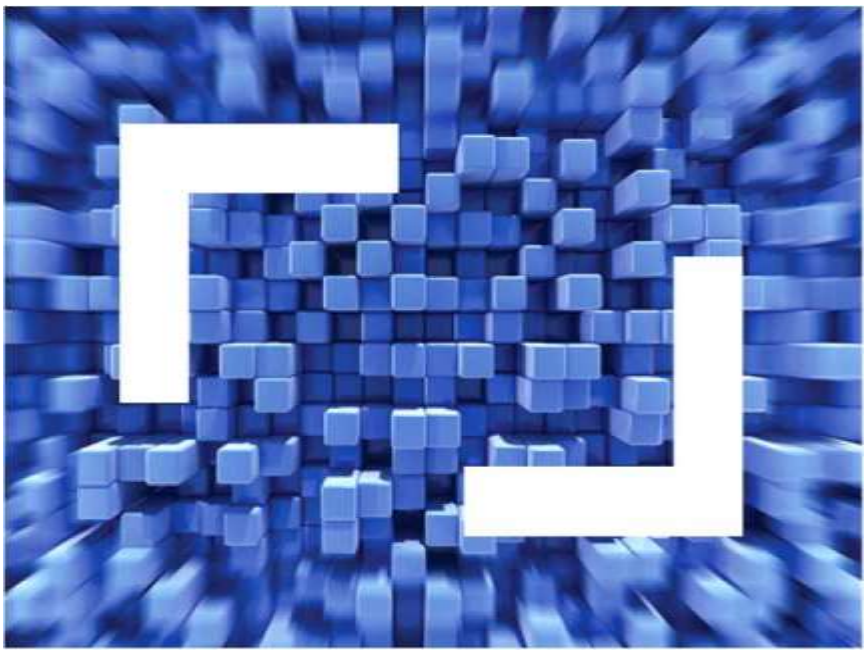


Introduzione al Visual COBOL e COBOL per .NET

MICRO
FOCUS
Leading the Evolution -

Micro Focus The Lawn
22-30 Old Bath Road Newbury, Berkshire RG14 1QN UK
http://www.microfocus.com

Contenuti

Visual COBOL e COBOL per .NET Introduzione al Linguaggio 4
Introduzione 4
Panoramica su Visual Studio 4
Codice Gestito e Codice Nativo 4
Caricare Visual COBOL 5
Progetti e Soluzioni 6
Una prima Applicazione Console 6
Creare una Applicazione Console 6
Costruire ed Eseguire una Applicazione da Visual Studio 7
Costruire ed Eseguire una Applicazione dal Prompt dei Comandi 7
Una Applicazione Grafica 8
Creare una Applicazione Grafica 8
Aggiungere una Label 8
Aggiungere un Pulsante (Button Control) 9
Aggiungere un Gestore di evento (Click Event Handler) 9
Aggiungere una Text Box 10
Leghiamo tutto insieme 10
Panoramica su Visual COBOL Debugging 11
Generare il codice per il Debug 11
Esecuzione con il Debug 12
I Breakpoint 12
Impostare un Breakpoint 12
Disabilitare un Breakpoint 13
Eliminare un Breakpoint 13
Navigare con il Debug 13
Migrazione verso il codice gestito (Managed Code) 13
Applicazione CblDemo1 14
Supporto al Codice Nativo 16
Margini COBOL 16
Identificatori non Usati 17
Tipi di Dati e Namespace 17
Tipi di Dati COBOL 18
Tipi di dati .NET 18
Spazio dei Nomi (Namespace) 18
Tipo dato DateTime 18
Tipo Dato String 20
Catturare l'Errore 22
Gestione delle Exception con i dati di tipo DateTime 23
Working With Classes 23
Creare una Classe 23
Riempire la Classe 24
Aggiungere Metodi alla Classe 24
Usare la Classe Persona 25
Ulteriori Miglioramenti 26
Appendice 28
Cambiare l'oggetto di inizio 28
Riferimenti ai Tipi di Dati 29
Lavorare con IntelliSense 30

Visual COBOL e COBOL per .NET Introduzione al Linguaggio

Introduzione

Questo documento fornisce una breve introduzione al Visual COBOL per Visual Studio e al linguaggio COBOL per .NET. Esso è un'introduzione ai concetti di base all'ambiente di sviluppo e mostra alcune applicazioni di base, i benefici, le caratteristiche e i miglioramenti del COBOL per .NET.

Il documento è suddiviso in due parti. Nella prima parte potrai creare una semplice applicazione per comprendere l'ambiente di sviluppo del Visual COBOL per Visual Studio. Al termine esamineremo gli strumenti di test disponibili in Visual COBOL. La seconda parte inizia con una applicazione commerciale in cui saranno introdotti progressivamente i concetti del Visual COBOL al fine di comprendere come un programma scritto con edizioni precedenti del COBOL può migrare verso la nuova versione ed i relativi vantaggi.

Panoramica su Visual Studio

Visual Studio è un ambiente di sviluppo integrato (IDE - Integrated Development Environment) usato per lo sviluppo di molte applicazioni per desktop che utilizza l'interfaccia grafica (IDE - Integrated Development Environment). Può anche essere usato per lo sviluppo di siti web.

L'editor grafico contiene numerosi strumenti e può gestire plug-in per funzioni aggiuntive sia della Microsoft che da terze parti (altri fornitori). Accetta diversi linguaggi di programmazione, alcuni già integrati, altri da installare a parte, come il Visual COBOL (plug-in).

Il COBOL per .NET

Il Visual COBOL per Visual Studio fornisce il COBOL per .NET, che è il COBOL con estensioni che gestiscono il framework per .NET. Questa estensione consente la Programmazione Orientata agli Oggetti The extensions allow (OOP - Object Oriented Program), il controllo della sintassi e l'accesso alle librerie .NET.

Codice Gestito e Codice Nativo

Il codice native è quello creato e compilato in linguaggio macchina per uno specifico microprocessore e sistema operativo. Non è eseguibile con differenti microprocessori e sistemi operativi.

