengono terminati.

ESEGUIRE IL DEBUG NELL´EDITOR

Per eseguire il debug di un Add-In nell´Editor:

1. Aprire l´Editor di zenon
2. Avviare **SharpDevelop**; per farlo:
   - Nell`Editor di zenon, selezionare il menù **Extras -> Apri Editor Add-In...**,
     oppure
   - Nello **Startup Tool**, selezionare alla voce **Tools** il collegamento **SharpDevelop IDE**.

3. Aprire il progetto.
4. In SharpDevelop, passare alla scheda **Add-In** delle proprietà del progetto.
5. Alla voce **Debug Target**. selezionare `Editor`.
6. Avviare il processo di debug premendo il tasto `F5`, oppure selezionando **Debug -> Run**.
7. Il progetto Add-In viene compilato e ne viene eseguito il deploy nell´Editor.

   Nota: l´Add-In viene copiato nella cartella `%ProgramData%\COPA-DATA\zenon800\EditorAddInCache`.

   Per rimuoverlo dall´Editor, procedere come descritto nel capitolo Installare e gestire Add-Ins per l´Editor (A pagina: 39).
   Se c´è già un Add-In con lo stesso ID, esso viene sostituito.
8. Eseguire il debug dell´Add-In.

ESEGUIRE IL DEBUG A RUNTIME

A Runtime, il debug viene eseguito separatamente per ogni progetto.
Per eseguire il debug di un Add-In a Runtime:

1. Avviare il Runtime di zenon.
2. Avviare **SharpDevelop.**
   Per farlo, in zenon **Startup Tool**, selezionare alla voce **Tools** il collegamento **SharpDevelop IDE**.
3. Aprire il progetto.
4. In SharpDevelop, passare alla scheda **Add-In** delle proprietà del progetto.
5. Alla voce **Debug Target**, selezionare il progetto a Runtime nel quale si vuole eseguire il debug dell´Add-In
   L'istanza corrispondente deve essere in esecuzione.
6. Avviare il processo di debug premendo il tasto `F5`, oppure selezionando **Debug -> Run**.
7. Il progetto Add-In verrà compilato e ne verrà eseguito il deploy nel progetto zenon selezionato.
   In caso di deploy ripetuto, assicurarsi che il Wizard o il servizio non siano in esecuzione.
   Attenzione: l´Add-In viene copiato temporaneamente in questa cartella:
   `...\RT\FILES\zenon\system\AddInCache\...`
   Dopo che il Runtime è terminato, viene poi rimosso da questa cartella.
   Per installare un Add-In in modo duraturo, importarlo come descritto nel capitolo Installare e gestire Add-Ins per il Runtime (A pagina: 44).
8. Eseguire il debug dell´Add-In.

> **Info**
>
> *In alternativa ai metodi descritti sopra, è possibile eseguire il debug degli Add-Ins mediante **Debug -> Attach to Process**. In questo caso, connettersi dal progetto Add-In in SharpDevelop al processo nell´Editor o nel Runtime di zenon.*

3.4.6 Gestione del codice sorgente

Per la gestione del codice sorgente si consiglia l'uso di un sistema di controllo della versione (come TFS, GIT o SVN).

> **Attenzione**
>
> *A differenza di VSTA e VBA, il Source Code di Add-Ins non viene salvato e gestito nel progetto di zenon.*

3.4.7 Analisi di Add-In e Packaging Utility (AddInUtility)

Il tool **AddInUtility** viene aperto automaticamente quando si compilano Add-In e svolge i seguenti compiti:

- Analisi delle dipendenze.
- Creazione di Add-Packages Add-Ins

Un pacchetto contiene l´Add-In Assembly e un file XML con l´estensione ***.scadaAddin**, che contiene a sua volta i metadati per l´Add-In.

Strumento riga di comando - AddInUtility.exe

AddInUtility crea package Add-Ins tramite la riga di comando.

Sintassi: **AddInUtility.exe [-[shortTerm] or [/ or --][longTerm] [argument value]]** ...

- Gli argomenti non sono case sensitive.
- I valori non sono case sensitive se contrassegnati come tali.
- Gli argomenti possono essere specificati in qualsiasi ordine.

Argomento	Descrizione
--action **(-a)**	Azione da eseguire. Valori possibili: ▸ `BuildPackage` ▸ `GetPackageInfo` Campo obbligatorio.
--path **(-p)**	Specificazione del percorso. Informazioni richieste per: ▸ **action** = `BuildPackage`: percorso del file Add-In Assembly. ▸ **action** = `GetPackageInfo`: percorso del fileAdd-In Package. Campo obbligatorio.
--targetDir **(-t)**	Cartella di destinazione. Informazioni richieste per: ▸ **action** = `BuildPackage`: Percorso della cartella in cui viene creato l´Add-In Package. Default: cartella corrente

Esempi

- Crea un Add-In Package (argomenti scritti per intero):
  AddInUtility.exe --action BuildPackage --path C:\Addin.dll --targetDir C:\Directory
- Crea un Add-In Package (argomenti scritti in forma abbreviata):
  AddInUtility.exe -a BuildPackage -p C:\Addin.dll -t C:\Directory
- Restituisce informazioni sull´Add-In Package specificato:
  AddInUtility.exe -a GetPackageInfo -p C:\Directory\Addin.scadaAddIn

3.4.8 Comportamento durante il reload

zenon mette a disposizione una funzionalità di reload per i progetti che supportano a Runtime Hot-Plugging o l´aggiornamento dei file di progetto modificati. In base alle funzioni attivate degli Add-Ins, la sequenza delle attività per VSTA e VBA è la seguente:

1. Pre-reload VSTA
2. Stop of Service Extensions
3. Reload VBA
4. Start of Service Extensions
5. Post-reload VSTA

Add-Ins si comportano come VSTA e mettono a disposizione una funzione Pre-reload- e una funzione Post-reload-. Queste funzioni vengono inizializzate dopo il Pre-reload di VSTA e prima del reload di VSTA e utilizzate come Wrapper per la funzione `reload` di VBA. Un ulteriore processo di reload viene ritardato fino al completamento dell'esecuzione delle Wizard Extensions.
Suggerimento: se si desidera utilizzare la funzionalità di reload, evitare Wizard-Extensions che bloccano o che sono eseguite per lungo tempo.

Quando si esegue un nuovo reload, tutte le estensioni servizi in esecuzione vengono riavviate. Le nuove estensioni servizi installate vengono avviate dopo il riavvio se come **DefaultStartMode** è stato impostato `automatico`.

3.4.9 Isolamento

Tutti gli Add-Ins vengono eseguiti in un'area di memoria isolata (App-Domain). In questo modo è possibile caricare in parallelo versioni diverse delle stesse Assemblies.

3.5 Utilizzo di Add-Ins nell´Editor e a Runtime

Add-Ins aiutano a svolgere operazioni di progettazione nell´Editor di zenon ed estendono le funzionalità di zenon a Runtime.

Gli Add-Ins si trovano:

- Nell´Editor: nel menù **Extras**, alle voci:
  - **GestioneAdd-Insper l´Editor...**
  - **Avvia Wizard in Editor..**
  - **Gestione servizi Editor...**
- A Runtime: nel nodo **Interfacce di programmazione- Add-Ins**

ADD-INS PER EDITOR E RUNTIME

- Add-Ins per l´Editor (A pagina: 38):
  Gli Add-Ins devono prima essere importati e installati e possono essere poi utilizzati nell´Editor.
  La loro gestione (A pagina: 39) avviene tramite le voci del menù **Extras**.
- Add-Ins per il Runtime (A pagina: 43):
  Gli Add-Ins devono essere prima importati; quando viene avviato il Runtime, vengono attivati automaticamente e possono poi essere utilizzati.
  Vengono gestiti (A pagina: 44) nella visualizzazione dettagli del nodo **Interfacce di programmazione** - **Add-Ins** nella struttura ad albero del progetto.

Add-Ins sono disponibili in forma di:

- Wizard: viene avviato dall´utente e terminato al termine dell'operazione.
  Per informazioni dettagliate sui Wizard forniti con zenon, fare riferimento al manuale **Wizard**.
- Servizio: può essere avviato automaticamente o manualmente e viene eseguito in background.

Add-Ins possono:

- Essere forniti con zenon.
- Essere sviluppati da system integrators che li forniscono poi ai loro clienti.
- Essere sviluppati dai clienti stessi.

PROPRIETÀ

Se si seleziona un Add-In nell´Editor, le sue proprietà vengono visualizzate nel nodo **Generale**:

- **Nome**: nome dell´Add-In.
- **Descrizione**: descrizione dell´Add-In.
- **Versione**: numero di versione dell´Add-In.
- **Codice sorgente**: indica se il codice sorgente è incluso nel package.
- **Versione minima**: versione zenon più bassa in cui è possibile utilizzare Add-In. Quest´ultima viene impostata quando si crea l´Add-In con:
  - **Microsoft Visual Studio** (A pagina: 25)
  - **SharpDevelop** **(A pagina: 29)**

Queste proprietà non possono essere modificate nell'Editor. Sono solo a scopo informativo.
Le stesse proprietà vengono visualizzate anche nella visualizzazione dettagli per tutti gli Add-Ins.

FILTRARE E ORDINARE LA LISTA DEGLI ADD-INS E DEI SERVIZI

Per filtrare una lista nella visualizzazione dettagli o in una finestra di dialogo:

1. Cliccare sul campo "Testo filtro" della colonna desiderata.
2. Digitare i criteri di filtro.
   Si possono usare placeholder.
   - ?: placeholder per un carattere qualsiasi.
   - *: placeholder per un numero qualsiasi di caratteri.
3. Si possono combinare più filtri.

Nota: per rimuovere tutti i filtri, cliccare nella barra degli strumenti sul simbolo corrispondente.

Per ordinare la lista:

1. Cliccare sull´intestazione della colonna in base alla quale deve essere eseguito l´ordinamento.

Il simbolo del binocolo indica la colonna in base alla quale viene eseguito l´ordinamento della lista.

La lista viene ordinata sulla base della colonna sulla cui intestazione si è cliccato.

Una freccia indica se l´ordinamento è crescente o decrescente.

2. Per modificare la successione di ordinamento, cliccare di nuovo sul titolo della colonna.

BACKUP E RIPRISTINO

Quanto segue si applica al backup e al ripristino di Add-Ins:

- Non viene creato un backup degli Add-Ins dell´Editor quando si esegue il backup del workspace.

  Nota: gli Add-Ins installati nell´Editor si trovano nella cartella `%ProgramData%\COPA-DATA\zenon800\EditorAddInStore`. Se necessario, è possibile eseguire un backup manuale.

- Add-In Packages di progetti: ne viene eseguito il backup durante la creazione dei backup dei progetti e del workspace e ripristinati durante la loro lettura.

LOG DELLE MODIFICHE

Se è stato attivato il Log delle modifiche per il progetto (nodo “Log delle modifiche” delle proprietà di progetto), vengono generati inserimenti corrispondenti per lo storico delle modifiche durante l'installazione e la disinstallazione Add-In Packages.

3.5.1 Come attivare Add-Ins in zenon

L´utilizzo di **Add-Ins** viene attivato nel file **zenon6.ini**. In questo file deve esistere la seguente voce:

[AddIns]

ON=1

Questa voce è abilitata per impostazione predefinita dopo l´installazione di zenon.

3.5.2 Comportamento durante l´installazione

Add-Ins possono essere utilizzati nell´Editor o nel Runtime.
Per l´installazione di Add-Ins vale quanto segue:

1. Nuova installazione:
   Il Package selezionato viene salvato nella cartella AddInStore.

- Cartella per installazione nell´Editor (A pagina: 39):
  `%ProgramData%\COPA-DATA\zenon800\EditorAddInStore`
- Cartella per installazione nel Runtime (A pagina: 44):
  `...\RT\FILES\zenon\system\AddInStore\...`

L´ID dell´Add-In viene utilizzato come nome file e viene integrato con l´estensione file **.scadaAddIn**.

2. Nuova installazione con lo stesso ID:
   L´Add-In esistente viene sostituito.
3. Durante l´installazione vengono sostituiti i file già esistenti di altre versioni (successive o precedenti).

Info

L´Add-In ID è univoco per ogni Add-In e viene definito tramite un attributo .NET nel file **AddInInfo.cs**.

Esempio (**Add-In ID** = AddInProject):

```
// Declares that this assembly is an add-in
[assembly: Addin("AddInProject", "1.0")]
```

3.5.3 Nodo “Add-Ins” nel manager di progetto

È possibile importare Add-Ins in un progetto usando il nodo **Scripting** della struttura ad albero del manager di progetto. Per maggiori dettagli, consultare il capitolo **Menù contestuale - Albero di progetto** (A pagina: 37).

Menù contestuale - Albero di progetto

Cliccando con il tasto destro del mouse sul simbolo Add-Ins del nodo **Scripting** dell´albero di progetto, si apre un menù contestuale:

Voce	Descrizione
Add-In Importa Add-In...	Apre la finestra di dialogo che consente di importare un **Add-In Package** per poterlo utilizzare a Runtime. Nota: il pacchetto selezionato verrà solamente importato, non installato. L´installazione verrà eseguita a Runtime.
Profilo Editor	Apre il menù a tendina contenente i profili Editor predefiniti.
Guida	Apre la guida online.

Menù contestuale e barra degli strumenti - Visualizzazione dettagli

Gli Add-In utilizzati a Runtime vengono visualizzati nella visualizzazione dettagli del nodo **Add-In** e gestiti tramite una barra degli strumenti e un menù contestuale.

Voci del menù contestuale e significato dei simboli da sinistra a destra:

Simbolo/Voce del menù contestuale	Descrizione
Add-In Importa	Apre la finestra di dialogo che consente di importare un **Add-In Package** per poterlo utilizzare a Runtime. Nota: il pacchetto selezionato verrà solamente importato, non installato. L´installazione verrà eseguita a Runtime.
Elimina	Disinstalla e cancella l´Add-In selezionato dopo una richiesta di conferma.
Rimuovi tutti i filtri	Rimuove tutti i filtri applicati correntemente nella lista degli Add-In.
Proprietà	Apre la finestra delle **proprietà**.
Guida	Apre la guida online.

3.5.4 Utilizzo nell´Editor di zenon:

Add-Ins da utilizzare nell´Editor di zenon, devono essere importati e installati nell´Editor e vengono gestiti in questa sede.

Add-Ins per l´Editor possono contenere:

- Editor Wizard Extensions (A pagina: 19), che vengono avviate manualmente tramite la voce di menù **Extras -> Avvia Wizards nell´Editor**.

- Editor Service Extensions (A pagina: 19), che vengono avviate quando viene caricato il workspace e la loro modalità di avvio è impostata su `automatico`. Vengono chiuse non appena si chiude il workspace.

Gli Add-Ins installati per l´Editor sono salvati nel seguente percorso:
`%ProgramData%\COPA-DATA\zenon800\EditorAddInStore`

Installare e gestire Add-Ins per l´Editor

Si possono importare, installare e gestire Add-Ins nell´Editor di zenon.

Nota: si possono importare e installare solo Add-Ins con Editor Extensions. Se il sistema non rileva Editor Extensions durante l'importazione, viene visualizzato un avviso corrispondente.

IMPORTARE E INSTALLARE ADD-IN

Per importare e installare un Add-In:

1. Aprire il menù **Extras**.
2. Selezionare la voce **Gestione Add-Insper l´Editor**.
   Si apre la finestra di dialogo che consente di gestire gli Add-Ins.
3. Cliccare sul simbolo **Importa e installa**, o selezionare questa voce nel menù contestuale, oppure premere il tasto `Ins`..

   Si apre la finestra di dialogo (A pagina: 40) che consente di selezionare un Add-In.
4. Selezionare l´Add-In desiderato.
5. Cliccare su **Apri**.

   L´Add-In verrà importato e installato.
6. Cliccare su **Chiudi** per chiudere la finestra di dialogo.

DISINSTALLARE E CANCELLARE ADD-IN

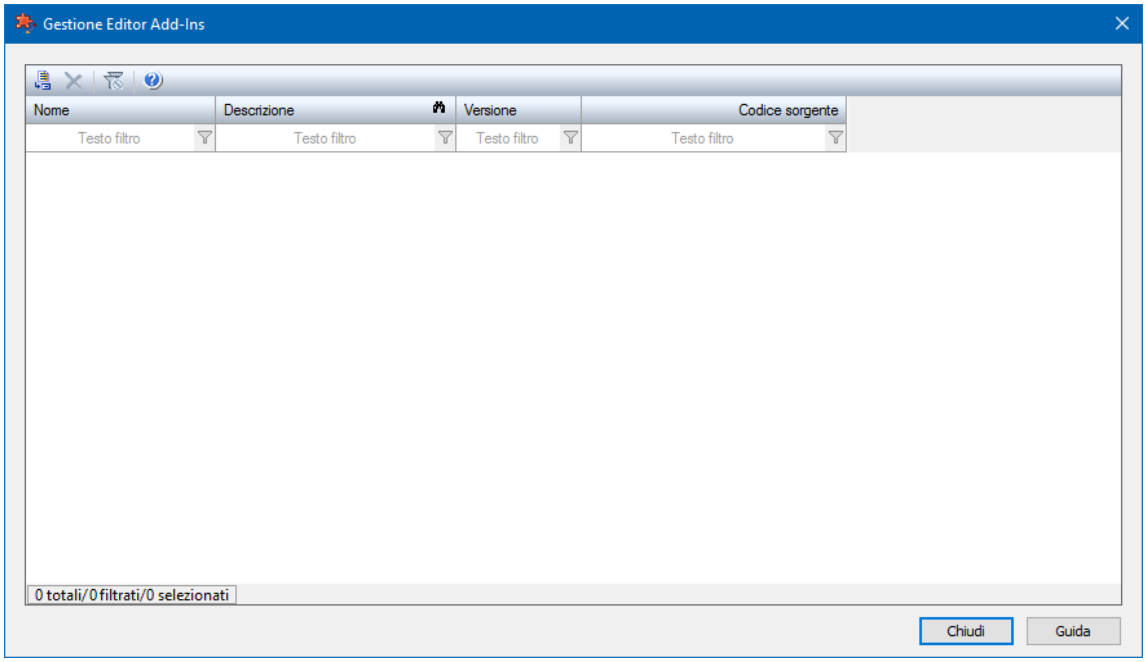
Per disinstallare e cancellare un Add-In:

1. Aprire il menù **Extras**.
2. Selezionare la voce **Gestione Add-Ins per l´Editor** .
   Si apre la finestra di dialogo che consente di gestire gli Add-Ins.
3. Selezionare un Add-In.
4. Cliccare sul simbolo **Disinstalla e cancella**, o selezionare questa voce nel menù contestuale, oppure premere il tasto `Canc`.