Creazione Codice Macchina

Il codice gestito è il codice eseguito in ambiente controllato. Computer con un microprocessore diverso da quello che riconosce il codice nativo non possono eseguire il codice nativo senza un sistema che gestisca l'esecuzione del codice. Sui sistemi Windows l'ambiente di controllo è .NET framework runtime (CLR - Common Language Runtime).
Nei sistemi Windows il codice sorgente è tradotto nel linguaggio intermedio (MSIL - Microsoft Intermediate Language) dal CLR. Tutti i linguaggi che rispettano le specifiche del CLR sono tradotti correttamente.

Prima dell'esecuzione il CLR compila l'MSIL nel linguaggio macchina del computer ospite. Poiché ciò avviene in ambiente controllato, il codice così prodotto può essere usato da ogni computer ospite. (NdT: Al contrario della compilazione 'normale', in cui tutto il codice sorgente è tradotto in linguaggio macchina del computer destinatario del codice eseguibile, in ambiente controllato il codice sorgente è prima tradotto in un linguaggio intermedio (MSIL) e quando deve essere eseguito sulla macchina ospite è infine tradotto nel linguaggio macchina del computer che eseguirà il codice nel suo specifico linguaggio macchina)

Compilazione Del Codice Sorgente

Creazione del Codice Macchina

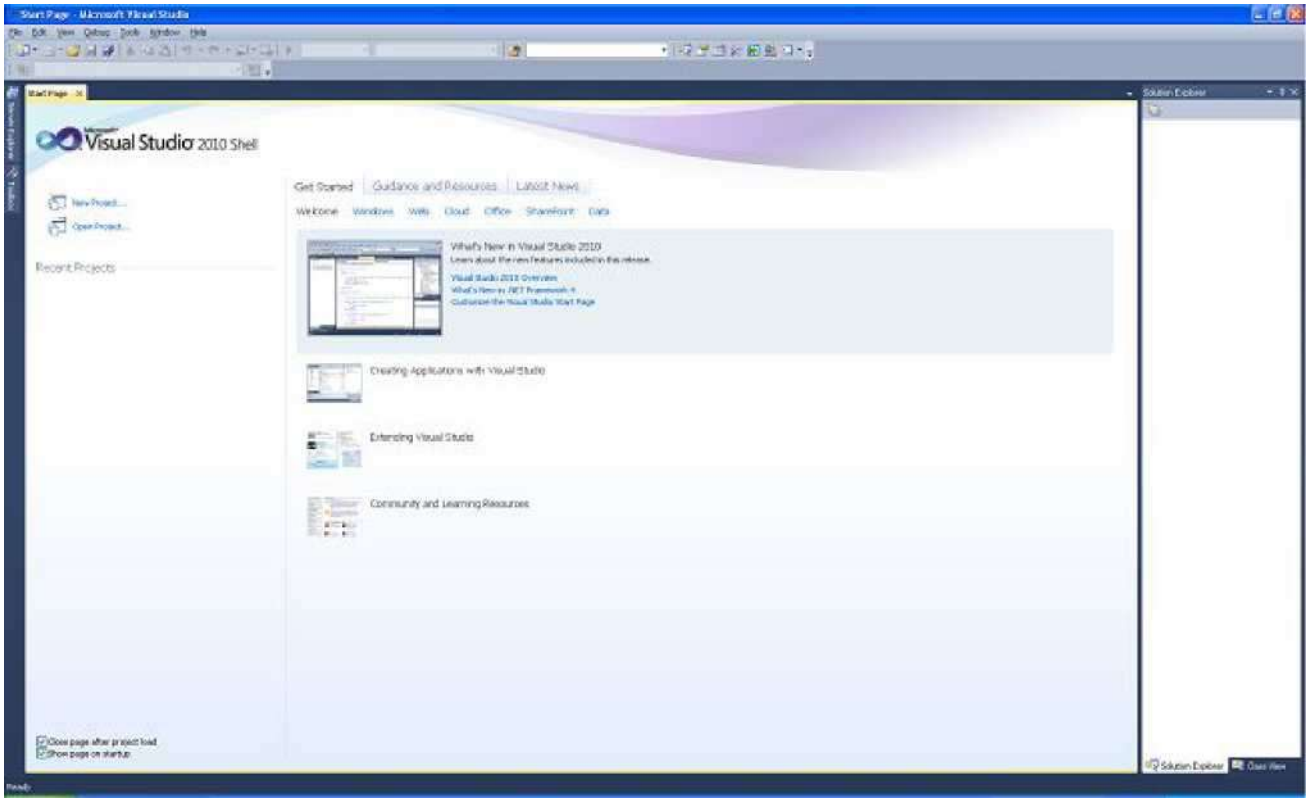
Caricare Visual COBOL

Far partire Visual COBOL per Visual Studio per caricare le librerie standard e gli strumenti di sviluppo per applicazioni riusabili con il COBOL.

• Click Start > Tutti i Programmi > Micro Focus Visual COBOL 2010 > Visual COBOL for Visual Studio.

Se viene chiesta l'impostazione dell'ambiente, scegliere 'General development environment'. Il 'General development environment' imposta Visual Studio per lavorare con progetti COBOL.

Visual COBOL parte in Visual Studio, e la pagina iniziale (Start) di Visual Studio mostra le informazioni aggiornate di Visual Studio come segue.



Usare il menu File per aprire ‘solutions and projects’. Solution Explorer appare a destra del pannello. Il ‘Solution Explorer’ mostra la struttura del progetto o soluzione aperti.

In ‘Solution Explorer’, fare doppio click sul nome del file per mostrare i suoi contenuti in un ‘Code Editor’ separato.

La finestra di Output mostra nella parte sottostante la linguetta del Code Editor nel quale appariranno i messaggi dell’IDE e del compilatore. La linguetta ‘Error List’ nella finestra di Output mostra tutti i messaggi di errore, i warning e gli altri messaggi risultanti dallo sviluppo del codice e dalla compilazione della soluzione o del progetto.