   Si apre una finestra di dialogo con una richiesta di conferma.
5. Cliccare su “Sì”.

L´Add-In verrà disinstallato e cancellato.

Finestra di dialogo “Gestione Editor Add-Ins”



Opzione	Descrizione
Barra degli strumenti	Contiene simboli per: ▸ Importare, installare e cancellare Add-Ins ▸ Rimuovere filtri ▸ Visualizzare la guida
Lista degli Add-Ins	Visualizza tutti gli Add-Ins installati. Sono visualizzate informazioni relative a: ▸ Nome ▸ Descrizione ▸ Versione ▸ Se è incluso codice sorgente Gli elementi possono essere filtrati e ordinati.
Chiudi	Chiude la finestra di dialogo.
Guida	Apre la guida online.

BARRA DEGLI STRUMENTI E MENÙ CONTESTUALE

Significato delle voci del menù contestuale e dei simboli da sinistra a destra:

Simbolo	Descrizione
Importa ed installa	Apre la finestra di dialogo che consente di selezionare un Add-In. Quest´ultimo potrà poi essere importato e installato.
Disinstalla e cancella	Disinstalla e cancella l´Add-In selezionato dopo una richiesta di conferma.
Rimuovi tutti i filtri	Rimuove tutti i filtri applicati correntemente nella lista degli Add-Ins.
Guida	Apre la guida online.

Gestione servizi Editor

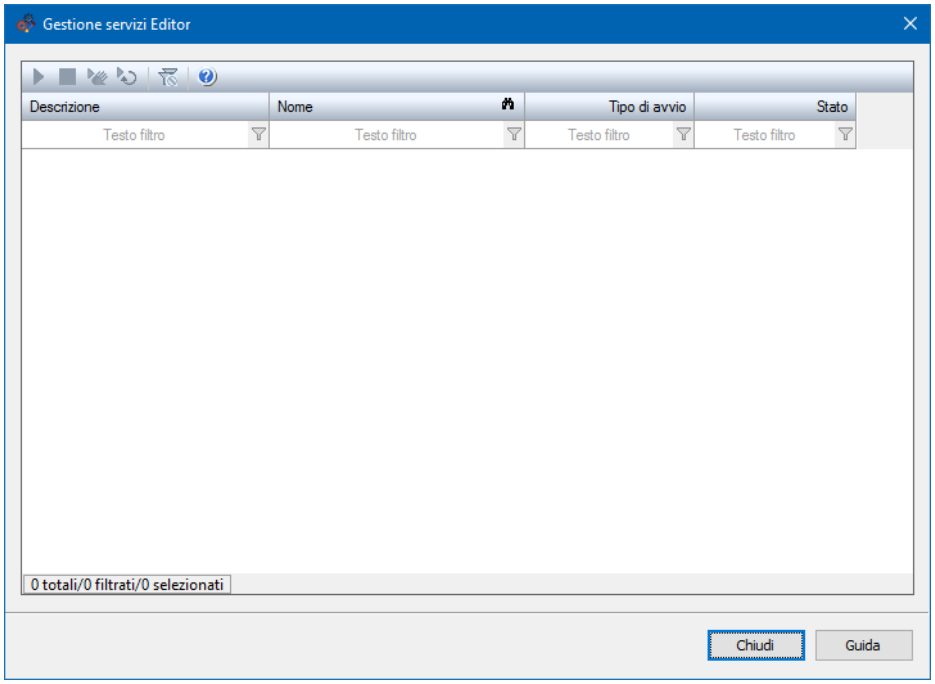
Add-In Extensionseseguite nell´Editor come servizi vengono gestite tramite questa finestra di dialogo.

Per gestire servizi Editor:

1. Aprire il menù **Extras**.
2. Selezionare la voce **Gestione servizi Editor...**.

   Si apre la finestra di dialogo che consente di gestire i servizi.

FINESTRA DI DIALOGO - GESTIONE SERVIZI EDITOR



Opzione	Descrizione
Barra degli strumenti	Contiene simboli per: ▸ Avviare e terminare i servizi. ▸ Impostazione della modalità di avvio Automatica Manuale ▸ Rimuovere filtri ▸ Visualizzare la guida
Lista dei servizi	Visualizza tutti i servizi (Services) installati. Sono visualizzate informazioni relative a: ▸ Nome ▸ Descrizione ▸ Tipo di avvio ▸ Stato Gli elementi possono essere filtrati e ordinati.
Chiudi	Chiude la finestra di dialogo.
Guida	Apre la guida online.

Nota: questa finestra di dialogo è disponibile anche a Runtime (A pagina: 47) tramite la funzione **Visualizza dialogo "Gestione dei Servizi Runtime"**.

BARRA DEGLI STRUMENTI E MENÙ CONTESTUALE

Significato delle voci del menù contestuale e dei simboli da sinistra a destra:

Simbolo	Descrizione
Servizio avviato	Avvia il servizio selezionato.
Servizio arrestato	Termina il servizio selezionato.
Avvio manuale	Imposta su `manuale` la modalità di avvio per il servizio selezionato.
Avvio automatico	Imposta su `automatico` la modalità di avvio per il servizio selezionato.
Rimuovi tutti i filtri	Rimuove tutti i filtri applicati correntemente nella lista dei servizi.
Guida	Apre la guida online.

COMPORTAMENTO PER START E STOP

Se per un Workspace Service Add-In è stata impostata la modalità di avvio `automatico`, il servizio verrà caricato all´avvio dell´Editor prima dell´Event **OnWorkspaceStartup**. Quando si chiude l´Editor, il servizio verrà interrotto dopo l´evento **OnWorkspaceExit**. L´Add-in può quindi reagire agli eventi (Events) di zenon.

3.5.5 Utilizzo nel Runtime di zenon

Add-Ins per l'utilizzo nel Runtime di zenon vengono importati e gestiti nell'Editor. Vengono installati automaticamente al momento dell´avvio del Runtime e potranno poi essere eseguiti nel Runtime stesso. Ogni progetto zenon gestisce i propri Add-Ins.

- Add-Ins vengono copiati nella seguente cartella quando si creano i file Runtime per un progetto : `...\RT\FILES\zenon\system\AddInStore\...`
- All'avvio del Runtime, gli Add-Ins vengono installati nella seguente cartella: `...\RT\FILES\zenon\system\AddInCache\...`

  Nel corso di questo processo:

  - Gli Add-In Packages vengono aggiornati.
    L´aggiornamento viene eseguito utilizzando la data di modifica dell´Add-In Packages.
  - Gli Add-Ins che non esistono più vengono eliminati.
  - Project Service Extensions (A pagina: 21) per cui è stata impostata la modalità di avvio “automatico”, vengono istanziate e avviate automaticamente.
  - Project Wizard Extensions (A pagina: 20) possono essere eseguite usando la funzione **Avvia Project Wizard Extension** di zenon.

Suggerimento

Il modo più veloce di aprire la cartella Runtime è quello di selezionare il progetto nell´Editor e premere la combinazione tasti `Ctrl+Alt+R`.

Installare e gestire Add-Ins per il Runtime

Add-Ins per il Runtime devono essere importati nell´Editor e vengono installati automaticamente all´avvio del Runtime. Per importare Add-Ins per il Runtime si può usare il menù contestuale o la barra degli strumenti della visualizzazione dettagli.

IMPORTARE ADD-IN

Per importare Add-in :

1. Nell´Editor, aprire la visualizzazione dettagli del nodo Add-Ins dell´albero di progetto.
2. Cliccare sul simbolo **Importa Add-In**, o selezionare questa voce nel menù contestuale, oppure premere il tasto `Ins`..

   Si apre la finestra di dialogo che consente di selezionare un Add-In.
3. Selezionare l´ Add-In desiderato.
4. Cliccare su **Apri**.

   L´Add-in verrà importato.

   Info:

   - L´Add-In verrà salvato nei file Runtime solo quando questi ultimi verranno compilati. Cartella di archiviazione: `...\RT\FILES\zenon\system\AddInStore\...`
   - Viene installato, modificato o aggiornato nella seguente cartella quando si avvia o si esegue un reload del Runtime:

     `...\RT\FILES\zenon\system\AddInCache\...`

   Durante la modifica, la sincronizzazione viene eseguita utilizzando il time stamp dell´Add-In Package.
5. Cliccare su **Chiudi** per chiudere la finestra di dialogo.

Nota: è possibile importare e installare solo Add-In con Project Extensions. Se il sistema non rileva Project Extensions al momento dell´importazione, viene visualizzato un avviso corrispondente.

INSTALLARE E DISINSTALLARE ADD-IN

Add-Ins vengono installati e aggiornati automaticamente quando si avvia o si esegue un reload del Runtime.

Procedura:

- All'avvio del Runtime, i Packages installati per il progetto vengono confrontati con quelli importati nell´Editor.
- La sincronizzazione avviene sulla base del time stamp.
- Il Package viene installato quando i time stamp differiscono o se ci sono nuovi Packages. Un pacchetto esistente Package con lo stesso nome viene sovrascritto.
- Se un Package esistente a Runtime non esiste più nell´Editor, viene rimosso anche dal Runtime

RELOAD DI ADD-INS

Gli Add-Ins modificati e nuovi possono essere ricaricati a Runtime.
Prima di eseguire il reload, il sistema verifica se l´Add-In Cache e l´Add-In Store differiscono. Se ci sono differenze, viene avviato il reload.

Procedura:

1. Tutti i servizi Add-In vengono interrotti.
2. Lo stato viene salvato.
3. La Cache viene aggiornata con il contenuto del Store.
4. Vengono ripristinati gli stati di tutti i servizi esistenti già prima della ricarica.
5. Vengono avviati i nuovi servizi aggiunti con la modalità di avvio `Automatico`.

Nota: nel caso dei Wizard, la ricarica non viene attivata fino a quando non viene completata un Wizard già in esecuzione.

CANCELLARE ADD-IN

Per cancellare un Add-In:

1. Nell´Editor, aprire la visualizzazione dettagli del nodo Add-Ins dell´albero di progetto.
2. Selezionare un Add-In.
3. Cliccare sul simbolo **Elimina**, oppure premere il stato `Canc`.

   Si apre una finestra di dialogo con una richiesta di conferma.
4. Cliccare su “Sì”.

L´Add-In verrà cancellato.
Dopo che i file Runtime saranno stati ricompilati, l´Add-In verrà disinstallato quando si avvierà o si eseguirà il reload del Runtime.

AVVIO DI WIZARD E GESTIONE DEI SERVIZI

I Wizard possono essere avviati e i servizi gestiti a Runtime.
Per farlo, si utilizzano funzioni di zenon. Queste ultime vengono progettate nell'Editor e richiamate a Runtime.
Per informazioni dettagliate sulle funzioni disponibili e sulla loro configurazione consultare il capitolo Funzioni di **zenon** (A pagina: 46).

3.5.6 Risoluzione dei problemi

ESTENSIONE .SCADAADDIN DISATTIVATA

A volte Visual Studio disattiva l´estensione .scadaAddIn.

In questo caso, l´estensione può essere attivata manualmente ed è disponibile dopo un riavvio. Dopo la chiusura di Visual Studio, tuttavia, l´estensione verrà disattivata alla successiva apertura.

Per i dettagli, vedere: https://github.com/madskristensen/BundlerMinifier/issues/214 (https://github.com/madskristensen/BundlerMinifier/issues/214)

SOLUZIONE:

Per modificare in modo permanente il comportamento:

1. Disinstallare COPA-DATA **SCADA Add-Ins**.
2. Utilizzare **DuplicateExtensiuonFinder** per rimuovere tutti i residui di installazione.
   L´applicazione si trova in: https://github.com/remcoros/DuplicateExtensionFinder (https://github.com/remcoros/DuplicateExtensionFinder)
3. Reinstallare COPA-DATA **SCADA Add-In**.

3.6 Funzioni di zenon

In zenon sono disponibili funzioni per Add-Ins:

- **Visualizza dialogo "Gestione dei servizi Runtime”** (A pagina: 47): apre la finestra di dialogo per la gestione dei servizi a Runtime.

- **Avvia Project Wizard Extension** (A pagina: 50): selezione di un Wizard da eseguire a Runtime.

3.6.1 Visualizza dialogo "Gestione dei Servizi Runtime"

Questa funzione apre a Runtime la finestra di dialogo **Gestione servizi Runtime**. Tutti i servizi Add-In installati sono disponibili per essere amministrati.
È possibile:

- Avviare e terminare servizi.
- Specificare se un servizio viene avviato automaticamente o manualmente.

CONFIGURARE LA FUNZIONE

Procedura di creazione della funzione:

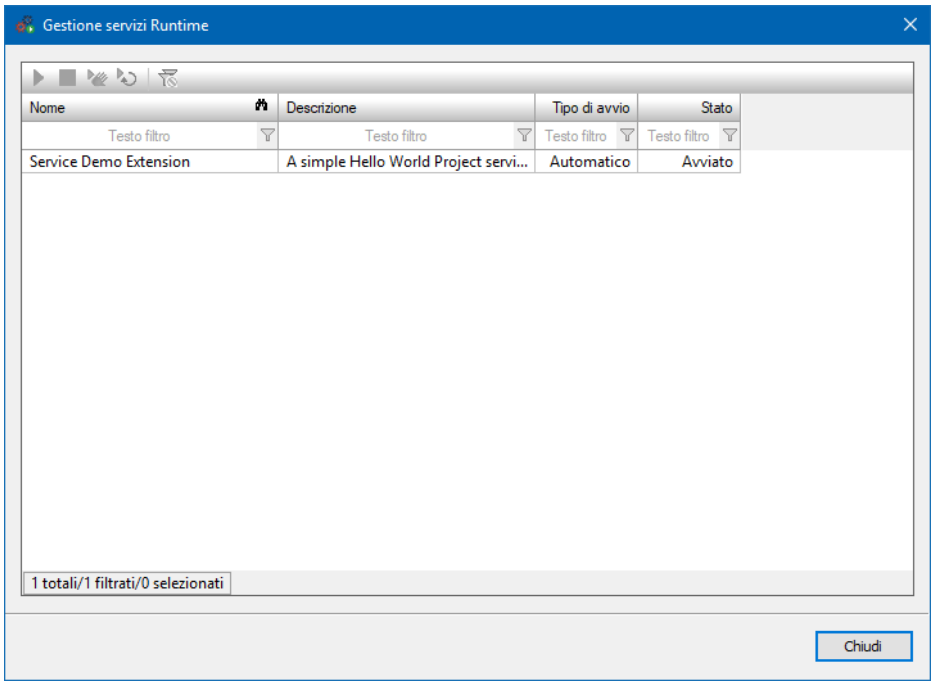
1. Creare una nuova funzione:

   Nella barra degli strumenti, oppure nel menù contestuale del nodo "Funzioni", selezionare il comando **Nuova funzione**.
   Si apre la finestra di dialogo che consente di selezionare una funzione.
2. Passare al nodo **Add-In**.
3. Selezionare la funzione **Visualizza dialogo "Gestione dei Servizi Runtime"**.

   La finestra di dialogo viene chiusa.
4. Assegnare un nome alla funzione nella proprietà **Nome**.

COMPORTAMENTO A RUNTIME

A Runtime, l'esecuzione della funzione apre la finestra di dialogo**Gestione dei Servizi runtime**:



Opzione	Descrizione
Barra degli strumenti	Contiene simboli per: ▸ Avviare e terminare i servizi. ▸ Impostazione della modalità di avvio: Automatico Manuale ▸ Rimuovere filtri
Lista dei servizi	Visualizza tutti i servizi (Services) installati. Sono visualizzate informazioni relative a: ▸ Nome ▸ Descrizione ▸ Tipo di start ▸ Stato Gli elementi possono essere filtrati e ordinati. Le azioni per gli elementi visualizzati possono essere eseguite dalla barra degli strumenti o dal menù contestuale.
Chiudi	Chiude la finestra di dialogo.

BARRA DEGLI STRUMENTI E MENÙ CONTESTUALE

Voci del menù contestuale e significato dei simboli (da sinistra a destra):

Simbolo	Descrizione
Servizio avviato	Avvia il servizio selezionato.
Servizio arrestato	Termina il servizio selezionato.
Avvio manuale	Imposta su `automatico` la modalità di avvio per il servizio selezionato.
Avvio automatico	Imposta su `automatico` la modalità di avvio per il servizio selezionato.
Rimuovi tutti i filtri	Rimuove tutti i filtri applicati correntemente nella lista dei servizi.

3.6.2 Avvia Project Wizard Extension

La funzione **Avvia Project Wizard Extension** consente di eseguire un Wizard a Runtime.
Il Wizard deve essere già stato importato nell'Editor.

CONFIGURARE LA FUNZIONE

Procedura di creazione della funzione:

1. Creare una nuova funzione:

   Nella barra degli strumenti, oppure nel menù contestuale del nodo "Funzioni", selezionare il comando **Nuova funzione**.
   Si apre la finestra di dialogo che consente di selezionare una funzione.

2. Passare al nodo **Add-In**.

3. Selezionare la funzione **Avvia Project Wizard Extension**.

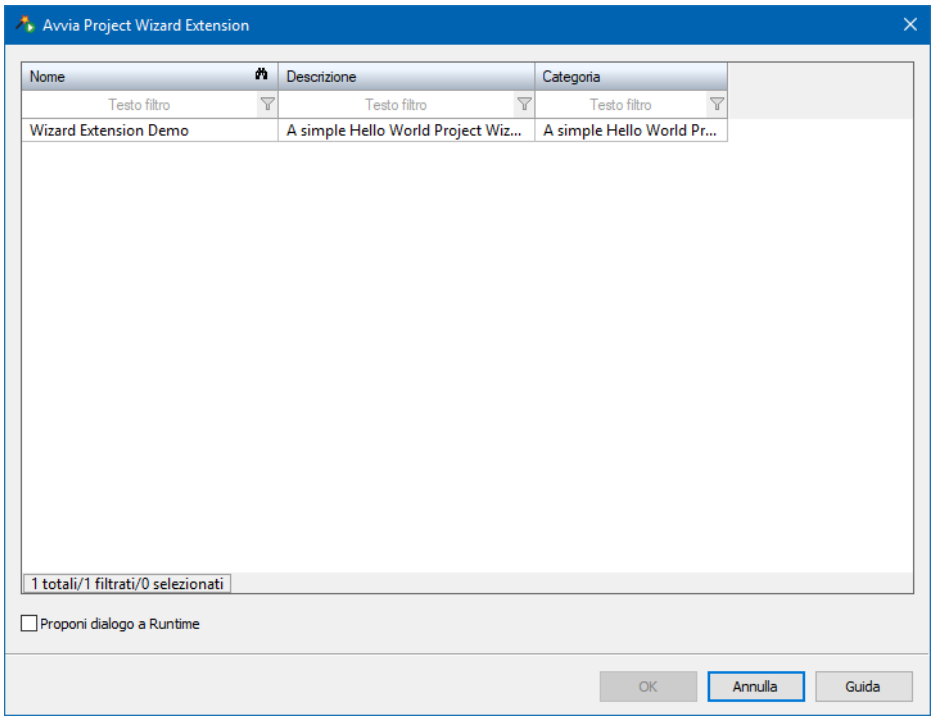
   Si apre la finestra di dialogo per la selezione di un Wizard.

4. Selezionare lil Wizard desiderato.

5. Chiudere la finestra di dialogo cliccando su **OK**.

6. Assegnare un nome alla funzione nella proprietà **Nome**.

FINESTRA DI DIALOGO "AVVIA PROJECT WIZARD EXTENSION"



Opzione	Descrizione
Lista dei Wizard	Visualizza tutti i Wizard presenti nel nodo **Scripting/Add-Ins** dell´albero di progetto. Gli elementi visualizzati possono essere filtrati e ordinati. Criteri: ▸ Nome ▸ Descrizione ▸ Categoria È possibile selezionare un solo Wizard.
Riga di stato	Visualizza: ▸ Numero complessivo dei Wizard. ▸ Numero dei Wizard filtrati. ▸ Numero dei Wizard selezionati.
Proponi dialogo a Runtime	`Attiva`: al momento della chiamata funzione a Runtime, verrà visualizzata questa finestra di dialogo e l´utente potrà modificare la configurazione prima dell´esecuzione della funzione. La finestra di dialogo verrà aperta a Runtime sul computer corrente. Nel funzionamento di rete, la finestra di dialogo verrà aperta sul client se la chiamata funzione avviene sul client.

CHIUDERE LA FINESTRA DI DIALOGO

OK	Applica le impostazioni e chiude la finestra di dialogo.
Annulla	Annulla tutte le modifiche e chiude la finestra di dialogo.
Guida	Apre la guida online. Disponibile solo nell´Editor.

COMPORTAMENTO A RUNTIME

A Runtime, un clic su un elemento collegato alla funzione esegue il Wizard configurato.
Se l'opzione **Proponi dialogo a Runtime** è stata attivata, la finestra di dialogo per la selezione di un Wizard viene aperta prima dell'esecuzione della funzione. È possibile così selezionare nuovamente il Wizard da eseguire.

Se si seleziona il pulsante **Annulla** in questa finestra di dialogo a Runtime, non viene eseguito nessun Wizard.

3.7 Cambio/conversione di VSTA

EDITOR

Procedura per convertire funzionalità VSTA esistenti dell´Editor.

FUNZIONALITÀ DI WIZARD

Le funzionalità prima implementate con Wizard VSTA, possono ora essere implementate mediante Editor Wizard Extensions (A pagina: 19).

Copiare il codice esistente in una Editor Wizard Extension e, se lo si desidera, modificarlo.

Per avviare il Wizard, utilizzare la finestra di dialogo **Avvia Wizards in Editor...**

FUNZIONALITÀ DI VSTA EVENTS

Funzionalità attivate da eventi (ad es. tramite **.OnElementCreated**, **.OnPreBuild**), possono ora essere implementate tramite Editor Service Extensions (A pagina: 19).

Per fare ciò, copiare il codice per inizializzare i gestori degli eventi nel metodo **Start** dell´Editor Service Extension. Copiare il codice per il rilascio nel metodo **Stop**.

Nota: Per l´avvio automatico bisogna impostare nell´**Extension** l´attributo `DefaultStartMode=DefaultStartupModes.Auto`. La modalità di avvio può essere modificata in un secondo momento nella finestra di dialogo **Gestione servizi Editor** (A pagina: 41).

FUNZIONALITÀ DI MACRO DI VSTA

La funzionalità di macro eseguite utilizzando il combobox **Esegui la macro VBA/VSTA**, può ora essere implementata utilizzando Editor Wizard Extensions (A pagina: 19).

Copiare il codice dalla macro VSTA esistente nel metodo **Run** dell´Editor Wizard Extension e apportare le necessarie modifiche al codice.

Per eseguire il codice, utilizzare la finestra di dialogo **Avvia Wizards in Editor...**

RUNTIME

Procedura per convertire funzionalità VSTA esistenti del Runtime:

FUNZIONALITÀ DI MACRO DI VSTA

Funzionalità che venivano eseguite tramite la funzione **Lanciare Macro-VSTA**, possono ora essere implementate tramite Project Wizard Extensions (A pagina: 20).

Copiare il codice dalla macro VSTA esistente nel metodo **Run** dell´Project Wizard Extension e apportare le necessarie modifiche al codice.

Per avviare la Project Wizard Extension utilizzare la funzione **Avvia Project Wizard Extension** di zenon.

FUNZIONALITÀ DI VSTA EVENTS

Funzionalità attivate da eventi (ad es. tramite **.DynPictures().Open**, **.Alarm().AlarmComes**), possono ora essere implementate tramite Project Service Extensions (A pagina: 21).

Per fare ciò, copiare il codice per inizializzare i gestori degli eventi nel metodo **Start** dell´Project Service Extension. Copiare il codice per il rilascio nel metodo **Stop**.

Nota: Per l´avvio automatico bisogna impostare nell´**Extension** l´attributo `DefaultStartMode=DefaultStartupModes.Auto`.

4. Lista Macro

Si può utilizzare VBA e VSTA per estendere autonomamente le funzionalità di zenon. Viene descritto l'uso di macro con zenon.

MENÙ CONTESTUALE

Voce di menù	Azione
Apri Editor-VBA	Apre l'Editor VBA.
Esporta tutti VBE...	Apre la finestra di dialogo per la selezione della posizione di memorizzazione per l'esportazione VBE.
Importa VBE	Apre la finestra di dialogo per la selezione del file di importazione VBE.
Profilo Editor	Apre il menù a tendina contenente i profili Editor predefiniti.
Guida	Apre la guida online.

Info

Quando le macro VBA vengono modificate nell'Editor:

- I file Runtime vengono compilati e trasferiti al Runtime.
- Viene eseguito un reload del Runtime.
- Anche gli elementi del VSTA vengono ricaricati, anche se non sono state apportate modifiche al VSTA.

VBA avvia lo stesso ambiente di sviluppo per **Workspace** e **progetto**.
Per aprire l´Editor di VBA:

1. Nell´Editor di zenon, passare al nodo **Scripting**.
2. Estendere questo nodo cliccando su `[+]`.
   Viene visualizzata la struttura del nodo.
3. Cliccare con il tasto destro del mouse su **Lista Macro**.
4. Nel menù contestuale, selezionare la voce **Apri Editor-VBA**.

In alternativa: Premere la combinazione di tasti `Alt+F11`

4.1 VBA: Barra degli strumenti e menù contestuale - Visualizzazione dettagli

BARRA DEGLI STRUMENTI

Voce di menù	Azione
Nuova macro VBA	Crea una nuova macro e apre l´Editor delle macro.
Apri Editor-VBA	Apre l'Editor VBA.
Salva	Salva macro.
Elimina	Cancella l´elemento selezionato.
Esporta tutti VBE...	Apre la finestra di dialogo per la selezione della posizione di memorizzazione per l'esportazione VBE.
Importa VBE	Apre la finestra di dialogo per la selezione del file di importazione VBE.
Rinomina	Consente di rinominare le macro selezionate.
Guida	Apre la guida online.

MENÙ CONTESTUALE - MODULI MACRO

Voce di menù	Azione
Apri Editor-VBA	Apre l'Editor VBA.
Salva	Salva macro.
Esporta tutti VBE...	Apre la finestra di dialogo per la selezione della posizione di memorizzazione per l'esportazione VBE.
Importa VBE	Apre la finestra di dialogo per la selezione del file di importazione VBE.
Guida	Apre la guida online.

MENÙ CONTESTUALE - MODULI

Voce di menù	Azione
Nuova macro VBA	Crea una nuova macro e apre l´Editor VBA.
Guida	Apre la guida online.

MENÙ CONTESTUALE - MACRO

Voce di menù	Azione
Modifica	Apre la macro nell´Editor per consentire le modifiche. Alternativa: `tasto INVIO` oppure doppio clic.
Elimina	Cancella la macro. Alternativa: `Tasto Cancella`.
Rinomina	Apre il campo del nome in modalità di modifica. Alternativa: tasto `F2`.
Guida	Apre la guida online.

BARRA DEGLI STRUMENTI DELL´EDITOR

Le macro create con VBA possono essere gestite usando la barra degli strumenti **Lista macro**.

Se si seleziona un filtro, questa selezione è visibile anche nella barra degli strumenti.

Simbolo	Descrizione
Aggiorna la lista delle macro VBA/VSTA	Carica nel menù a tendina della barra degli strumenti tutte le macro `Public Sub Name ()` contenute in **myWorkspace** e nei moduli.
Cerca macro	Ricerca di macro effettuata usando il campo di inserimento del menù a tendina, oppure selezionandone una fra quelle offerte dal menù a tendina. Quando si apre il menù a tendina, la sua dimensione viene adeguata a quella dell´elemento più largo.
Menù a tendina delle macro	Contiene tutte le macro caricate che potranno essere selezionate.
Esegui macro selezionata.	Esegue la macro selezionata nel menù a tendina.
Esegui la macro assegnata #<x>.	Esegue la macro assegnata al simbolo.
Assegna macro	Apre la finestra di dialogo per l'assegnazione delle macro. È possibile collegare un massimo di 5 macro ai simboli dall´1 al 5.
VBA	Filtra per macro VBA. Verranno visualizzate solamente le macro VBA.
VSTA	Filtra per macro VSTA. Verranno visualizzate solamente le macro VSTA.
Visualizza tutte le macro	Il filtro attualmente applicato viene rimosso e vengono visualizzate tutte le macro.
AZ	Ordina le macro in ordine ascendente, da 0 a 9 e dalla A alla Z.
ZA	Ordina le macro in ordine discendente, dalla Z alla A e da 9 a 0.
Opzioni barre degli strumenti	Cliccando sulla freccia si apre il sotto-menù: `Attiva`: la barra degli strumenti viene visualizzata. Se la barra degli strumenti non viene visualizzata, la si può attivare usando il menù **Visuale-> Toolbar**. Nota: in caso di posizionamento libero della barra degli strumenti (sganciata dall´Editor), le opzioni non vengono visualizzate. La barra degli strumenti può essere chiusa cliccando sul pulsante X.

Info

Se nella finestra di dialogo di collegamento delle macro non vengono visualizzate tutte la macro contenute in `myWorkspace`, eseguire la funzione **Aggiorna la lista delle macro VBA**.

4.2 VBA su sistemi a 64-Bit

A partire dalla versione 7.10, zenon supporta i sistemi operativi a 64-Bit. VBA è stato pertanto convertito nella versione VBA 7.1. Per questa ragione VBA è disponibile adesso anche nella versione di zenon a 64-Bit. Se nel codice VBA si accede alle API di Windows o ad altre funzioni DLL importate, queste chiamate devono essere adattate a 64 bit. In generale vale quanto segue: Un file VBA creato per una versione a 32-Bit non può essere utilizzato senza modifiche-in una versione a 64-bit di VBA.