Progetti e Soluzioni

Una Soluzione è un contenitore di uno o più progetti che lavorano insieme per creare un’applicazione. Un file Soluzione ha estensione .sln ed è un file di testo.

Un Progetto è una unità che costituisce ua libreria o un eseguibile. Un file di progetto Managed COBOL ha estensione .cblproj ed è un file di testo.

Sia i Progetti che le Soluzioni non dovrebbero essere modificati al di fuori di Visual Studio.

Una prima Applicazione Console

Questa sezione mostra come creare un’applicazione console gestita COBOL a partire dai template forniti con il Sistema e modificandola per creare un’applicazione tradizione di tipo ‘Hello World’. Vedremo inoltre come costruire ed eseguire un’applicazione Visual COBOL e come eseguirla dalla linea di comando (PROMPT).

Creare una Applicazione Console

Seguire i passi seguenti per creare una nuova Soluzione e un nuovo Progetto. La nuova Soluzione è mostrata in ‘Solution Explorer’. Il programma COBOL creato dal template per il progetto è mostrato nella

linguetta ‘Code Editor’.

1. Click File > New > Project.
   Appare la dialog box per ‘New Project’.
2. Click COBOL > Managed nel pannello ‘Installed Templates’.
   I template disponibili COBOL compaiono nella lista dei template.
3. Selezionare ‘Console Application’ dalla lista.
4. Scrivere ‘`ConsoleHelloWorld`’ come nome del progetto nel campo ‘Name’, e fare click su OK.
   Visual Studio crea una nuova soluzione e un progetto nella Soluzione chiamata ‘`ConsoleHelloWorld`’, e un nuovo template di programma chiamato ‘`Programl.cbl`’. `Programl.cbl` è visualizzato nella linguetta ‘Code Editor’. Il codice corrente per ‘`Programl.cbl`’ è il codice del template, come mostrato di seguito.

```
program-id. Programl as "ConsoleHelloWorld.Programl".

data division. working-storage section.

procedure division.

       goback.

end program Programl.
```

5. Aggiungere una istruzione a questo programma per visualizzare il testo "Hello World" quando è eseguito. Per ottenere questo, posizionare il cursore dopo la linea ‘`procedure division`’ e premere il tasto ‘`Tab`’ per muovere il cursore all’inizio dell’Area B (colonna 12). Aggiungere l’istruzione ‘`display`’ nel programma tra ‘`procedure division`’ e l’istruzione ‘`goback`’. Il programma sarà come segue.

```
program-id. Programl as "ConsoleHelloWorld.Programl".

data division. working-storage section.

procedure division.
       display "Hello
       World" goback.

end program Programl.
```

6. Click File > Save *Programl.cbl* per salvare il codice così aggiornato.

Costruire ed Eseguire una Applicazione da Visual Studio

Costruire un’applicazione in Visual Studio vul dire convertire il codice in MSIL. Un’applicazione si può eseguire in Visual Studio oppure dal prompt dei comandi. Quando un’applicazione è eseguita, il codice MSIL è compilato dal CLR nel codice macchina del computer ospite.

1. Click Build > Build *ConsoleHelloWorld.*
   ConsoleHelloWorld è il nome del progetto corrente.La finelstra di output mostra I messaggi generate durante la compilazione.
2. Una volta completata la compilazione, click Debug > Start without debugging.
   Il prompt della console mostra il testo ‘Hello World’. Quindi ti viene chiesto di premere un tasto per continuare. La pressione del tasto chiude quindi la console del prompt.

Costruire ed Eseguire una Applicazione dal Prompt dei Comandi

Per eseguire un’applicazione dal prompt dei comandi invece che da Visual COBOL, aprire un prompt di comandi COBOL e scrivere il commando per eseguire l’applicazione.

1. Click Start menu > All Programs > Micro Focus Visual COBOL 2010 > Visual COBOL Tools > Visual COBOL Command Prompt (32-bit).
   Questo apre un prompt di commando nella cartella 'Documenti' per default. Da questa cartella naviga fino alla cartella che contiene l'applicazione. La cartella è '*\Visual Studio 2010\Projects \ConsoleHelloWorld\ConsoleHelloWorld\bin\debug* by default. Varificare che l'applicazione sia effettivamente in questa cartella.
2. Per navigare fino alla cartella che contiene l'applicazione, scrivere come segue: `'cd \[default document directory] \Visual Studio 2010\Projects\ConsoleHelloWorld\ConsoleHelloWorld\bin\debug' nel prompt dei comandi e premere 'Enter'`.
3. Scrivere '`ConsoleHelloWorld' e premere '` Enter'.
   Parte l'applicazione e mostra il testo '`Hello World'` e quindi termina. Il testo '`Press any key to continue...'` non appare. Questo testo appare soltanto se l'applicazione è eseguita in Visual COBOL per permettere di capire che l'applicazione è terminata.

Una Applicazione Grafica

Questa sezione mostra come creare una applicazione Gestita COBOL a partire da un template ed usarlo per creare una applicazione tradizione 'Hello World'. Mostra inoltre come estenderla inserendo un input da utente e un pulsante.

Creare una Applicazione Grafica

Seguire questi passi per creare una nuova Soluzione per l'applicazione grafica.

1. Click File > New > Project.
   Compare la dialog box del Nuovo Progetto.
2. Click COBOL > Managed in the Installed Templates pane.
   E' mostrata la lista dei template COBOL disponibili.
3. Seleziona Windows Forms Application dalla lista.
4. Scivere '`WindowsHelloWorld'` come nome del progetto nel campo 'Name' e click OK.
   Visual Studio crea una nuova Soluzione e un nuovo Progetto nella Soluzione chiamata '*WindowsHelloWorld',* e un nuovo template di programma chiamato '`Main.cbl'`. Crea anche un nuovo form associate al programma chiamato '`Form1.cbl'` e '`Form1.Designer.cbl'`. La struttura del progetto è mostrata in Solution Explorer, e il `Form1.cbl` nell'editor del codice.

Aggiungere una Label

Usare le label per posizionare del testo o titoli nel form. Il testo contenuto in una label non può essere modificato dall'utente. Seguire i seguenti passi per creare e posizionare una label in un form.