In VBA esistono alcune definizioni/funzioni che consentono di scrivere un codice compatibile per 32 Bit e per 64 Bit. Per 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
```

Informazioni utili sulla portabilità di codice VBA a 32-bit su VBA a 64 bit sono fornite anche da Microsoft:

- Microsoft Office 2010, note sulla portabilità: 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 Nozioni di base

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

4.3.1 Proprietà degli oggetti - PROPERTIES

Una proprietà oggetto (property) è un attributo specifico dell'oggetto. Nel caso di un oggetto “variabile”, ad esempio, la proprietà può essere il valore, il nome o l'identificatore. Nel caso di un oggetto “cerchio”, ad esempio, la posizione o il colore del cerchio nell'immagine. Ogni oggetto ha almeno una proprietà (di solito più), ogni proprietà ha un certo valore. Mentre il **Property Name** è un testo, **Property Value** ha un valore compreso fra 0 e, per es., 1000.

La caratteristica speciale delle Properties è che è possibile modificare il comportamento o l'aspetto dell'oggetto modificando i valori delle proprietà in un programma VBA. Se, ad esempio, si assegna un valore diverso a **Property Value** (valore) dell'oggetto “variabile”, si ottiene il che la variabile selezionata ottenga il valore desiderato. Non è però possibile assegnare un nuovo valore a ogni proprietà. La **Property Count** (quantità) dell'oggetto “variabile”, ad esempio, è fissa, perché il numero di variabili è determinato dalle variabili create. Dando a Count un valore più alto, non si aggiungono nuove variabili. Alcune Properties sono di sola lettura, cioè il loro valore può essere letto ma non modificato.

4.3.2 Metodi ad oggetti - METHODS

Oltre alle Properties, ogni oggetto può avere dei metodi. Un metodo non è un attributo, ma una richiesta all'oggetto di fare qualcosa. Così un form ha il metodo `Show`. Quale effetto ha questo metodo? Richiede al form di visualizzare se stesso sullo schermo. Di conseguenza, il form scompare con il comando `Unload`.

Il vantaggio di un metodo è che il programmatore non ha bisogno di sapere nulla sulla struttura dell'oggetto e soprattutto non ha la possibilità di ruotare i dati interni dell'oggetto.

La chiamata del metodo Show o del comando Unload ha il seguente aspetto:

`frmSollwert.Show` oppure `Unload frmSollwert`

Se si desidera, ad esempio, passare a un altro form, il metodo rimane lo stesso, cambia solo il nome del form (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 necessariamente avere metodi.

4.3.3 Eventi di oggetti - EVENTS

Anche se nel 90 per cento dei casi si ha a che fare con properties e metodi quando si lavora con oggetti, c'è una terza categoria di caratteristiche che un oggetto può avere: Events. Alcuni oggetti nella gerarchia degli oggetti possono reagire ad eventi. Gli eventi si verificano automaticamente quando si lavora con zenon.

Esempio

Ogni volta che viene visualizzata un'immagine, viene attivato un open-event nell'oggetto "picture" che si trova alla base di essa. Come programmatore, è possibile definire istruzioni all'interno della cosiddetta procedura evento (procedura che viene chiamata quando si verifica un evento) che determinano cosa dovrebbe accadere in questo caso. Un esempio di questo è la modifica di una variabile. È possibile creare un Event che reagisce alla variazione di valore di una variabile.

4.3.4 Modello di oggetti VBA in zenon

C'è sempre una lista di oggetti e oggetti nella struttura del progetto.

Esempio:

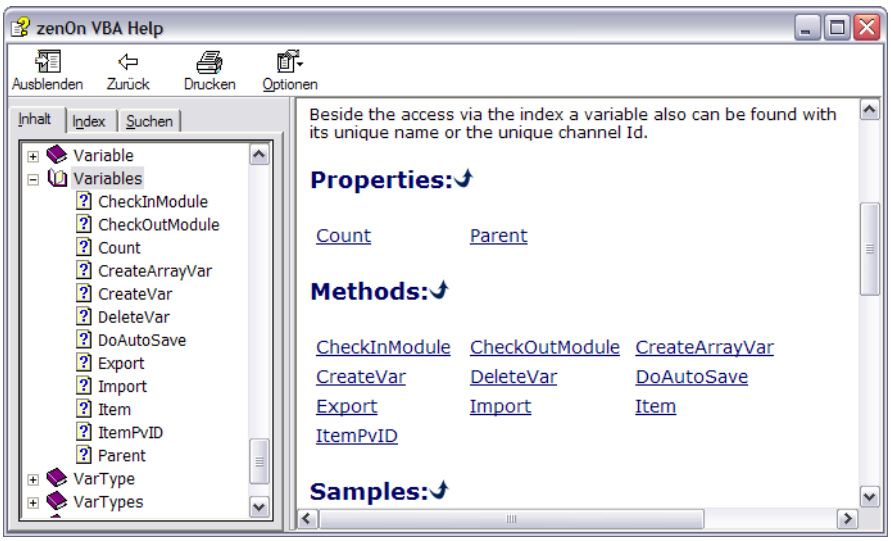
Projects – Project

Variables – Variable

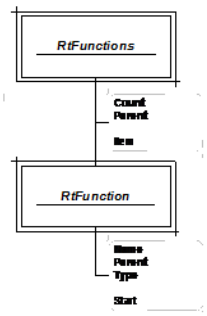
Elements – Element

Ulteriori informazioni sul modello oggetto si trovano:

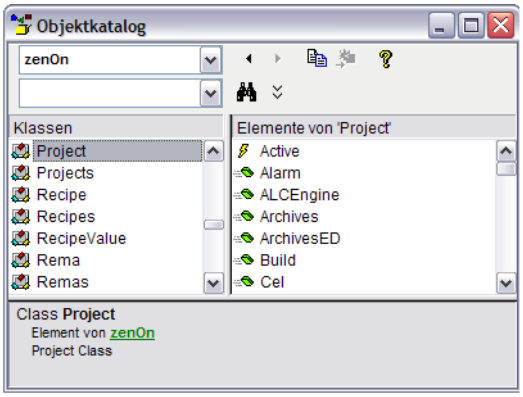
- Nella guida in linea VBA



- Nella panoramica grafica, che si può ottenere sotto forma di panoramica stampata da COPA-DATA

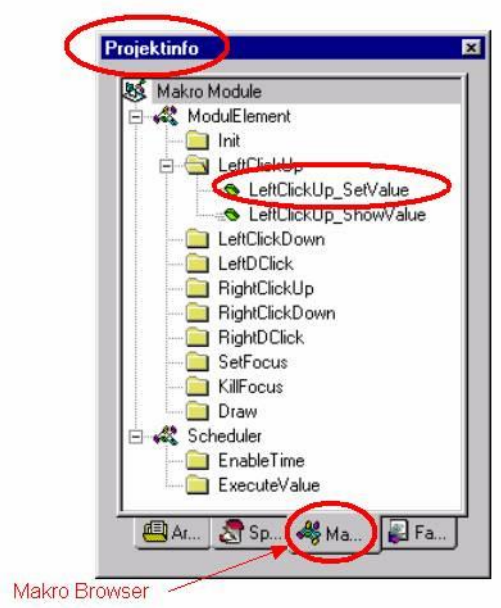


- Nell´Object-Browser di VBA



4.3.5 Come utilizzare le macro VBA?

Per creare una nuova macro, selezionare nella finestra **Informazione progetto** alla voce **Browser macro** 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 menù.

Selezionare la voce di menù “Nuova macro” In questo modo viene generata una procedura da zenon:

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

Se una macro esiste già, è possibile modificarla, cancellarla o rinominarla facendo clic con il pulsante destro del mouse.

> **Attenzione**
>
> *Se si seleziona la voce di menù **Rinomina macro**, assicurarsi di non modificare il nome dell'evento (ad esempio: LeftClickUp_...) del nome attuale. In caso contrario, il cambio di nome non viene eseguito. Se si cambia il nome della macro, è necessario anche cambiare manualmente il nome della subroutine da richiamare nell'Editor VBA.*

Dopo aver popolato la procedura generata da zenon con il codice sorgente da eseguire, si deve collegare la macro generata con un elemento.

Facendo doppio clic su un elemento desiderato, le proprietà dell'elemento vengono visualizzate nella schermata.

Nella scheda “Eventi”, collegare la macro all'elemento.

Facendo clic con il tasto sinistro del mouse, si attiva l'evento LeftClickDown dell'elemento e si esegue la macro collegata.

Integrazione di macro esistenti

Per integrare macro esistenti in un altro progetto, procedere come segue:

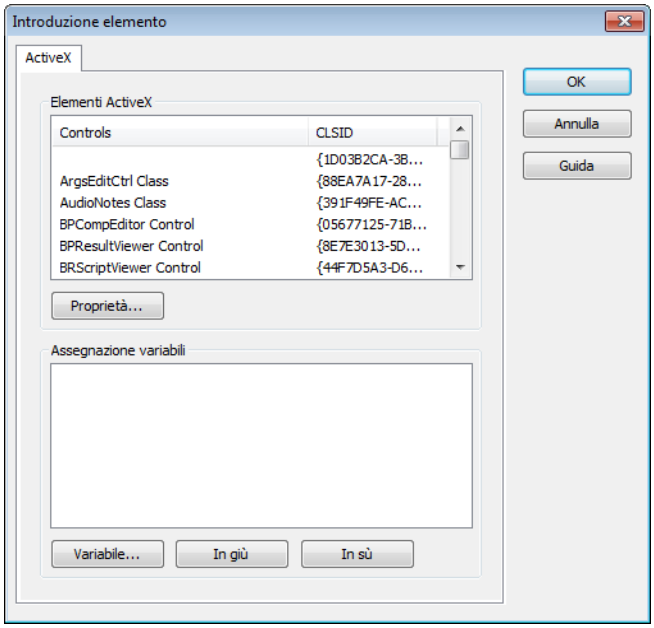
1. Esportare tutti i form e i moduli richiesti nell'Editor VBA e importarli nel progetto desiderato.
2. Le macro correlate agli eventi, in `ModulElement.bas`, non sono ancora visualizzate nel browser delle macro. Pertanto, è ora necessario creare nuove macro nel browser delle macro.

   Il modo più semplice è usare il nome della macro esistente.

   Per es..:
   LeftClickUp_ `DateSet2`
   LeftClickUp_ `DateSet4`
   LeftClickUp_ `TimeSet`
   Draw_ `Date2`
   Draw_ `Date4`
   Draw_ `Time`
3. Durante la creazione di macro, zenon genera procedure con lo stesso nome delle macro esistenti. Queste nuove procedure generate devono essere cancellate.
4. Collegare le macro a un elemento dinamico come di consueto.

4.3.6 Come implementare un elemento ActiveX in zenon?

Come un elemento dinamico, un elemento ActiveX viene semplicemente trascinato nella dimensione desiderata in un'immagine; si apre quindi una finestra in cui è necessario selezionare un elemento ActiveX.



- Quando l'elemento è stato selezionato nella lista sopra visualizzata, è possibile assegnargli una variabile. Per farlo, cliccare sul pulsante **Variabile...** e selezionare una variabile o crearne una nuova.
- Il passo successivo è quello di dare all´elemento ActiveX un nome oggetto, che potrà essere usato per accedervi in seguito in VBA.
- Nel nostro esempio, assegniamo il nome oggetto Slider6_DW18, in quanto si tratta di un elemento Slider ActiveX che è collegato alla variabile Data Word 18.
- Ora l'elemento slider deve essere attivato e ulteriormente elaborato tramite l'Editor VBA.
- Per fare questo, si crea una nuova macro, come già descritto nel capitolo "Come utilizzare le macro VBA? (A pagina: 62)".

La macro Init_slider passa l'elemento da inizializzare a una subroutine nell'oggetto `thisProject`, che crea un riferimento di progetto al progetto corrente:

```
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
```

Il codice sorgente elencato di seguito viene inserito in questo oggetto thisProject:

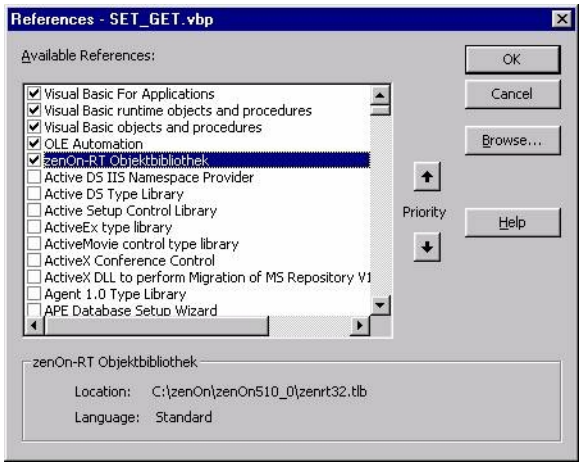
```
Public Declarations
Public WithEvents obSlider As Slider
Public obSliderPV As Variable
Public Sub Init_Slider (obElem As Element)
Set obSlider = obElem.ActiveX
'ActiveX exists
If obSchieberegler Is Nothing Then
Exit Sub
End If
Set obSliderPV = obElem . ItemVariable(0)
'variable exists
If obSliderPV 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 (IMMAGINE_1)
'variable exists
If obSliderPV Is Nothing Then
Exit Sub
End If
```

4.3.7 Accesso mediante un programma esterno

L'interfaccia COM consente l'accesso ai dati di zenon tramite un programma esterno come Visual Basic. Questa interfaccia COM è quella utilizzata anche da VBA. Quindi ci sono solo alcune piccole differenze che devono essere prese in considerazione.

Visual Basic 6

Per accedere all'interfaccia COM, è necessario integrarla



Questa libreria di tipi consente di accedere all'oggetto applicazione di zenon (il Runtime).

Poiché qui non c'è nessun oggetto thisProject, è necessario crearne uno per ottenere i dati.

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

Se il progetto VB funziona per tutti i progetti di zenon e deve essere indipendente dal nome del progetto, può anche essere impostato in questo modo:

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

Ad esempio, se l'oggetto progetto (thisProject) è stato creato, è possibile accedere alle variabili in lettura e scrittura.

Lettura

```
Valore = obProject.Variables.Item(nome variabile).Value
```

Scrittura:

```
obProject.Variables.Item(nome variabile).Value = valore
```

4.3.8 Funzionalità di una variabile online

Una variabile online VBA può essere immaginata come un contenitore, questo contenitore contiene variabili, che naturalmente devono essere inserite. Se il valore di una variabile nel contenitore cambia, si attiva un evento.

Funzionalità dell'evento

Se il contenitore è attivato (`Container.Define`), tutte le variabili nel contenitore vengono inizializzate una volta in modo che il valore corrente della variabile sia noto. La procedura "`Container_VariableChange`" viene quindi eseguita una volta per ogni variabile del contenitore. Quando tutte le variabili sono state inizializzate, l'evento si attiva ogni volta che il valore di una variabile cambia nel contenitore

Non è quindi possibile leggere un valore che non corrisponda al valore attuale della variabile.

Definire e creare contenitori

Definizione:

```
Public WithEvents Container As OnlineVariable
```

Questa riga di codice definisce il contenitore.

Creazione:

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

Immissione di variabili nel contenitore

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

Questo processo viene ripetuto fino a quando tutte le variabili desiderate non sono nel contenitore.

Creazione di un evento

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

Questo evento viene creato automaticamente quando il contenitore viene selezionato nella combobox di sinistra nella parte superiore dell'Editor VBA. La procedura sopra descritta viene quindi inserita nel codice sorgente. Utilizzare "`obVar`" per specificare la variabile il cui valore è stato modificato. Quando si verifica l'evento, ad esempio, è possibile richiedere il valore corrente (`ob-Var.Value`). Le proprietà e i valori che possono essere richiesti dalla variabile sono visibili nella gerarchia degli oggetti della documentazione VBA.

Attivare un evento

```
Container.Define
```

Questa riga di comando attiva il monitoraggio delle variabili nel contenitore. Il contenitore è attivo solo quando viene eseguito il comando "**Define**“.

Disattivazione dell'evento