1. Click View > Toolbox per mostrare il Toolbox.
2. Drag and drop il controllo Label control sul form.
   Questo crea una Label chiamata 'label1' contenente il testo '*label1'.*
3. Nel Solution Explorer, evidenziare la proprietà Text nella sezione Appearance del pannello Properties.
   Il pannello Properties è in basso a destra dell'IDE di Visual Sutdio sotto il pannello Solution Explorer.
4. Scrivere '`Hello World'` per impostare il valore e premere tasto Enter.
   Il valore appena impostato appare nella label.

Usare le istruzioni contenute in ***Costruire ed Eseguire una Applicazione da Visual Studio*** per costruire ed eseguire l'applicazione. Ciò apre un form chiamato '*form1'* che mostra una label con scritto '`Hello World'`. Click sulla x nell'angolo in alto a destra per chiudere il form fermare l'applicazione.

Aggiungere un Pulsante (Button Control)

Un pulsante è un controllo sul form che l'utente può cliccare per eseguire un predeterminato compito. Un gestore degli eventi definisce il compito che eseguirà il pulsante. Usare I seguenti passi per posizionare un pulsante sul form `Formi.cbl`.

1. Click View > Toolbox per mostrare il Toolbox.
2. Drag and drop un Button control sul form.

   Tip: Visual Studio mostra una linea vertical ed una orizzantale sul form per favorire l'allineamento del controllo.

3. Nel Solution Explorer, evidenziare il valore della proprietà Text nella sezione Appearance del pannello Properties.
4. Scrivere '`Say Hello'` per cambiare il valore della proprietà e premere tasto Enter.
   Si aggiorna il valore della proprietà del Pulsante.

Usare le istruzioni del paragrafo ***Costruire ed Eseguire una Applicazione da Visual Studio*** per costruire ed eseguire l'applicazione. Il pulsante è mostrato nel form. Comunque cliccando sul pulsante non accade nulla poichè non è stato ancora definite alcun evento per esso. Il prossimo esercizio definirà il gestore dell'evento per il pulsante.

Aggiungere un Gestore di evento (Click Event Handler)

Un click event handler è il codice che viene eseguito quando un pulsante è cliccato. Il codice può essere costituito da una o più azioni (istruzioni), compresa la navigazione in altre pagine (o form). I seguenti passi aggiungono un click event handler al pulsante.

1. Fare doppio sul pulsante.
   Viene visualizzato il codice per `Formi.cbl`. E' generato il codice per un metodo chiamato *button1_click* che è mostrato in un'altra linguetta dell'Editor. AL metodo è dato lo stesso nome del pulsante con una estensione '_click.
   Questo metodo contiene che viene eseguito quando si clicca sull pulsante, come mostrato di seguito.

```
working-storage section.

method-id NEW. procedure division.
    invoke self::InitializeComponent goback. end method.

method-id button1_Click final private.
procedure division using by value sender as object e as type
System.EventArgs.
end method.

end class.
```

2. Aggiungere una istruzione per inserire il testo 'Hello World' nella proprietà Text della label chiamata '*label1'* creata precedentemente. Il valore precedente della label è sostituito Posizionare il cursore dopo la linea '`procedure division'` e premere tasto Enter.
   E' generate una nuova linea ed il cursore posizionato all'inizio dell'Area A (Colonna 8).
3. Premere il tasto Tab per posizionarlo all'inizioe dell'Area B e scrivere il gestore dell'evento come segue.
   `set self::label1::Text to "Hello World"`
   Questo codice impost ail testo 'Hello World' nella proprietà Text della label *labell*. Il valore precedente *labell* è sostituito con il nuovo valore.

   Tip: Visual COBOL usa IntelliSense per suggerire il completamento del testo mentre scrivi. Vedi *Working With* " *IntelliSense* per maggiori informazioni.

   Il codice dovrebbe essere il seguente.

```
working-storage section.

method-id NEW. procedure division.
    invoke self::InitializeComponent
goback. end method.

method-id button1_Click final private.
procedure division using by value sender as object e as type System.EventArgs.
    set self::label1::Text to "Hello World"
end method.

end class.
```

4. Click sulla linguetta Forml.cbl [Design] il alto nell'editor del codice per mostrare il form nella linguetta dell'Editor.
5. Click sulla Label.
   Vengono mostrate le proprietà nel pannello delle Proprietà.
6. Pulire il valore della proprietà Text e premere Enter.
   Viene aggiornato il valore della proprietà.

Usa le istruzioni contenute il ***Costruire ed Eseguire una Applicazione da Visual Studio*** per costruire ed eseguire l'applicazione.. Quando parte, si apre il form ed è mostrato un pulsante con il testo '`Say Hello`'. Click sul pulsante per far apparire '`Hello World`' sul form. Click la x nell'angolo in alto a destra per chiudere e fermare l'applicazione.

Aggiungere una Text Box

Una text box è un controllo che permette di inserire del testo. Seguire questi passi per per inserire una text box sul form.

1. Click sulla linguetta *Form1.cbl [Design]* per visualizzare la finestra del designer del form.
2. Click View > Toolbox per visualizzare il Toolbox.
3. Drag and drop un controllo Text box sul form.
   Sul form è aggiunto un controllo text box.

Usa le istruzioni contenute in ***Costruire ed Eseguire una Applicazione da Visual Studio*** per costruire ed eseguire l'applicazione che visualizza una text box e un pulsante. Per ora puoi scrivere qualunque cosa nella text box poichè non è presa in considerazione dal codice.

Leghiamo tutto insieme

La nostra applicazione ora è composta da Label, Pulsante e Text box e un gestore di evento click. Resta ora di far apparire una label per il campo di input ed un gestore di eventi al pulsante per far qualcosa con il campo in input. Procedere come segue.

1. Aggiungere una Label al form.
   Posizionare la label vicino la text box per dare all'utenete l'indicazione di cosa scrivere nella text box.
2. Nel Solution Explorer, è evidenziato il valore della proprietà Text nella sezione Appearance del pannello delle Proprietà.
3. Scivere '`First Name`' per cambiare il valore della proprietà. Premere tasto Enter.

   TViene aggiornato il valore della proprietà nella label.
4. Doppio click sul pulsante.
   Viene visualizzato il codice per `Formi.cbl`. Questo codice contiene il gestore dell'evento clickper il pulsante come nell'esercizio precedente, ma con un metodo addizionale per la nuova label, come mostrato di seguito.

```
working-storage section.

method-id NEW. procedure division.
```

```
    invoke self::InitializeComponent
goback. end method.

method-id button1_Click final private.
procedure division using by value sender as object e as type System.EventArgs.
    set self::label1::Text to "Hello " & self::textBox1::Text
end method.

method-id label2_Click final private.
procedure division using by value sender as object e as type
System.EventArgs.
end method.

end class.
```

5. Sostituire il codice inserito precedentemente in modo chhe appaia come segue:

   ```
   set self::label1::Text to "Hello " & self::textBox1::Text
   ```

   Questa istruzione concatena la costante testo '*Hello'* e il contenuto della text box *textBoxI*, e mette il risultato dentro la label *labell* quando si clicca sul pulsante.

Costruire ed eseguire l'applicazione per mostrare una text box descritta con la label '`First Name'` e un pulsante. Scrivere '`Fred'` nella text box e cliccare il pulsante. Viene visualizzare il testo '`Hello Fred'`.

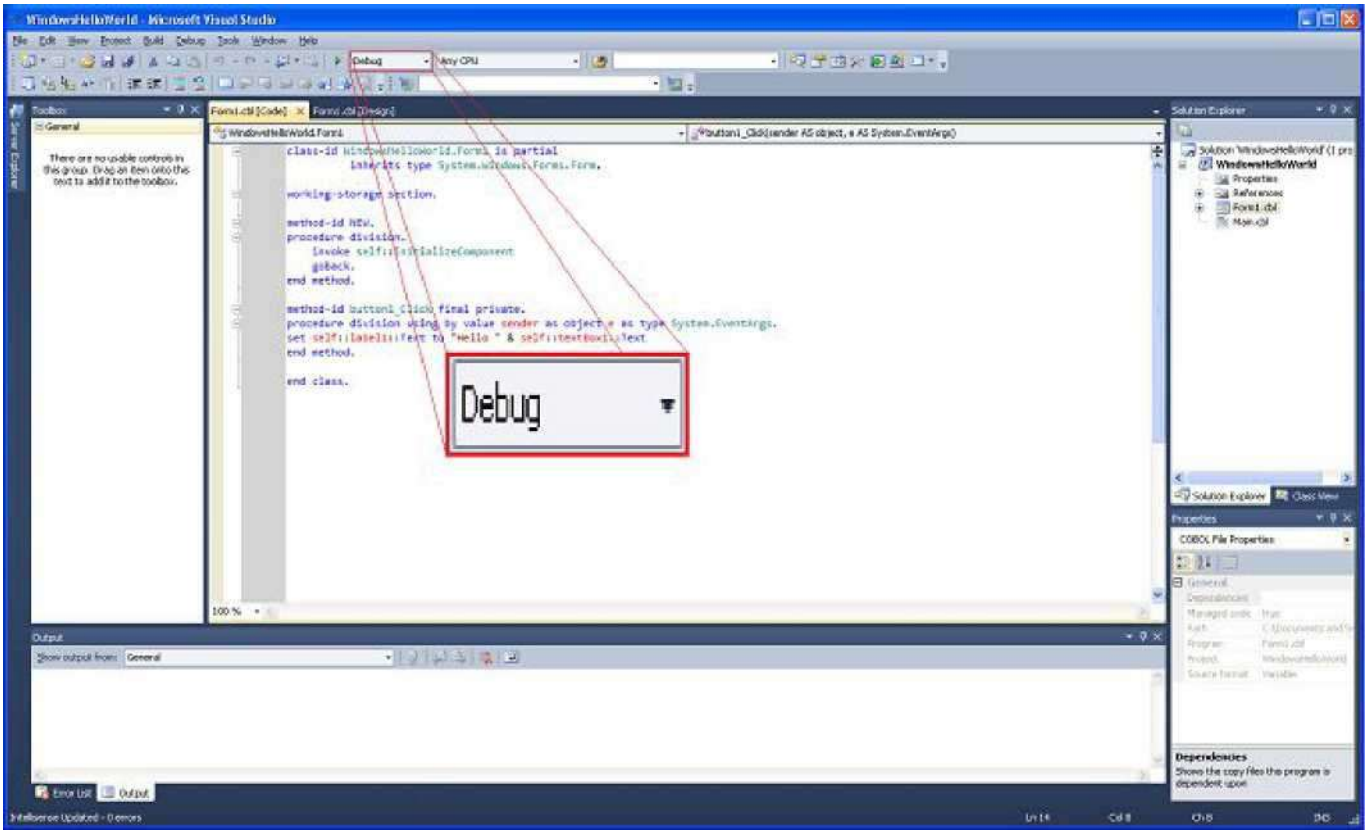
Panoramica su Visual COBOL Debugging

Scrivere una applicazione come 'Hello World' è molto semplice. La possibilità di generare errori scrivendo una parola errata è bassa poiché il Visual COBOL effettua un controllo dinamico durante la digitazione tramite IntelliSense.

Per applicazioni più complesse la possibilità di generare errori aumenta di conseguenza. Visual COBOL fornisce gli strumenti per individuare e rimuovere questi errori. Questa sezione mostra come usare le caratteristiche di debug.

Generare il codice per il Debug

Per fare il debug di una applicazione, assicurarsi che sia selezionata la configurazione del debug. La list box della Solution Configurations sulla toolbar (barra strumenti) del Visual Studio mostra il tipo di configurazione attuale. Per cambiare la configurazione, selezionare l'opzione appropriata dalla list box.