```
Container.Undefine
```

Con questo comando il monitoraggio nel contenitore viene nuovamente disattivato, l'evento (VariableChange) non viene più eseguito da questo momento.

Rimuovere all'uscita

Per non lasciare veramente nulla nella memoria al momento della chiusura del Runtime, il container deve essere eliminato al più tardi al momento del termine del Runtime.

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

Solo quando questo contenitore è stato cancellato si può creare un altro contenitore con lo stesso nome.

4.3.9 Lista dei bit di stato

Numero bit	Nome breve	Nome lungo	Nome zenon Logic
0	M1	Stato utente 1, o tipo di azione "Blocco" nel modulo "Comandi", oppure Service Tracking (Main.chm::/IEC850.chm::/117281.htm) del driver IEC 850.	_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	Comando Runtime exceeded (command runtime exceeded)	_VSB_RTE
12	MAN_VAL	Valore manuale	_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	Interrogazione generale	_VSB_GR
17	SPONT	Spontaneo	_VSB_SPONT
18	INVALID	Non valido	_VSB_I_BIT
19	T_STD_E	Orario standard (ora solare) esterno Attenzione: fino alla versione 7.50, questo era il bit di stato T_CHG_A	_VSB_SUWI
20	OFF	Spento	_VSB_N_UPD
21	T_EXTERN	Real time - time stamp esterno	_VSB_RT_E
22	T_INTERN	Time stamp interno	_VSB_RT_I
23	N_SORTAB	Non ordinabile	_VSB_NSORT
24	FM_TR	Messaggio di errore Valore	_VSB_DM_TR

		trasformatore	
25	RM_TR	Messaggio default Valore trasformatore	_VSB_RM_TR
26	INFO	Informazione della variabile	_VSB_INFO
27	ALT_VAL	Valore di riserva	_VSB_AVALUE
28	RES28	Riservato a uso interno (lampeggio allarmi)	_VSB_RES28
29	N_UPDATE	Non aggiornato (rete di zenon)	_VSB_ACTUAL
30	T_STD	Orario standard interno	_VSB_WINTER
31	RES31	Riservato a 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	Conferma negativa del comando dal dispositivo (IEC 60870 [P/N])	_VSB_PN_BIT
39	TEST	Test-Bit (IEC 60870 [T])	_VSB_T_BIT
40	WR_ACK	Conferma di scrittura	_VSB_WR_ACK
41	WR_SUC	Scrittura eseguita 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	Time stamp esterno non valido	non definito
50	CB_TRIP	Intervento sezionatore rilevato	non definito
51	CB_TR_I	Rilevamento di intervento sezionatore disabilitato	non definito
52	OR_DRV	Valore fuori intervallo di validità (OR_DRV)	non definito

53	T_UNSYNC	ClockNotSynchronized (IEC 61850)	non definito
54	PR_NR	Non registrato nel process recorder	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

Info

Nelle formule sono disponibili tutti i bit di stato. Per altri usi, i bit di stato possono essere disponibili solo in forma ridotta.

Per i dettagli relativi all´elaborazione degli stati, si prega di consultare il capitolo Elaborazione stato.

4.3.10 Lazo per selezionare elementi dinamici a Runtime

Gli elementi dinamici collegati a una variabile o a una funzione, possono essere preselezionati a Runtime con il lazo e quindi utilizzati per gli eventi.

Esempio: è possibile utilizzare il lazo per selezionare elementi dinamici dalla barra di navigazione e inserirli nel Trend esteso utilizzando DragDrop. Trascinare gli elementi dinamici selezionati nell'area del diagramma ETM o nell'elenco delle curve ETM. Le curve corrispondenti vengono quindi create automaticamente per questi elementi dinamici.

Nota: per gli elementi dinamici che hanno più di una variabile, solo la variabile principale viene utilizzata per la funzione Drag&Drop Funktion.

Il metodo **SelElements** può essere usato a Runtime per identificare gli elementi dinamici scelti dall´utente nel Runtime come selezionat dall'utente come selezionati. Questi **DynPicture.SelElements** possono quindi essere utilizzati per gli eventi tramite Drag&Drop.

SELEZIONE MEDIANTE LAZO

Perché sia possibile selezionare elementi a Runtime mediante lazo, devono essere eseguite le seguenti configurazioni nell´Editor:

- Nelle proprietà di progetto deve essere attivata la proprietà **Selezioni multiple in Runtime (Lazo)** del gruppo **Interazione**.

  Default: non attiva.

- Nel gruppo **Runtime** delle proprietà dell'elemento dinamico, attivare la proprietà **selezionabile tramite lazo**.

  Default: attiva.

- La proprietà **Muovi modello via mouse** deve essere disattivata nel gruppo **Interazione** delle immagini desiderate.

  Default: non attiva.

A seconda delle impostazioni per il funzionamento touch, a Runtime sono disponibili diversi metodi per la selezione degli elementi:

- Gestione Touch disabilitata, o Windows 7 (proprietà **Analisi** `Disattivata` oppure `Windows 7`):
  - `Clic con il tasto sinistro del mouse + movimento`: viene creata una nuova selezione.
  - `Clic con il tasto sinistro del mouse + tasto Ctrl + movimento`: la selezione viene estesa.
- Gestione Touch nativa Windows 8 attiva (proprietà **Analisi** su `Windows 8`):
  - `Clic con il tasto sinistro del mouse + movimento`: l´immagine viene spostata.
  - `Clic con il tasto sinistro del mouse + tasto Ctrl + movimento`: l´immagine viene spostata.
  - `Clic con il tasto sinistro del mouse + tasto Shift + movimento`: viene creata una nuova selezione.
  - `Clic con il tasto sinistro del mouse + tasto Ctrl + tasto Shift + movimento`: la selezione viene estesa.

Regole:

- È possibile selezionare solo gli elementi visibili.
- Se si seleziona un pulsante e si fa clic su di esso, viene eseguita la funzione memorizzata.
- Se viene eseguito un clic del mouse su un pulsante selezionato e il mouse viene spostato prima di rilasciare il pulsante, la funzione memorizzata non viene eseguita.
- Cancellare la selezione: fare clic con il tasto sinistro del mouse in un'area esterna alla selezione.

4.4 Macro nell´Editor

Le macro possono essere eseguite nell'Editor tramite una barra degli strumenti (A pagina: 73) configurabile. Per fare questo, le macro sono collegate ai pulsanti nella barra degli strumenti **VBA** .

Le macro possono essere eseguite anche manualmente utilizzando l'Editor VBA.

Con Wizard è possibile eseguire automaticamente lavori di progettazione ricorrenti o creare interi progetti con la semplice pressione di un pulsante. Alcuni Wizard vengono forniti a titolo di esempio con zenon e possono essere ampliati e completate a piacere. Aiutano a creare progetti, importare ed esportare, creare variabili e molto altro ancora.. I dettagli si trovano nel capitolo Wizards

EDITOR-EVENTS

Gli eventi dell'Editor fanno parte del workspace di VBA e consentono di reagire agli eventi nella programmazione workspace, ad esempio per per Wizards oppure Trasporto Remoto. Per esempio:

- OnElementCreated
- OnElementDeleted
- OnElementDoubleClicked
- OnObjectCreated
- …

È possibile trovare tutti gli eventi e le informazioni su di essi nella guida di Object Model (A pagina: 158).

4.4.1 Barra degli strumenti "Lista macro"

Le macro create con VBA possono essere gestite usando la barra degli strumenti **Lista macro**.

Se si seleziona un filtro, questa selezione è visibile anche nella barra degli strumenti.

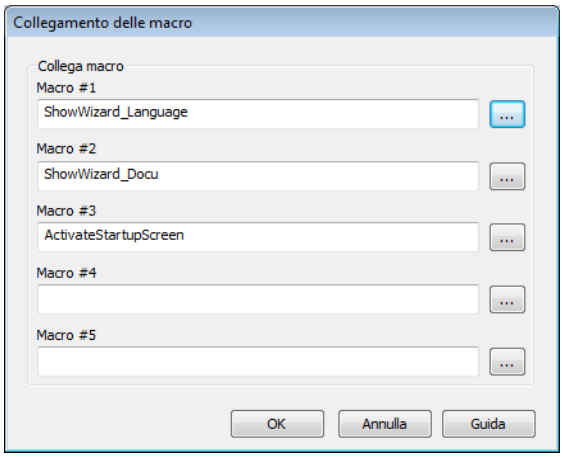
Simbolo	Descrizione
Aggiorna la lista delle macro VBA/VSTA	Carica nel menù a tendina della barra degli strumenti tutte le macro `Public Sub Name ()` contenute in **myWorkspace** e nei moduli.
Cerca macro	Ricerca di macro effettuata usando il campo di inserimento del menù a tendina, oppure selezionandone una fra quelle offerte dal menù a tendina. Quando si apre il menù a tendina, la sua dimensione viene adeguata a quella dell´elemento più largo.
Menù a tendina delle macro	Contiene tutte le macro caricate che potranno essere selezionate.
Esegui macro selezionata.	Esegue la macro selezionata nel menù a tendina.
Esegui la macro assegnata #<x>.	Esegue la macro assegnata al simbolo.
Assegna macro	Apre la finestra di dialogo per l'assegnazione delle macro. È possibile collegare un massimo di 5 macro ai simboli dall´1 al 5.
VBA	Filtra per macro VBA. Verranno visualizzate solamente le macro VBA.
VSTA	Filtra per macro VSTA. Verranno visualizzate solamente le macro VSTA.
Visualizza tutte le macro	Il filtro attualmente applicato viene rimosso e vengono visualizzate tutte le macro.
AZ	Ordina le macro in ordine ascendente, da 0 a 9 e dalla A alla Z.
ZA	Ordina le macro in ordine discendente, dalla Z alla A e da 9 a 0.
Opzioni barre degli strumenti	Cliccando sulla freccia si apre il sotto-menù: `Attiva`: la barra degli strumenti viene visualizzata. Se la barra degli strumenti non viene visualizzata, la si può attivare usando il menù **Visuale-> Toolbar**. Nota: in caso di posizionamento libero della barra degli strumenti (sganciata dall´Editor), le opzioni non vengono visualizzate. La barra degli strumenti può essere chiusa cliccando sul pulsante X.

Info

Se nella finestra di dialogo di collegamento delle macro non vengono visualizzate tutte la macro contenute in `myWorkspace`, eseguire la funzione **Aggiorna la lista delle macro VBA**.

4.4.2 Assegnazione di macro

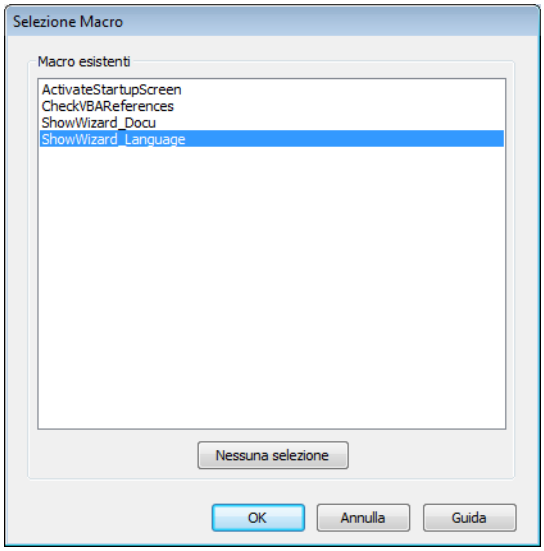
Le macro possono essere richiamate tramite un pulsante nella barra degli strumenti. In questo modo è possibile collegare fino a cinque macro. La finestra di dialogo per l'assegnazione delle macro si apre con il pulsante **Assegna macro**.



Opzione	Descrizione
Macro #	Il numero della macro corrisponde al pulsante nella barra degli strumenti. Cliccando sul pulsante **...** si apre la finestra di dialogo per la selezione della macro.
OK	Crea collegamenti con i pulsanti e chiude la finestra di dialogo.
Annulla	Annulla tutte le modifiche e chiude la finestra di dialogo.
Guida	Apre la guida online.

Selezione macro per l'assegnazione

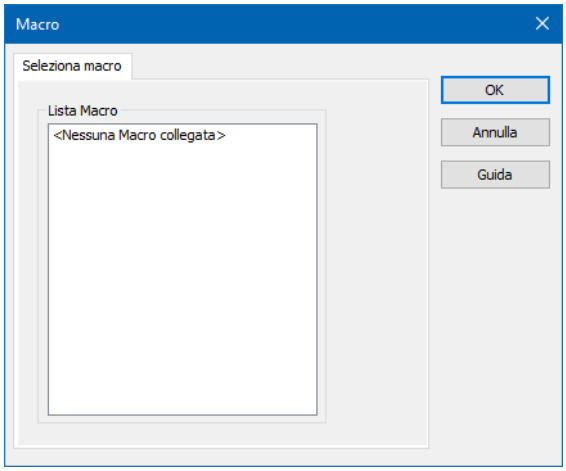
Parametro	Descrizione
Macro #	Il numero della macro corrisponde al pulsante nella barra degli strumenti. Cliccando sul pulsante **...** si apre la finestra di dialogo per la selezione della macro.
OK	Crea collegamenti con i pulsanti e chiude la finestra di dialogo.
Annulla	Annulla tutte le modifiche e chiude la finestra di dialogo.
Guida	Apre la guida online.



Parametro	Descrizione
Macro esistenti	Elenco delle macro che possono essere collegate.
Nessuna selezione	Elimina l'assegnazione esistente per questo pulsante.
OK	Assegna la macro selezionata al pulsante.
Annulla	Annulla tutte le modifiche e chiude la finestra di dialogo.
Guida	Apre la guida online.

Selezione macro per elementi

Gli elementi nelle immagini possono essere collegati con macro VBA. Si apre una finestra di dialogo per la creazione del collegamento.



Parametro	Descrizione
Lista Macro	Lista delle macro che possono essere collegate.
OK	Assegna la macro selezionata al pulsante.
Annulla	Annulla tutte le modifiche e chiude la finestra di dialogo.
Guida	Apre la guida online.

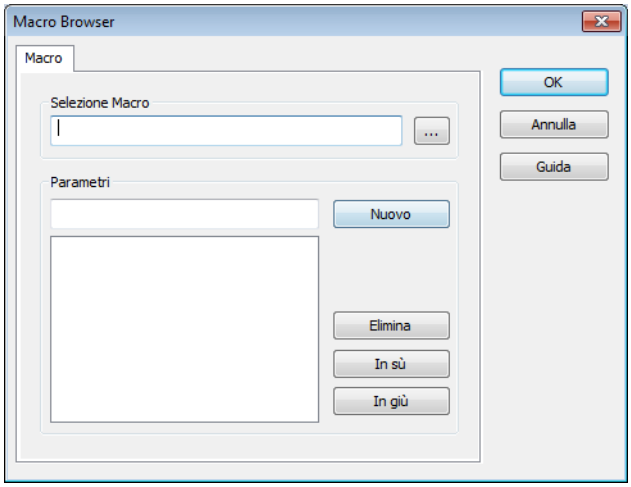
4.5 Funzioni in zenon

Nella finestra di dialogo **Seleziona funzioni**, si trovano le seguenti funzioni alla voce **VBA**.

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`. Pertanto, la funzione di zenon **Lanciare Macro-VBA** non viene più eseguita in script, in quanto VBA non è più in esecuzione in questo momento.
VBA-Visualizza dialogo Macro	Apre la finestra di dialogo macro VBA.

4.5.1 Lanciare Macro-VBA

Quando si seleziona la funzione "Lanciare Macro-VBA", viene visualizzata la seguente finestra di dialogo.



Sono disponibili le seguenti impostazioni:

Parametro	Descrizione
Selezione Macro	Apre la finestra di dialogo per la selezione della macro (vedere Selezione Macro (A pagina: 79)) Consiglio: elenca solo le macro VBA che corrispondono al numero di parametri definiti nella funzione Parametri (sotto).
Parametri	Immettere il valore desiderato (stringa) per un parametro.
Nuovo	Cliccando su questo pulsante si aggiunge il valore del parametro all'elenco dei parametri disponibili.
Elimina	Cliccare su questo pulsante per eliminare la voce selezionata dall'elenco dei parametri disponibili. È possibile eliminare solo una voce alla volta.
In su	Cliccare su questo pulsante per spostare la voce selezionata di una posizione verso l'alto. L´inserimento in questione verrà spostato di una posizione verso l´alto nella lista dei parametri.
In giù	Cliccare su questo pulsante per spostare l'immissione selezionata di una posizione verso il basso.. L´inserimento in questione verrà spostato di una posizione verso il basso nella lista dei parametri.

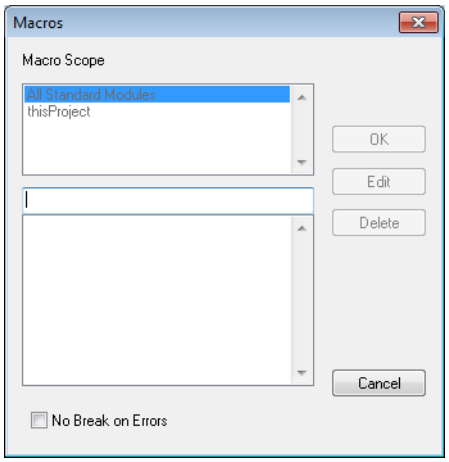
È possibile memorizzare stringhe per macro create con parametri. Queste stringhe vengono trasferite a Runtime come parametri individuali durante l'esecuzione della macro.

Info

È necessario assicurarsi che il numero di parametri della macro collegata corrisponda al numero di parametri creati in questa sede.

Selezione Macro

Dopo aver cliccato sul pulsante **...** viene visualizzata la seguente finestra di dialogo.



Selezionare la macro desiderata tra quelle disponibili, quindi fare clic su **OK**.

4.6 Sviluppare Wizard in VBA

A partire dalla versione 6, è possibile automatizzare la progettazione di progetti tramite procedure guidate. Ciò significa che le operazioni che devono essere ripetute spesso possono essere assegnate a una procedura guidata che esegue le azioni richieste, ad esempio la creazione di un progetto, la creazione di modelli e immagini secondo uno standard definito.

Un'altra applicazione per li Wizard è la modifica automatica di un progetto esistente, ad esempio la modifica delle proprietà degli elementi dinamici in tutte le immagini di un progetto esistente.

I Wizard si basano su Microsoft Visual Basic for Applications (VBA), e sul modello di oggetto di zenon.

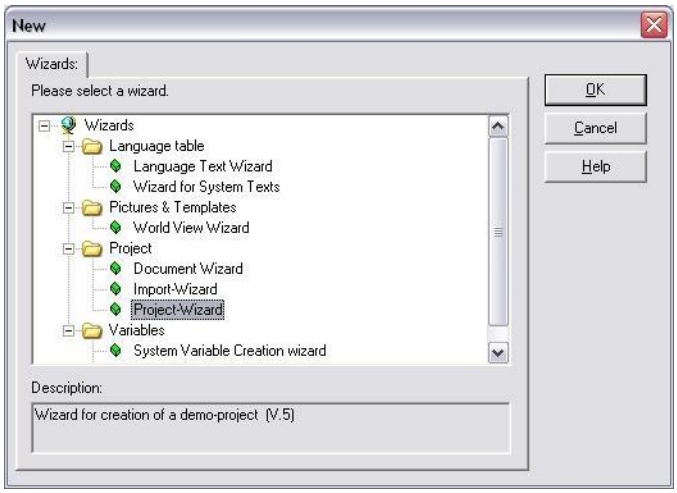
Al momento sono disponibili i seguenti Wizard:

- Wizard di progetto
- Wizard per l'importazione
- World view wizard
- Find VBA-Text Wizard
- Wizard for keyword-creation
- Wizard for keyword-translation
- Wizard for creating variables

I Wizard sono disponibili come file di codice sorgente VBA sul supporto di installazione. Nuovi Wizard possono essere integrati tramite l'ambiente VBA.

4.6.1 Utilizzo di un Wizard

La voce **Avvia Wizards in Editor** del menù **File** apre la finestra di dialogo di selezione dei Wizard. In questa finestra di dialogo vengono visualizzati tutti i Wizard disponibili, suddivisi per categorie.
Se i Wizard non dovessero contenere nessuna categoria, viene creato automaticamente l´inserimento **Non collegato**. Tutte le procedure guidate non collegate sono elencate sotto questa voce.



Si avvia un Wizard selezionandolo e cliccando su **OK**.

Info

I Wizard non supportano progetti multiutente.

4.6.2 Struttura di un Wizard

Un Wizard è un UserForm che viene memorizzato nel nodo specifico dell'applicazione. Questo UserForm di solito consiste in un elemento a più pagine in cui vengono visualizzati i singoli passi della procedura guidata.

Cliccando su un pulsante **Avanti**, si passa alla prossima pagina dell´elemento multipagina. Tutte le voci devono essere salvate temporaneamente; la creazione di oggetti, per esempio, modelli, immagini, .. può avvenire solo quando si clicca su **Termina** e si chiude il Wizard.

Info

Gli UserForm che devono essere utilizzati come procedure guidate, devono contenere alcuni public method che forniscono le informazioni al sistema di controllo tramite il Wizard. Se queste routine mancano in un UserForm, questo non viene trattato come un Wizard.

4.6.3 Integrazione in VBA

Le procedure guidate si trovano nel nodo **ZWorkspace** specifico per l'applicazione. Questo oggetto rappresenta il workspace caricato nell´Editor ed è disponibile solo nell´Editor di zenon.

Tutti gli oggetti di questo progetto VBA possono accedere al workspace corrente tramite l'oggetto **MyWorkspace**. Questo è sempre collegato al progetto attualmente attivo al quale si può accedere tramite la proprietà **ActiveDocument**.

Il contenuto dell'oggetto ZWorkspace si trova nel file `ZenWorkspace.vba`. Questo file viene copiato nella directory di installazione con la prima installazione della versione 6. Non verrà sovrascritto da aggiornamenti successivi. Maggiori informazioni sull'aggiornamento delle procedure guidate sono disponibili alla fine del tutorial.

4.6.4 Sviluppo di un Wizard

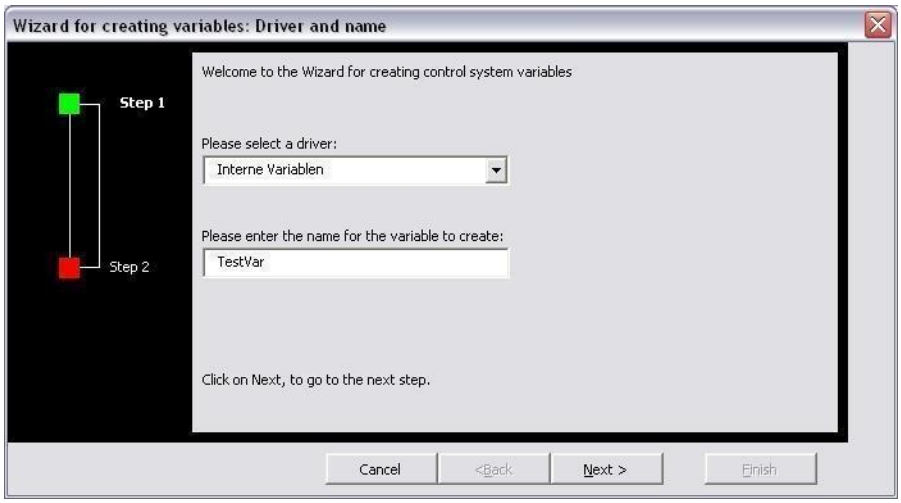
Questo tutorial crea un Wizard per la creazione di variabili per un driver configurato.

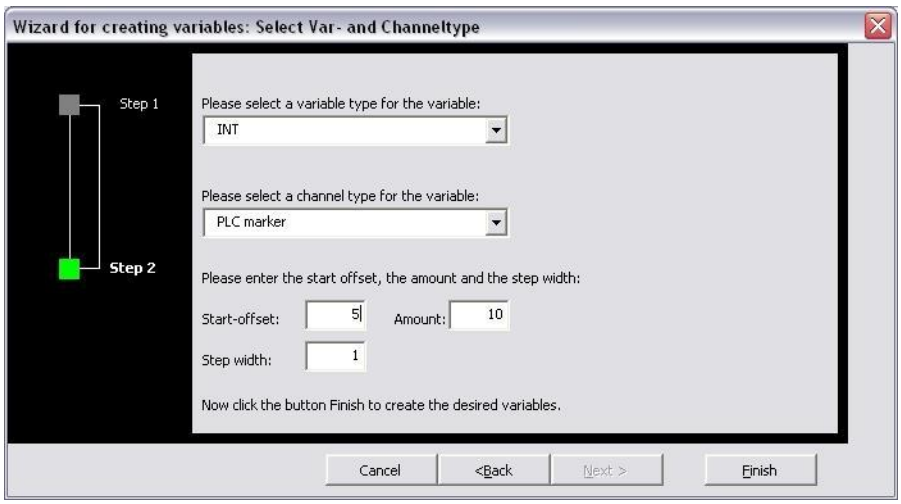
Aprire l´ambiente VBA dall´Editor di zenon e passare alla cartella **ZWorkspace/Formulare**. Questo file contiene le basi per lo sviluppo di una procedura guidata. Cambiare il nome di UserForm.

Se il Wizard non è disponibile nella cartella di cui sopra, è possibile importarlo scegliendo il comando **Importa file**.

Info

La conoscenza del modello oggetto di zenon e di VBA è necessaria per lo sviluppo di una procedura guidata. Questi argomenti non sono spiegati in questo tutorial.





Creare l'interfaccia mostrata sopra. Quindi passare al modulo codice di UserForm e scorrere fino alla fine del file. Qui si trovano i seguenti metodi:

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

  Restituisce il nome univoco della procedura guidata. Cambiare il contenuto in `Wizard for creating variables`

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

  Restituisce una breve descrizione visualizzata nella selezione Wizard. Modificare il contenuto in `Wizard for creating variables to a selected driver`

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

Restituisce la categoria del Wizard. Nella selezione Wizard, le procedure guidate vengono suddivise in categorie e visualizzate in una struttura ad albero. Modificare il contenuto in `Variables`.

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

  Visualizzare il Wizard nella selezione Wizard. Se questo metodo restituisce "False", la procedura guidata non viene visualizzata, ad esempio, perché non è ancora terminata. Cambiare il tipo di ritorno in `True`.

Questi metodi forniscono le informazioni sul Wizard richieste dal sistema di controllo. Si noti che un Wizard viene visualizzato nella “Selezione Wizard” solo se il metodo **IsZenOnWizard** restituisce `True`.

Passare all'evento **Initialize** di UserForm e cambiare il contenuto dell'array di stringhe **m_strCaption**. Poiché la nostra procedura guidata consiste di soli due passaggi, è possibile eliminare le altre assegnazioni.

Inserire le seguenti definizioni nell'area superiore del Code-Modul:

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

Creare un metodo per inizializzare il Combobox dei driver. Il compito della routine è quello di visualizzare tutti i driver configurati del progetto corrente 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
```