Esecuzione con il Debug

Il Visual COBOL fornisce una grande varietà di strumenti per il debug che permettono di tene traccia degli errori dell'applicazione. Questi strumenti sono disponibili durante l'esecuzione del programma quando l'esecuzione è configurata per il debug. Per configurare il debug seguire i passi seguenti.

1. Click Debug > Start Debugging.
   Il Visual COBOL ricostruisce automaticamente l'applicazione se vi è stato qualcghe cambiamento nel codice dall'ultima compilazione, quindi viene eseguita l'applicazione.
2. (opzionale) Impostare i breakpoint per fermare l'esecuzione in specifici punti del codice.

I Breakpoint

Un breakpoint è un segnaposto inserito nel codice sorgente. L'esecuzione del programma si ferma quando si deve eseguire l'istruzione della linea alla quale è stato inserito il segnaposto. Si possono interrogare le variabili, impostare nuovi segnaposto.

Tip: Mettere in pausa l'applicazione durante il debug è anche conosciuto come 'break mode'.

Impostare un Breakpoint

Per impostare un breakpoint seguire I seguenti passi.

1. Click sulla linea dove vuoi fare pausa durante l'esecuzione del programma.
2. Click Debug > Toggle Breakpoint.
   E' inserito un breakpoint. Un punto rosso appare nel margine sinistro della linea selezionata.

'*=> Nota: Se si imposta un breakpoint su una linea dove non vi è codice, il programma si fermerà alla linea successiva.

Disabilitare un Breakpoint

Durante il debug si può disabilitare un breakpoint senza eliminarlo e tenerlo per un uso future. Questo non può esser fatto tramite il menu 'option' ma si può usare il menu contestuale come segue.

1. Click tasto destroy sulla linea contenente il breakpoint per visualizzare il menu contestuale.
2. Selezionare Breakpoint > Disable Breakpoint.
   Viene disabilitato il breakpoint come indicato dal cerchio rosso nel margine sinistro. Per ri-abilitarlo selezionare Breakpoint > Enable Breakpoint dal menu contestuale.

Eliminare un Breakpoint

Per rimuovere il breakpoint, fare come segue.

1. Selezionare il breakpoint cliccando sulla linea corrispondente.
2. Click Debug > Toggle Breakpoint.
   Il breakpoint selezionato è rimosso. Ripetendo questi passi alternativamente su imposta e si rimuove il breakpoint.

Navigare con il Debug

Poter interrogare I dati ed il loro flusso durante l'esecuzione di un programma è una caratteristica utile e potente dell'ambiente Visual COBOL. Viene mostrata la linea correntemente eseguita nel Code Editor. Durante l'esecuzione è evidenziata la linea di codice in eseguita. Nella lista seguente vi sono le opzioni disponibili durante il debug.

- Stepping Over (Debug > Step Over)

  Esegue la linea corrente e procede alla linea di codice sguente nel blocco di codice. Ogni metodo è eseguito completamente. Quando si raggiunge la fine del blocco, l'esecuzione torna al blocco chiamante.
- Stepping Into (Debug > Step Into)

  Esegue un comando alla volta. Per una operazione numerica, il dedugger si comporta come lo Step Over. Per operazioni più complesse, il debugger pass ail controllo al metodo chiamato ed esegue la sua prima linea, esegue un passo alla volta nel resto del metodo prima di tornare la metodo chiamante. Il Code Editor mostra la linea in esecuzione.
- Stepping Out (Debug > Step Out)

  Esegue fino al termine del metodo corrente e si ferma alla prossima linea del metodo chiamante.
- Restarting the debugger (Debug > Restart)

  Riprende il debug dalla prima linea del programma.
- Stopping the debugger (Debug > Stop Debugging)

  Termina il debugger.
- Setting the next statement.

  Permette di impostare il prossimo stament da eseguire. Per far questo, click destroy sulla line ache deve essere eseguita e selezionare Set Next Statement.

Migrazione verso il codice gestito (Managed Code)

Questa parte del document illustra I maggiori benefici nell'uso del codice gestito del COBOL in ambiente Windows. Vedremo come trarre vantaggio dall'uso degli strumenti e tecniche per velocizzare la produzione di programmi efficienti ed indipendenti dalla piattaforma di destinazione.

Scrivendo una semplice applicazione console, vedremo I benefici dell'uso del COBOL come parte di una soluzione gestita. Inseriremo codice Object Oriented (OO) per usufruire delle estensioni disponibili.

Realizzeremo una soluzione semplice chiamata *CblDemoSolution*, che conterrà il progetto *CblDemoProject.*

CblDemoProject contiene programmi che riguardano le sezioni seguenti che iniziano con *CblDemol.* Il progetto include anche una classe chiamata *Person,* che sarà usata nella sezione finale. Per eseguire il programma, assicurarsi che il progetto di partenza sia impostato correttamente.Per cambiare le impostazione, vedere *Cambiare l'oggetto di inizio* in appendice.

Applicazione CblDemol

CblDemo1 è una applicazione console che chiede in input nome, data di nascita e data odierna e fornisce in output Il numero di giorni che mancano al prossimo compleanno.

L'applicazione seguente è stata lasciata scarna e priva della parte relativa al formattazione dei dati in modo da permettere di completarla secondo i propri gusti..

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
ENVIRONMENT
DATA DIVISION.
WORKING-STORAGE SECTION.
01 today.
     03 today-month  PIC 99.
     03 today-day    PIC 99.
01 dob.
     03 dob-month    PIC 99.
     03 dob-day      PIC 99.
01 test-date.
     03 test-month   PIC 99.
     03 test-day     PIC 99.
01 max-month         PIC 99.
01 max-days          PIC 99.
01 total-days        PIC 999.
01 full-name         PIC X(30 ).
01 display-output    PIC X(60 ).
 01 greeting-output  PIC X(60 ).
01 show-non-usage    PIC X(12 ).
01 txt1              PIC X(10 ) value  Hello Mr ".
01 diff-output.
     03 txt2         PIC X(11 ) value   you have ".
     03 output-days  PIC ZZ9.
     03 txt3         PIC X(40 ) value   days until next birthday".
                               "
 PROCEDURE DIVISION
     DISPLAY "Enter your  full name"
     ACCEPT full-name
     DISPLAY "Enteryour  date of birth in mmdd format"
     ACCEPT test-date
```

```
    PERFORM validate-test-date MOVE test-date TO' dob
    DISPLAY "Enter the date today in mmdd format"
    ACCEPT test-date
    PERFORM validate-test-date
    MOVE test-date TO today
    PERFORM add-day-difference
    PERFORM add-mth-difference UNTIL (max-month = DOB-month)
    MOVE total-days TO output-days
    STRING txt1, full-name DELIMITED BY SIZE INTO greeting-output
    DISPLAY greeting-output
    STRING full-name, diff-output DELIMITED BY SIZE INTO display-output
    DISPLAY display-output STOP RUN


validate-test-date SECTION.
    IF (test-day = 0)
        STOP "Invalid day field"
        STOP RUN
    END-IF.
    IF ((test-month = 0) OR (test-month > 12))
        STOP "Invalid month field"
        STOP RUN
    END-IF.
    MOVE test-month TO' max-month.
    PERFORM set-max-days.
    IF (test-day > max-days)
       STOP "Invalid day field"
       STOP RUN END-IF


add-day-difference SECTION.
    MOVE today-month TO max-month PERFORM set-max-days IF (DOB-day =
    today-day)
        MOVE ZERO' TO' total-days
    ELSE
        IF (DOB-day > today-day)
            COMPUTE total-days = DOB-day - today-day
        ELSE
            COMPUTE total-days = (max-days - today-day) + DOB-day
    PERFORM increment-test-month END-IF END-IF


add-mth-difference SECTION.
    PERFORM set-max-days
    ADD max-days TO' total-days
    PERFORM increment-test-month


increment-test-month SECTION.
    ADD 1 TO max-month
    IF (max-month > 12)
        MOVE 1 TO max-month
    END-IF


unused SECTION. EXIT SECTION.
```

```
 set-max-daysSECTION
     EVALUATEmax-month
          WHEN  1MOVE 31 TO max-days
          WHEN  2MOVE 28 TO max-days
          WHEN  3MOVE 31 TO max-days
          WHEN  4MOVE 30 TO max-days
          WHEN  5MOVE 31 TO max-days
          WHEN  6MOVE 30 TO max-days
          WHEN  7MOVE 31 TO max-days
          WHEN  8MOVE 31 TO max-days
          WHEN  9MOVE 30 TO max-days
          WHEN10 MOVE 31 TO max-days
          WHENll MOVE 30 TO max-days
          WHEN12 MOVE 31 TO max-days
END-EVALUATE