Inoltre, abbiamo bisogno di una routine che visualizzi tutti i tipi di variabile configurati del progetto in un Combobox.

```
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
```

Quando si apre il Wizard, il sistema analizza le variabili già configurate per cercare un offset di avvio libero per le variabili da creare. Questo viene fatto utilizzando il 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
```

Passare all'evento “Initialize" di UserForm e aggiungere le seguenti righe a questo metodo:

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

L'attribuzione a **txtStart** imposta l'offset di start proposto per le variabili da creare. Il metodo **InitializeDriver** viene utilizzato per popolare il combobox con i driver configurati..

Creare un evento "change" per il combobox dei driver e inserire il codice seguente. Dopo aver selezionato un driver, vengono determinati i tipi di variabili. L'oggetto driver selezionato viene memorizzato nella variabile **m_obDriver** per un utilizzo successivo.

```
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
```

Creare un trattamento evento **Change** per VarType-Combobox e inserire il codice seguente. Il tipo di variabile selezionato viene memorizzato nella variabile m_obVarType per un utilizzo successivo.

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

Nun müssen wir nur mehr die Ereignisroutine zum Erstellen der Variablen mit den getroffenen Einstellungen erstellen. Questo 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
```

Al termine della procedura guidata, viene verificata la validità delle impostazioni effettuate. In caso contrario, viene visualizzato un messaggio di errore e viene chiesto all'utente di correggere l'inserimento errato.

Se le impostazioni sono corrette, vengono create le variabili. Le variabili sono denominate con un nome e un indice. Se nel progetto esiste già una variabile con lo stesso nome, viene determinato il successivo indice libero. Nel nostro esempio di codice viene sempre creata una variabile di tipo Merker-PLC. Ad ogni passaggio del loop, l'offset della variabile aumenta.

4.6.5 Aggiornare i Wizard.

Per aggiornare i Wizard:

1. Nel menù **Extras**, selezionare l´inserimento **Aggiorna Wizards VBA/VSTA in Editor...**.

   Viene visualizzata una finestra di dialogo per l'aggiornamento dei Wizard esistenti.
2. Selezionare il Wizard desiderato
3. Avviare l´update cliccando su **Inizio aggiornamento**

Se un Wizard o una classe sono già presenti nel workspace, viene visualizzato un messaggio di avviso.

> **Attenzione**
>
> *I Wizard già esistenti vengono sovrascritti durante l'aggiornamento. Le modifiche singole effettuate nel Wizard vanno quindi perse.*

4.7 Domande ricorrenti

Questo capitolo risponde ad alcune delle domande più frequenti.. Ulteriori soluzioni sono disponibili anche online in COPA-DATA Forum utenti (http://www.copadata.com/forums/).

4.7.1 Perché il pulsante rimane premuto?

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

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

4.7.2 La macro non viene eseguita al 1° clic

La soluzione corrisponde a quella della questione: Perché il pulsante rimane premuto? (A pagina: 88):

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

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

4.7.3 Le macro non funzionano più a RT?

Questo effetto si può verificare se l'Editor VBA è aperto a Runtime e quindi si preme Stop/Start per arrestare/avviare VBA. In questo caso, gli oggetti (variabili online, oggetti immagine, ...) diventano non validi, poiché perdono il loro riferimento quando si esegue una nuova inizializzazione.

4.7.4 Windows CE e VBA

VBA è disponibile nell'Editor CE per le procedure guidate. VBA non è supportato a Runtime. Dettagli sull'Editor sono disponibili nel capitolo Creare progetti per CE.

4.8 Esempi

Ecco alcuni esempi di VBA.

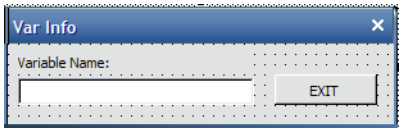
4.8.1 MouseEvents and ActiveX Control initialization

```
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 Visualizzazione di informazioni variabili

Visualizzare il nome variabile per l'elemento cliccato:

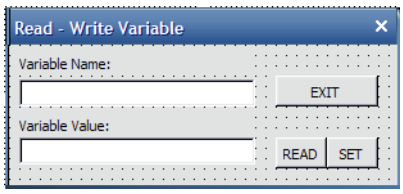



```
Option Explicit

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 Lettura e scrittura di valori variabili

Lettura e backup di valori da una variabile:

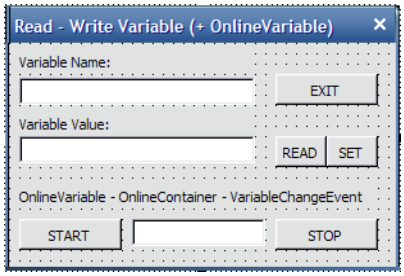



```
Option Explicit

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 Lettura e scrittura di variabili e implementazione di variabili online

Leggere le informazioni sulle variabili, scrivere i valori e implementare le variabili online:



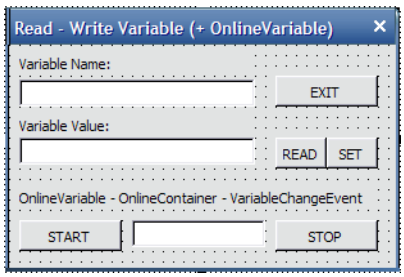

```
Option Explicit

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 della finestra di 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
```

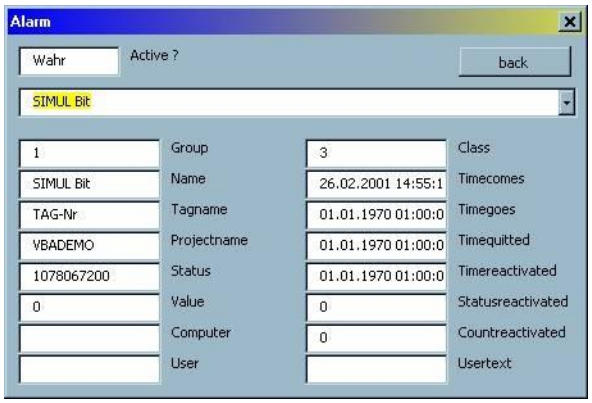
4.8.6 Alarm – Events and ActiveX Control handling

```
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 modulo frmAlarm è possibile selezionare un allarme in memoria in un combobox. Dopo la selezione, tutti i dati relativi all'allarme vengono inseriti nelle sottostanti caselle di testo (gruppo, classe, variabile,...).

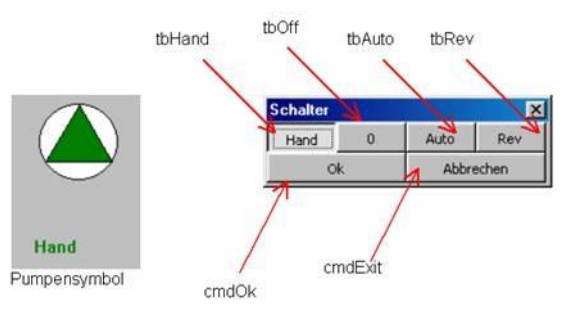
Per visualizzare frmAlarm, utilizziamo un evento macro indipendente, poiché non abbiamo bisogno di alcun riferimento a un elemento in questo esempio.

```
Sub Alarm ()
frmAlarm.Show
End Sub
'La macro viene eseguita utilizzando la funzione "esegui macro.."
'Quando il modulo viene attivato, viene inizializzato e viene eseguita la seguente
procedura. Questa procedura assicura che il combobox creato nel modulo riceva tutti gli
```

```
allarmi attualmente 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


'Quando si seleziona un allarme del Combobox, l'evento Change del Combobox reagisce.
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 Impostare un interruttore (lavorare con variabili di processo)



In questo esempio si disegna una pompa composta da un cerchio e da un triangolo. Il triangolo è definito come il proprio simbolo. Un elemento multibin è posizionato sopra di esso e collegato con tre variabili bit-merker

Inoltre, si determina quale colore assume il triangolo al centro quando cambia il valore della variabile.

In primo luogo, l'elemento Multibin è collegato ad una macro, che fa apparire un form frmSwitch.

Con il form frmSwitch è possibile modificare il valore delle tre variabili bit-merker.

Info

Solo una delle tre variabili può contenere il valore 1. (cioè se una variabile è impostata su 1, le altre due devono essere impostate su 0)

Per utilizzare più volte questa macro in un progetto con riferimento a variabili diverse, è possibile collegare all'elemento Multibin solo variabili bit-merker, che già contengono lo stato della pompa che controllano nel nome della variabile.

Per es.:
Variable_Auto
Variable_Hand
Variable_Revi

Info

Le estensioni _Auto, _Hand e _Revi sono fisse nel codice sorgente dell'esempio.

Queste estensioni a 5 cifre dei nomi delle variabili possono ora essere usate per determinare quale variabile è impostata su 1 quando si fa clic su uno dei pulsanti di commutazione e quale su 0.

Nella macro LeftClickUp_Schalter viene richiamata una subroutine FindVariable in form frmSchalter, a cui viene passato l'elemento cliccato 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 Globale Variabile del Modulo:

```
Dim cmdLast As ToggleSchaltfläche
Dim strHand As String
Dim strAuto As String
Dim strRevi As String
Nella subroutine FindVariable vengono prima verificate tutte le variabili offerte
dall'elemento specificato.
```

A seconda dell'estensione (_Auto, _Hand o _Revi), i nomi delle variabili sono assegnati alle variabili stringa sopra dichiarate.

Viene inoltre determinato lo stato della variabile e, a seconda del valore (1 o 0), viene premuto o meno anche il relativo pulsante di commutazione.

Quando si apre il modulo frmSwitch, il nome del pulsante di commutazione attualmente premuto viene scritto in una variabile stringa. Se l'utente decide di annullare le variazioni di valore, i valori originali vengono ripristinati.

```
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 funzione autogenerata VarExists verifica solo se esistono anche le variabili collegate, in caso contrario viene emessa 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 "Annulla", la modifica del valore viene annullata e lo stato originale viene ripristinato.

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

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

Quando si preme un pulsante di commutazione, non è possibile premerne un altro.

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

L'evento di clic di ogni pulsante di commutazione verifica se non è premuto e se la variabile esiste. Se entrambi gli stati sono veri, i valori vengono passati alle variabili collegate.

5. VSTA

Con **Visual Studio Tools for Applications (VSTA)**, le funzionalità del Runtime e dell´Editor di zenon possono essere estese in modo indipendente tramite la programmazione .NET.

> **Attenzione**
>
> *Per VSTA è possibile integrare solo `Assemblies` (DLLs) fino a .NET Framework Version 3.5.*

VSTA è disponibile anche su zenon Web Client con alcune restrizioni.

VSTA fornisce ambienti di sviluppo separati per **Workspace** e **progetto**. Solo uno di essi può essere utilizzato alla volta. All'avvio dell´Editor VSTA, qualsiasi altro ambiente di sviluppo VSTA già aperto verrà chiuso.

Per aprire l'Editor VSTA per il workspace:

1. Premere la combinazione di tasti `Alt+F10`.
   Viene visualizzata una finestra di dialogo che contiene un avviso.

   Viene visualizzato il codice del workspace e di tutti i progetti caricati.

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

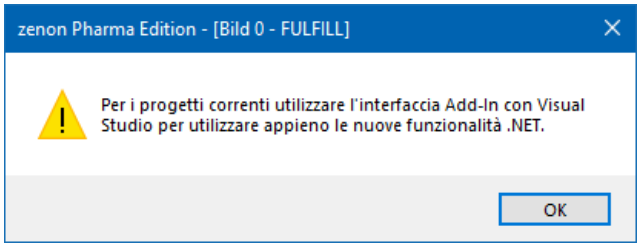
1. Passare al nodo **Scripting**
2. Espandere la visualizzazione di questo nodo su `[+]`.
   Viene visualizzata la struttura del nodo.
3. Cliccare con il tasto destro del mouse su **VSTA**.
4. Nel menù contestuale, selezionare **Apri Editor VSTA con ProjectAddin**.
   Viene visualizzata una finestra di dialogo che contiene un avviso.

   Cliccare sul pulsante **OK** per proseguire con la progettazione VSTA.
   L'Editor VSTA si apre per il progetto attualmente caricato.

> **Info**
>
> *Quando le macro VBA vengono modificate nell'Editor:*
>
> - I file Runtime vengono compilati e trasferiti al Runtime.
> - Viene eseguito un reload del Runtime.
> - Anche gli elementi del VSTA vengono ricaricati, anche se non sono state apportate modifiche al VSTA.

MESSAGGIO DI AVVERTIMENTO

Prima dell´apertura dell´Editor VSTA in zenon viene visualizzata una finestra di dialogo di avvertimento. Fare clic sul pulsante **...** per aprire l'Editor VSTA.



5.1 Nozioni di base

VSTA è un set di strumenti Microsoft basato sulla tecnologia.NET. È necessario avere una conoscenza di base della programmazione orientata agli oggetti .NET e C#/Visual Basic.NET.

Attenzione

A Runtime, bisogna tenere presente quanto segue: Le DLL referenziate non possono essere sostituite quando caricate. Per poter aggiornare un file referenziato, si deve prima chiudere il Runtime.

5.1.1 Configurazione dell'ambiente VSTA

Per impostazione predefinita, il supporto per VSTA è già abilitato in zenon. Quando si disattiva il supporto VBA, anche l'ambiente VSTA non è più disponibile.

Con la seguente voce del file **zenon6.ini** è possibile attivare o disattivare manualmente l'ambiente VSTA:

Attivare VSTA	Disattivare VSTA
[VSTA] **ON**=1	**[VSTA]** **ON**=0

Il supporto per VBA viene abilitato o disabilitato come segue

Attivare VBA	Disattivare VBA
[VBA] **EIN**=1	**[VBA]** **EIN**=0

L'ambiente di sviluppo per VSTA è quindi disponibile in zenon.

Info

VSTA permette lo sviluppo di progetti nei linguaggi di programmazione C# e Visual Basic.NET. Il linguaggio predefinito per i progetti VSTA nell'Editor è C#. La seguente voce può essere usata per cambiare la lingua in Visual Basic.NET::

[VSTA]

CSHARP*=0*

5.1.2 Accesso al modello oggetti in zenon

In VSTA è possibile accedere al modello oggetto di zenon utilizzato anche in VBA. Il modello a oggetti offre le stesse funzionalità in entrambi gli ambienti di sviluppo.

Attenzione

A causa delle restrizioni nella denominazione degli oggetti VSTA, sono state apportate alcune modifiche al modello oggetto. Questi dati sono riportati nella tabella che segue

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 avviene in VSTA tramite l'oggetto `this` e sostituisce quindi l'oggetto `MyWorkspace`. I seguenti metodi e oggetti sono identici. Nel metodo seguente, viene creato un modello denominato "TemplateName" in zenon.

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

Info

A differenza di VBA, VSTA è sensibile alle maiuscole e alle minuscole e utilizza le parentesi per nome di funzione.

Per richiamare il metodo in zenon, il progetto deve essere salvato e compilato nel seguente modo:

1. Cliccare su File -> Save MyWorkspace.cs per salvare il progetto.
2. Cliccare su Build -> Build WorkspaceAddin per compilare il progetto.

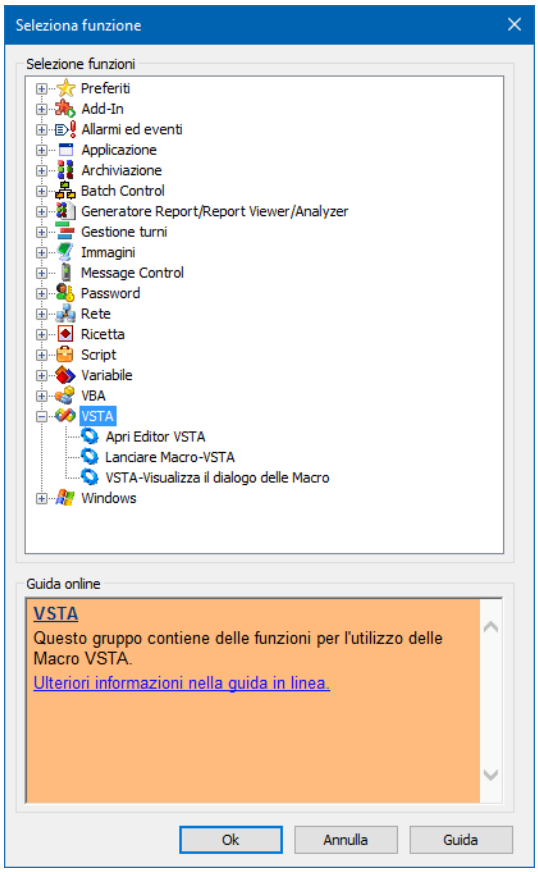
Il metodo è quindi disponibile come macro nella barra degli strumenti macro di VBA nell'Editor di zenon. Se non tutte le macro contenute in MyWorkspace sono disponibili per la selezione nella finestra di dialogo di assegnazione delle macro, la funzione "Ricarica lista macro VBA" deve essere eseguita nella barra degli strumenti.

Info

Le macro VSTA con parametri, ad esempio `Public void MacroWithParam(string mString)`*, non sono supportate e non sono disponibili nella barra degli strumenti macro.*

5.1.3 Funzioni in zenon

In zenon sono state sviluppate le seguenti funzioni per VSTA. Queste funzioni si trovano nel nodo **VSTA**.

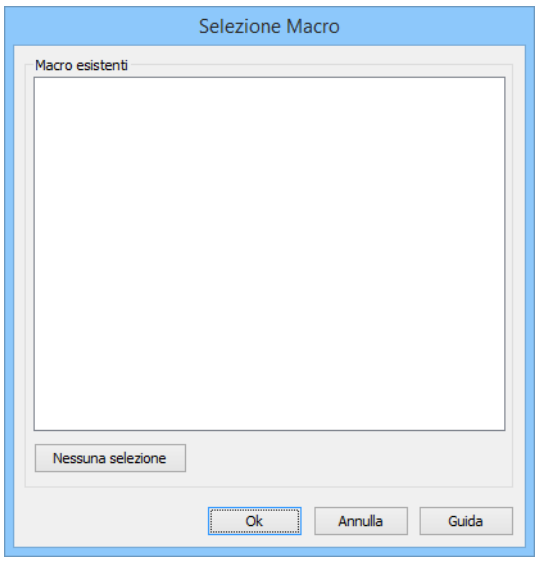


Parallelamente alle funzioni VBA esistenti, funzioni simili sono state implementate per VSTA:

Nome funzione	Descrizione
Apri Editor VSTA	Apre l´Editor VSTA a Runtime.
Lanciare Macro-VSTA	Nell'Editor è possibile selezionare una macro VSTA che viene avviata a Runtime all'esecuzione della funzione. Nota: le macro VSTA con parametri, ad esempio `Public void MacroWithParam(string mString)`, non sono supportate. Non sono disponibili né durante la progettazione nell'Editor, né all'avvio della funzione a Runtime
VSTA-Visualizza il dialogo delle Macro	A Runtime viene visualizzata una finestra di dialogo nella quale è possibile visualizzare, selezionare ed eseguire le macro VSTA esistenti.

Lanciare Macro-VSTA

Finestra di dialogo per la configurazione della funzione **Lanciare Macro-VSTA**.



Parametro	Descrizione
Macro esistenti	
Lista delle macro VSTA	Elenca tutte le macro VSTA esistenti. Selezione di una macro dall'elenco con un clic. Nota: fare doppio clic per eseguire la macro selezionata.
Nessuna selezione	Annulla la selezione e chiude la finestra di dialogo.
OK	Esegue la macro selezionata e chiude il dialogo.
Annulla	Annulla la selezione e chiude la finestra di dialogo.
Guida	Apre la guida online.

5.1.4 Eseguire il debug di un Add-in VSTA

È possibile utilizzare l'Editor VSTA per eseguire il debug di Add-in scritti dall'utente. Bisogna tenere presente che si può eseguire il debug di Projekt Add-ins solamente nel Runtime di zenon, di Workspace-AddIns solo nell´Editor di zenon.

Una sessione di debug viene avviata mediante il menù Debug - Start Debugging. Come nell'Editor VBA, breakpoint possono essere impostati facendo clic con il pulsante sinistro del mouse sulla barra dei breakpoint grigia a sinistra della riga in questione.

Info

Quando si esegue il debug di Add-in a Runtime, tenere presente quanto segue:

I file Runtime modificati in zenon devono essere generati di nuovo prima di avviare il Debugger.

5.1.5 Events in VSTA

Visto che un AddIn viene terminato quando si compila un codice modificato, si avvia una sessione di debug o si termina una sessione di debug, sono stati implementati in VSTA eventi corrispondenti. Questi consentono, ad esempio, la valutazione e il rilascio di un riferimento oggetto e la memorizzazione dei dati esistenti.

Per ogni chiusura esistono due Events. Il primo evento viene avviato poco prima della chiusura, il secondo dopo l'inizio di una nuova sessione Add-in.

Event	Descrizione
`OnPreVSTADebugStart`	Attivato poco prima dell'avvio di una sessione di debug. All'avvio, un AddIn attivo viene rimosso, i riferimenti devono essere rilasciati e, se necessario, i dati esistenti devono essere salvati.
`OnVSTADebugStart`	Attivato dopo l'avvio di una sessione di debug.
`OnPreVSTADebugStop`	Attivato poco prima dell'arresto di una sessione di debug. Quando una sessione di debug viene interrotta, l´Add-in viene rimosso, i riferimenti devono essere rilasciati e i dati esistenti devono essere salvati, se necessario.
`OnVSTADebugStopped`	Attivato dopo l'arresto di una sessione di debug.
`OnPreVSTAUpdate`	Si attiva prima che l´Add-ins venga rimosso, se una nuova versione dell´AddIn è stata creata con successo.
`OnPostVSTAUpdate`	Attivato quando si carica una nuova versione dell'Add-in.

5.1.6 Backup di progetti VSTA

A Runtime, i progetti VSTA vengono compressi automaticamente in formato ZIP al momento della generazione dei file Runtime e inclusi in backup del workspace.

Nell'Editor, invece, i progetti VSTA devono essere salvati manualmente.
I progetti VSTA dell´Editor si trovano nella cartella
C:\ProgramData\COPA-DATA*version*\VSTAWorkspace\.

5.2 Creazione di un progetto VSTA

Come VBA, VSTA offre la possibilità di creare progetti (estensioni) sia per l'Editor che per il Runtime. In linea di principio, i progetti nell'Editor sono implementati nel linguaggio di programmazione C#. Sia C# che Visual Basic.NET sono disponibili per il Runtime e l'Editor.

Attenzione

Per VSTA è possibile integrare solo `Assemblies` (DLLs) fino a .NET Framework Version 3.5.

Info

Nell´Editor di VSTA è possibile visualizzare un solo progetto alla volta. Inoltre, può essere attiva solo un'istanza alla volta dell'Editor VSTA. All'avvio dell'Editor VSTA, ogni altra istanza esistente viene chiusa.

5.2.1 Progetti VSTA nell´Editor

Quando si crea un progetto per l'Editor di zenon, viene caricato un Add-in VSTA per il workspace. Per modificare l´Add-In, si deve aprire l´editor VSTA mediante File - Apri l´Editor VSTA.... L´interfaccia utente dell´Editor VSTA è identica a quella dell'ambiente di sviluppo di Visual Studio di Microsoft.

Info

La guida dell'Editor VSTA può essere richiamata tramite il menù Help / Contents . Questa guida fornisce una panoramica delle funzioni dell'Editor, delle caratteristiche del Framework.NET e della programmazione in Visual Basic.NET e C#.

L´Add-In VSTA consiste fondamentalmente della classe **`MyWorkspace`**. È possibile aggiungere i propri metodi a questa classe. Per impostazione predefinita, la classe contiene i due metodi seguenti:

Funzione	Descrizione
`MyWorkspace_Startup`	Viene eseguito automaticamente all'avvio di zenon, dopo la creazione di una build e all'avvio di una sessione di debug.
`MyWorkspace_Shutdown`	Viene eseguito automaticamente quando si chiude zenon, prima di creare una nuova build e quando si termina una sessione di debug.

Attenzione

I nomi dei metodi possono iniziare solo con `Macro` (ad esempio Macro1, MacroVSTA) , non possono contenere parametri e devono essere definiti come `Public` . Inoltre, il nome della classe e altri metodi ed eventi creati da VSTA non possono essere modificati.

Per richiamare il metodo in zenon, il progetto deve essere salvato e compilato nel seguente modo:

1. Cliccare su File -> Save MyWorkspace.cs per salvare il progetto.
2. Cliccare su Build -> Build WorkspaceAddin per compilare il progetto.

Il metodo è quindi disponibile come macro nella barra degli strumenti macro di VBA nell'Editor di zenon. Se non tutte le macro contenute in MyWorkspace sono disponibili per la selezione nella finestra di dialogo di assegnazione delle macro, la funzione "Ricarica lista macro VBA" deve essere eseguita nella barra degli strumenti.

5.2.2 Progetti VSTA a Runtime

Per creare un progetto VSTA per il Runtime, è necessario richiamare l'ambiente VSTA.
Per farlo, procedere nel modo seguente:

1. Nel Manager di progetto, selezionare la voce **Scripting**.
2. Aprire il menù contestuale di **VSTA**.
3. Cliccare su **Apri Editor VSTA...**.

   Nota: quando viene aperto per la prima volta, viene visualizzata una finestra di dialogo di selezione. In questa finestra di dialogo, selezionare la lingua di programmazione.

   Viene creato un progetto con il linguaggio di programmazione desiderato.

> **Info**
>
> *Il linguaggio di programmazione non può essere modificato in un secondo tempo. Questa finestra di dialogo viene quindi visualizzata solo la prima volta che si apre l´Editor.*

In questo progetto, zenon crea una classe chiamata `ThisProject` , che contiene i due metodi seguenti:

Funzione	Descrizione
`ThisProject_Startup`	Viene eseguito automaticamente all'avvio del Runtime.
`ThisProject_Shutdown`	Viene eseguito automaticamente alla chiusura del Runtime.

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

> **Attenzione**
>
> *I nomi dei metodi possono iniziare solo con `Macro` (ad esempio Macro1, MacroVSTA) , non possono contenere parametri e devono essere definiti come `Public` . Inoltre, il nome della classe e altri metodi ed eventi creati da VSTA non possono essere modificati.*

Tutte le funzionalità Runtime sono accessibili tramite il modello oggetto zenon. Le funzionalità specifiche dell'Editor non possono essere utilizzate, come in VBA.

Il Runtime di zenon viene avviato automaticamente all'avvio del Debugger. Per ulteriori informazioni, vedere Eseguire il debug di un Add-in VSTA (A pagina: 109).

Info

La guida dell'Editor VSTA può essere richiamata tramite il menù Help / Contents . Questa guida fornisce una panoramica delle funzioni dell'Editor, delle caratteristiche del Framework.NET e della programmazione in Visual Basic.NET e C#.

5.2.3 Sviluppo di Wizard in VSTA

Come VBA (A pagina: 80), l'ambiente VSTA consente di sviluppare i propri Wizard.

Per accedere al modello oggetto zenon in un form, un riferimento ad esso deve essere passato al form. A questo scopo, viene creato un metodo nella classe `MyWorkspace` . L'esempio seguente istanzia un form con il nome `Wizard` e richiama il metodo `ZenonInstance` con un riferimento al modello oggetto zenon come parametro. La chiamata `ShowDialog()` visualizza il form della procedura guidata.

```
public void Macro1()
{
Form1 Wizard = new Form1();
Wizard.ZenonInstance(this.Application);
Wizard.ShowDialog();
}
```

Una variabile member deve essere creata nel codice del form che adotta il modello oggetto zenon.

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

Infine, viene creato il metodo `ZenonInstance` . Questo metodo accetta il riferimento al modello oggetto e lo imposta sull'oggetto `m_Zenon`.

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

Ora si possono sviluppare classi e metodi propri nel form che accedono al modello oggetto. Tutti i metodi, gli oggetti e gli attributi sono a disposizione tramite l´oggetto `m_Zenon`.

5.3 Esempi

Ecco alcuni esempi di come usare il VSTA sia nel Runtime che nell'Editor.

5.3.1 Creazione di variabili nell´Editor di zenon

In questo esempio, viene aperto un file di testo e il contenuto viene utilizzato per creare 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 prima il driver interno nell'Editor di zenon. Viene quindi visualizzata una finestra di dialogo di selezione file in cui l´utente deve selezionare il file di testo. Poi il metodo legge il file di testo e crea le variabili. Il metodo **GetDataType** è necessario per determinare e assegnare il tipo di dati corrispondente durante la creazione delle variabili.

C#-Code

```
using System;
using System.Windows.Forms;
using System.IO;
using zenOn;

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 IVarType GetDataType(string vType)
{
//gets the corresponding vartypes for bool, int, real and strings
IVarType retType;
switch (vType)
{
case "BOOL":
retType = this.ActiveDocument.VarTypes().Item("BOOL");
break;
case "INT":
retType = this.ActiveDocument.VarTypes().Item("INT");
```

```
break;
case "REAL":
retType = this.ActiveDocument.VarTypes().Item("REAL");
break;
case "STRING":
retType = this.ActiveDocument.VarTypes().Item("STRING");
break;
default:
retType = this.ActiveDocument.VarTypes().Item("INT");
break;
}
return retType;
}

//Reads a defined text file and creates corresponding variables on the zenon inte
rn driver
public void MacroCreateVariablesFromFile()
{

//create objects that will take the intern driver and the variable type
IDriver zenonInternDriver = null;
//search for the Intern driver and throw exception if no driver was found
try
{
for (int driverCount = 0; driverCount < this.ActiveDocument.Drivers().Count; driv
erCount++)
{
if (this.ActiveDocument.Drivers().Item(driverCount).Name == "Intern")
{
zenonInternDriver = this.ActiveDocument.Drivers().Item(driverCount);
break;
}
}
}
catch (Exception driverEx)
{
MessageBox.Show("Unable to find zenon 'Intern' driver. Error: " + driverEx.Messag
e);
throw;
```

```
}
this.ActiveDocument.Variables().DoAutoSave(false);
try
{
OpenFileDialog varFileSelect = new OpenFileDialog();
String[] varLine = new String[2];

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

//read in line by line, split the lines when a ',' occurs and create variables
while ((line = importReader.ReadLine()) != null)
{
varLine = line.Split(new Char[] { ',' });
this.ActiveDocument.Variables().CreateVar(varLine[0], zenonInternDriver, tpKanalt
ypes.tpSystemVariable, GetDataType(varLine[1]));
}
importReader.Close();
}
}
catch (Exception fileEx)
{
MessageBox.Show("An error occured while opening the file: " + fileEx.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

Questo esempio mostra come è possibile accedere alla finestra di emissione dell'Editor di zenon tramite VSTA. Il metodo chiamato `Macro1` legge le immagini di processo create nel progetto, determina il modello corrispondente e determina tutti i driver disponibili e i loro nomi.

C#-Code

```
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 MacroPrintDebugInformation()
{
string picName = string.Empty;
string corTemp = string.Empty;
string driverName = string.Empty;
string driverDescription = string.Empty;

//print start string into output window
```

```
this.Application.DebugPrint(" --------------START--------------", zenOn.tpDebugPr
intStyle.tpMsg);

//go through all pictures and print name and used template into output window

for (int i = 0; i < this.ActiveDocument.DynPictures().Count; i++)
{
picName = this.ActiveDocument.DynPictures().Item(i).Name;
corTemp = this.ActiveDocument.DynPictures().Item(i).get_DynProperties("Template")
.ToString();
this.Application.DebugPrint(" Picture '" + picName + "' uses Template '" + corTem
p + "'", zenOn.tpDebugPrintStyle.tpMsg);
}

//print separator string into output window

this.Application.DebugPrint(" ------------------------------", zenOn.tpDebugPrint
Style.tpMsg);

//go through all drivers and print name and description into output window

for (int i = 0; i < this.ActiveDocument.Drivers().Count; i++)
{
driverName = this.ActiveDocument.Drivers().Item(i).Name;
driverDescription = this.ActiveDocument.Drivers().Item(i).Identification;
this.Application.DebugPrint(" Driver '" + driverName + "' has description '" + dr
iverDescription + "'", zenOn.tpDebugPrintStyle.tpMsg);
}
//print end string into output window
this.Application.DebugPrint(" ---------------END---------------", zenOn.tpDebugPr
intStyle.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 Lettura di variabili in zenon tramite Regular Expressions

Nell'esempio seguente, le variabili zenon vengono lette in un progetto Runtime e memorizzate in un file di testo locale.

Tramite Regular Expressions vengono lette solo le variabili i cui nomi iniziano con 3 numeri e un successivo trattino basso (per esempio "001_var" o "234_xyz"). All'utente viene quindi richiesto di selezionare una cartella In questa cartella viene creato un file di testo con un nome che dipende dal tempo. Il nome, l´identificazione e il valore corrente di tutte le variabili applicabili sono memorizzati in questo file, separati da un punto e virgola.

Info

*Potrebbe essere necessario aggiungere manualmente dei riferimenti per eseguire l'esempio nel Runtime di zenon. Per farlo, aprire il menù contestuale nel Project Explorer e cliccare su **Add Reference...**.*
I rifermenti necessarie in questo esempio sono:

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

C#-Code

```
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 MacroGetFilteredVariables()
{
string filename = string.Empty;
string name = string.Empty;
string description = string.Empty;
string value = string.Empty;

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

//create stream writer to the .txt file
StreamWriter matchedVariables = new StreamWriter(filename, 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 val
ue
if (match.IsMatch(this.Variables().Item(i).Name))
{
name = this.Variables().Item(i).Name;
description = this.Variables().Item(i).Tagname;
value = this.Variables().Item(i).get_Value(0).ToString();
//write information to the .txt file
matchedVariables.WriteLine(name + ";" + description + ";" + value);
}
}
```

```
//close stream
matchedVariables.Close();
}
catch (Exception ex)
{

MessageBox.Show("An error occured -> " + ex.Message);
throw;
}
}
private string FolderSelection(String caption)
{
string selectedPath = string.Empty;
//create a dialog for selecting the output folder
FolderBrowserDialog folderSelect = new FolderBrowserDialog();
folderSelect.Description = caption;
try
{
if (folderSelect.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;
selectedPath = folderSelect.SelectedPath + "\\zenonVar" + currentTime.ToFileTime(
) + ".txt";
}
}
catch (Exception ex)
{
MessageBox.Show("An error occured: " + ex.Message);
throw;
}
return selectedPath;
}
#region VSTA generated code
private void InternalStartup()
{
this.Startup += new System.EventHandler(ThisProject_Startup);
```

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

5.3.4 Conversione dei colori tra.NET e zenon

Nell'API di zenon, i colori vengono memorizzati come tipi di dati numerici. I seguenti esempi di codice possono essere utilizzati per convertire i colori in.NET (o viceversa) in VSTA; inoltre si mostra la gestione delle tavolozze di colori.

CLASSE HELPER PER CONVERTIRE I COLORI.NET IN ZENON E VICEVERSA.

C#-Codice

```
namespace WorkspaceAddin
{
class ZenonColorHelper
{
// convert .net Color to zenon API color
public static uint Color2Api(System.Drawing.Color color)
{
// zenon stores as ABGR, where A is the highest significant bit
return (uint)((color.A << 24) | (color.B << 16) | (color.G << 8) | (color.R << 0));
}

// convert zenon API color to .net Color
public static System.Drawing.Color Api2Color(uint apiColor)
{
// zenon stores as ABGR, where A is the highest significant bit
byte a = (byte)(apiColor >> 24);
byte b = (byte)(apiColor >> 16);
byte g = (byte)(apiColor >> 8);
byte r = (byte)(apiColor >> 0);

return System.Drawing.Color.FromArgb(a, r, g, b);
}
```

```
// convert hex color string (#FFFFFF) to zenon API color
public static uint ColorString2Api(string colorString)
{
System.Drawing.Color color = System.Drawing.ColorTranslator.FromHtml(colorString);
return Color2Api(color);
}

// convert zenon API color to hex color string (#FFFFFF)
public static string Api2ColorString(uint apiColor)
{
System.Drawing.Color color = Api2Color(apiColor);
return System.Drawing.ColorTranslator.ToHtml(color);
}

// extracts the index in the color palette, which is encoded in the zenon API color
// returns -1 if the given color does not represent a color palette index
public static int GetColorPaletteIndex(uint apiColor)
{
System.Drawing.Color color = Api2Color(apiColor);
if (color.A == 128) // alpha value == 128 -> ind from color palette
{
// G defines the nr of 8bit chunks, R is the actual index
return color.G * 256 + color.R;
}

return -1;
}

// encodes the index in the color palette into a zenon API color
public static uint EncodeColorPaletteIndex(int index)
{
int chunks = (int)(index / 256.0);
int idx = (int)(index % 256);

System.Drawing.Color color = System.Drawing.Color.FromArgb(128, idx, chunks, 0);
return Color2Api(color);
}
```

```
}
}
```

ESEMPIO DI UTILIZZO DELLA CLASSE HELPER

C#-Codice

```
using zenOn;

public void Macro_ColorHandlingSample()
{
// first we need a project to work with

IProject project = Application.MyWorkspace.ActiveDocument;
if (null == project)
{
System.Diagnostics.Debug.Print("Error! No project loaded / activated");
return;
}

// create a sample picture with a filled rectangle in it (color defined directly in
the element)

// create sample frame for picture if it not yet exists
if (null == project.Templates().Item("SampleFrame"))
project.Templates().Create("SampleFrame", false);

// delete and recreate sample picture or create a new one if it not yet exists
IDynPicture pic = null;
IDynPictures pictures = project.DynPictures();
for (int i = 0; i < pictures.Count; i++)
{
IDynPicture picture = pictures.Item(i);
if (picture.get_DynProperties("Title").ToString() == "SamplePicture")
{
pic = picture;
break;
}
}
```

```
if (null != pic)
{
pic.Save();
pictures.Delete(pic.Name);
}
pic = project.DynPictures().Create("SamplePicture", "SampleFrame",
tpDynPicturesTypes.tpStandard);

// create a rectangle for demo and assign a background color

IElement ele = pic.Elements().Create("Rectangle", tpElementTypes.tpRectangle);
ele.BackColor = ZenonColorHelper.Color2Api(System.Drawing.Color.Aqua);

System.Diagnostics.Debug.Print("zenon color: " + ele.BackColor);
System.Diagnostics.Debug.Print("As Html: " +
ZenonColorHelper.Api2ColorString(ele.BackColor));

System.Drawing.Color color = ZenonColorHelper.Api2Color(ele.BackColor);
System.Diagnostics.Debug.Print("As Color: r" + color.R + " g" + color.G + " b" + color.B
+ " a" + color.A);

// create a filled rectangle (color defined directly from a color palette)

// look for existing color palette in current project or create a new one
IColorPalettes palettes = project.ColorPalettes();
int nrOfPalettes = (int)palettes.get_DynProperties("Palette");
if (0 == nrOfPalettes)
{
//Create a new palette
palettes.CreateDynProperty("Palette");
//Define a color in the new palette
String htmlColor = "#00FFFF"; // System.Drawing.Color.Aqua
palettes.set_DynProperties("Palette[0].PaletteColors", htmlColor);
}

// link color palette entry to element background color
ele.BackColor = ZenonColorHelper.EncodeColorPaletteIndex(0);
System.Drawing.Color index = ZenonColorHelper.Api2Color(ele.BackColor);
System.Diagnostics.Debug.Print("Index encoded in color: r" + index.R + " g" + index.G
```

```
+ " b" + index.B + " a" + index.A);
System.Diagnostics.Debug.Print("Palette index:" +
ZenonColorHelper.GetColorPaletteIndex(ele.BackColor));



// output of the sample:
//
// zenon color: 4294967040
// As Html: #00FFFF
// As Color: r0 g255 b255 a255
//
// Index encoded in color: r0 g0 b0 a128
// Palette index: 0
}
```

6. Process Control Engine (PCE)

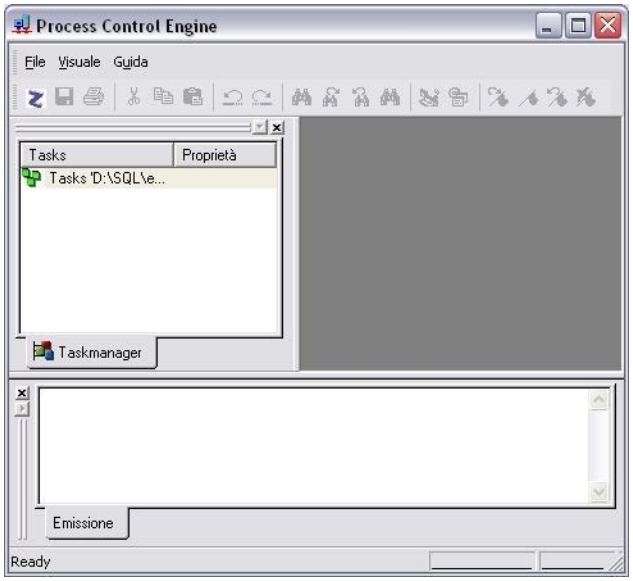
Attenzione

PCE non viene più supportato a partire dalla versione 7.20 e non viene più visualizzato nella struttura ad albero dei moduli di zenon. Quando si convertono dei progetti realizzati con versioni precedenti alla 7.20 e contenenti dei task PCE, viene visualizzato di nuovo il nodo PCE per questi progetti. È stato sospeso il lavoro di sviluppo e di documentazione per PCE.

Raccomandazione: *Usate **zenon Logic** invece di PCE.*

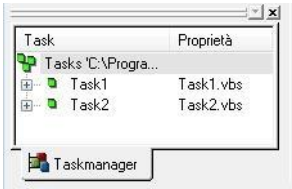
6.1 L'editor PCE

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

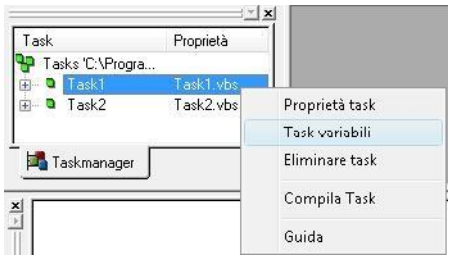


6.1.1 Il task manager

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 menù contestuale di un task.



Il menù contestuale di un task ha quattro voci:

Parametro	Descrizione
Proprietà task	Apre il dialogo "Proprietà" del task.
Variabili task...	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.

6.1.2 Area di editazione

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

6.1.3 Finestra di emissione

6.1.4 I menù dell'editor PCE

Menù "File".

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

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

Menù "Modifica".

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

Parametro	Descrizione
Annulla	Annulla l'ultima azione eseguita.
Ripeti	Ripete l'ultima azione eseguita.
Taglia	Sposta un testo negli "appunti" di Windows.
Copia	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.

Menù "Esegui".

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

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

Menù "Visualizza"

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

Parametro	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

Menù "Finestra"

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

Chiudi
Disponi
Dividi
Riordina simboli
Lista delle finestre aperte

Menù "Guida"

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

Comando	Azione
Guida	Apre la guida online.
Informazioni su...	Aprite una finestra con informazioni su zenon: ▸ Numero di serie ▸ TAG autorizzati ▸ 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.

6.1.5 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:

Parametro	Descrizione
Chiudi	Chiude l'editor PCE.
Salva tutti	Salva i task nuovi o quelli modificati.
Stampare l´immagine attiva	Stampa il task attivo al momento.
Taglia	Sposta un testo negli "appunti" di Windows.
Copia	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.

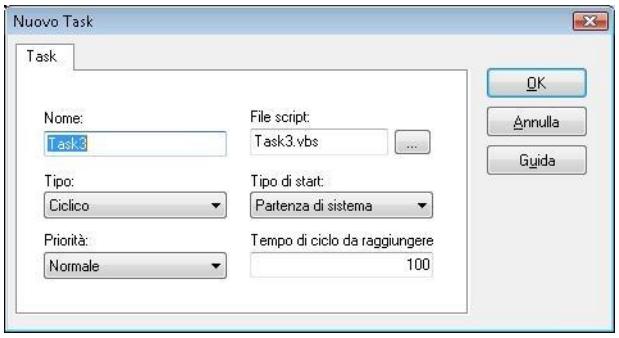
6.2 Procedura

6.2.1 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:

Parametro	Descrizione
Nome	Denominazione univoca del task.
Tipo	I task possono essere eseguiti **ciclicamente** 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...**.

6.2.2 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 contemporaneamente il corrispondente file VBS o JS.

Vengono eseguite automaticamente quattro procedure:

Parametro	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	Metodo	
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	Metodo	Avvia un task
Sleep	Metodo	Ferma un task.
StartTimer	Metodo	Il metodo StartTimer avvia un timer del task.
Stato	Proprietà	
Stop	Metodo	Arresta un task.
StopTimer	Metodo	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.

6.2.3 Funzione Show PCE

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

6.2.4 Esecuzione di task

I task possono eseguiti con lo start di sistema, oppure in modo gestito dagli eventi.

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`

Nota: Per eseguire un task più volte, lo si deve esplicitamente terminare ad esecuzione completata, in modo tale da poterlo avviarlo di nuovo.
Indicazione: **`thisProject.Tasks.Item(Taskname).Stop`**
Dopo uno stop corretto, il task potrà essere avviato nuovamente.

Sul terminale CE

Visto che Windows CE non supporta VBA, la procedura sopra descritta, riguardante l´esecuzione su un PC, 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`

6.3 VBScript - Introduzione

6.3.1 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	Significato
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.
Stringa	Contiene una stringa di lunghezza variabile composta da unmassimo di circa 2 miliardi di caratteri.
Object	Contiene un oggetto.
Errori (errors)	Contiene un numero di errore.

6.3.2 Variabili

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 superare i 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)
```

6.3.3 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.

Nota sulla programmazione

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

La stringa letterale è racchiusa tra virgolette (ad es. "testo campione"). 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 (#);

Nota sulla programmazione

```
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.

6.3.4 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.

6.3.5 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. In questo esempio, la parola chiave `Else` è 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. In questo esempio, la parola chiave `Else` è 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 out of range!
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
```

6.3.6 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:

Parametro	Descrizione
Utilizzo di cicli Do (A pagina: 147):	le istruzioni vengono ripetute fino a quando una condizione risultaTrue o rimane
Utilizzo di cicli While...Wend (A pagina: 149):	le istruzioni vengono ripetute fino a quando una condizioneè True.
Utilizzo di cicli For...Next (A pagina: 149):	le istruzioni vengono ripetute il numero di volte specificato da uncontatore.
Utilizzo di cicli For Each...Next (A pagina: 150):	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
 myNum = myNum - 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 DoMyProc50Times()
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='Hierauf klicken'><p>
</FORM>
</CENTER>
</BODY>
</HTML>
```

6.3.7 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.

Esempi:

```
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
```

6.3.8 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 (_).

Esempi:

```
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".

Per 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.

Sottotipo	Prefisso	Esempio
Boolean	bln	blnFound
Byte:	byt	bytRasterData
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance
Errori (errors)	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
Stringa	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
A livello di routine	Evento, funzione, o subroutine.	Visibile nella routine in cui è stata dichiarata.
A livello di 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 dell'effetto della routine su variabili esterne, controllie altri elementi.
`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 di feedback`	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 feedback: 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 blnGefunden ' 'Target found' flag.
intFindUser = -1
i = 0 ' Initialize loop counter
Do While i <= Ubound(strUserList) and Not blnFound
If strUserList(i) = strTargetUser Then
blnGefunden = True 'Set flag to True
intSuche Benutzer = i ' Set return value to loop count
End If
i = i + 1 ' Increment loop counter
Loop
 End Function
```

7. Object Model

In quest'area si trova la guida per l´Object Model di zenon. È disponibile per Add-Ins, per VBA, VSTA e per l'interfaccia COM esterna..

La versione più recente è disponibile online all'indirizzo: https://onlinehelp.copadata.com (https://onlinehelp.copadata.com).