    END PROGRAM CblDemol.
```

Supporto al Codice Nativo

Il codice nativo COBOL esistente puoò essere compilato ed eseguito in Visual COBOL senza alcun cambiamento. Ciò vuol dire che le applicazioni scritte in COBOL non hanno bisogno di alcuna conversione nel nuovo ambiente.

In Visual COBOL alcune regole sintattiche non sono più necessarie. Esse sono ancora riconosciute dal Visual COBOL, ma non sono compilate. In *CblDemol* alcune linee,riportate di seguito, sono state inserite per scope illustrativi, ma potrebbero essere rimosse.

- IDENTIFICATION DIVISION
- ENVIRONMENT DIVISION
- DATA DIVISION
- WORKING-STORAGE SECTION

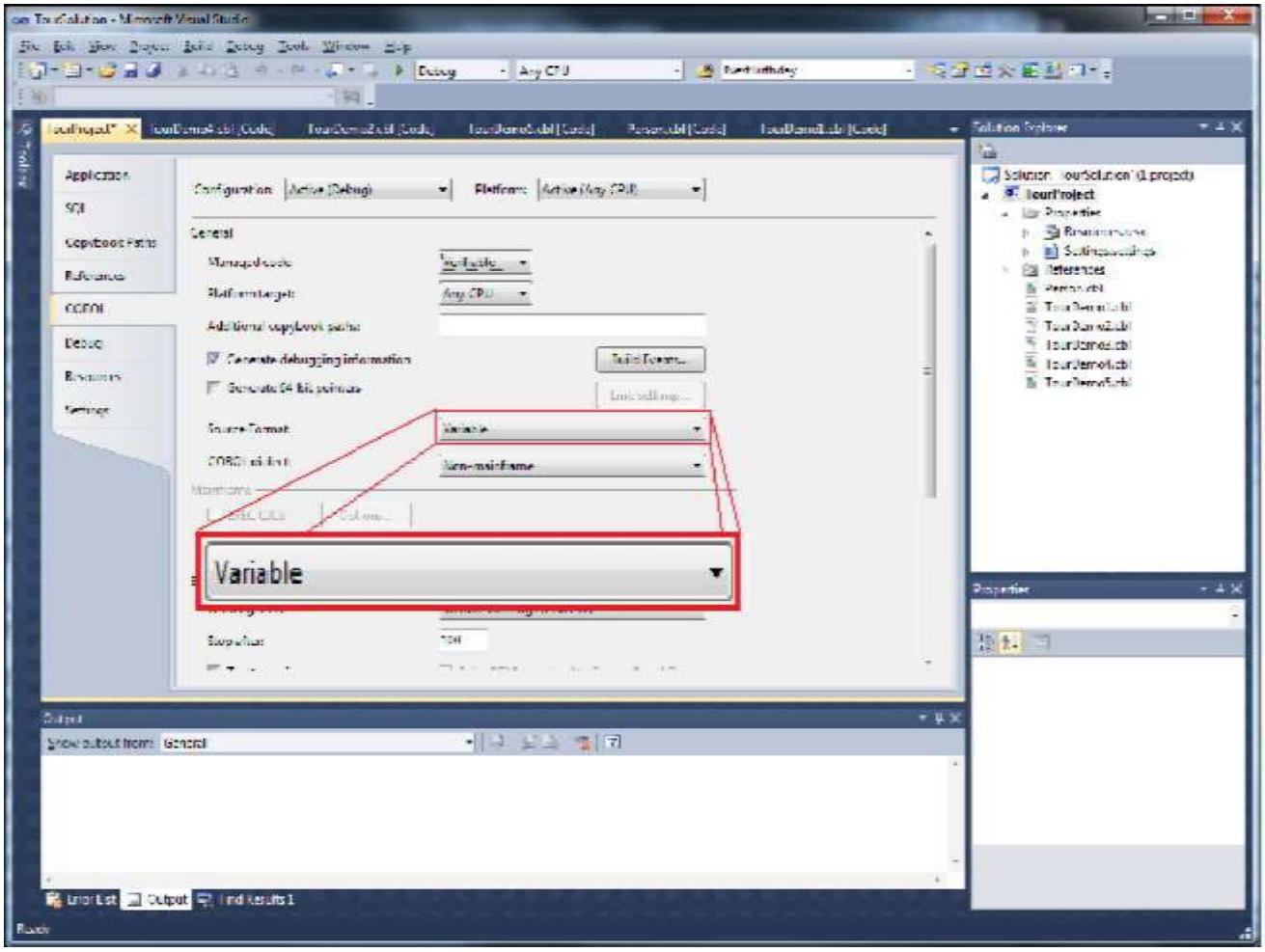
Margini COBOL

Il Visual COBOL mostra l'insieme dei margini COBOL. Quando il formato è impostato come `Variable` è visualizzato un margine grigio. Un ulteriore margine sulla destra è visualizzato quando quando il formato è impostato come `Fixed`.

Cambiare i margini in COBOL

Seguire I seguenti passi per cambiare i margini in Visual COBOL:

1. Click destro sul progetto in Solution Explorer e selezionare Properties.
   Le proprietà del progetto sono visualizzate nella pagina delle Proprietà.
2. Nella pagina Proprietà selezionare la linguetta COBOL.
   Appare la linguetta delle proprietà COBOL.
3. Usare la list box del Source Format per selezionare il tipo di margini.
   Questo imposta I margini mostrati per il codice COBOL in Visual COBOL.



Usare l'impostazione Source Format invece della direttiva del compilatore `SOURCEFORMAT` per impostare i margini.

Identificatori non Usati

Visual COBOL fornisce un efficiente aiuto per trovare variabili e sezioni non usate.

Per default, I nomi delle variabili non usate appaiono in grigio. Il colore del testo cambierà quando saranno usate nell'applicazione.

Il colore del testo può aiutare a comprendere quali variabili possono essere rimosse. *CblDemol* include un esempio di variabile non usata di nome *show-non-usage.*

Le sezioni non usate sono identificate ponendo il cursore sul nome della sezione. E' mostrato un ToolTip che indica il numero di volte che è referenziato. Se non appare alcun numero (*No References)* la sezione può essere rimossa. In *CblDemol* vi è la sezione *unused* che non ha referenze.

Tipi di Dati e Namespace

Questa sezione descrive il codice per I tipi di dati e i namespaces e come sono usati. Inoltre:

- Usa *CblDemo1* per mostrare il tipo `DateTime` e come ottimizzarne l'uso.
- Usa *CblDemo3* per mostrare il tipo `String` e come ottimizzarne l'uso.

Nella soluzione di esempio, *CblDemo2* contiene il codice completo basato sul tipo `DateTime` degli esercizi di questa sezione. *CblDemo3* contiene il codice completo basato sul tipo `String` degli esercizi di questa sezione. Vedere l'appendice 'Cambiare l'oggetto di inizio' per informazioni sui programmi di partenza.

Tipi di Dati COBOL

Managed COBOL riconosce sia I tipi di dati .NET sia i tradizionali del COBOL. I tipi di dati più frequentemente usati, come Strings e Integers hanno dei sinonimi in COBOL. Puoi riferirti agli altri tipi usando la parola chiave TYPE seguita dal nome del tipo come mostrato.

```
01 dob                  type DateTime.
01 display-output            String.
```

Vedi appendice per la lista completa dei tipi di dati COBOL.

Tipi di dati .NET

Il Framework .NET ha tipi di dati 'strong' ('forti'). Questo significa che un tipo di dato non può essere scambiato liberamente con un altro tipo. La conversione è consentita implicitamente o esplicitamente. Tutti i tipi di dati derivano dal tipo dato `System.Object,` che può essere assegnato ad ogni tipo di variabile.

Spazio dei Nomi (Namespace)

Un namespace, o in italiano spazio dei nomi, è una collezione di nomi di entità. Lo scopo dei namespace è quello di evitare confusione ed equivoci nel caso siano necessarie molte entità con nomi simili, fornendo il modo di raggruppare i nomi per categorie.

Un namespace si riferisce alle sue classi e metodi disponibili. Per esempio, il campo *dob* è definite essere di tipo `DateTime` che è raggruppato nel namespace *System:*

```
01 dob                  type System.DateTime.
```

.NET Framework dispone di altri tipi di namespaces quali Accessibility, System.Data, System, System.IO, and System.Runtime. Tutti i namespaces includono le relative classi e metodi.

Visual COBOL importa il namespace `System` per default, e quindi non vi è la necessità di prefissare classi da questo namespace. Ciò vuol dire che l'esempio di sopra può essere scritto anche come segue:

```
01 dob                  type DateTime.
```

Eliminando così il prefisso del Namespace

Aggiungi i namespace direttamente nelle proprietà del tuo progetto in modo da evitare di scrivere il nome complete del namespace quando scrivi il codice. Il namespace `System` è incluso per default. Per aggiungere altri namespace alle proprietà del tuo tuo progetto seguire i passi::

1. Click destro sul nome del progetto nel Solution Explorer e selezionare Properties.
   Sono visualizzate le proprietà del progetto nella pagina Proprietà.
2. Nella pagina Properties selezionare la linguetta References.
   Appare la linguetta delle References.
3. Scorrere verso il basso fino alla lista Imported Namespaces.
   Questa lista contiene tutti i namespace disponibili.
4. Selezionare la check box accanto ad ogni namespace della lista per importarla direttamente nel progetto.
   I namespace non selezionati richiederanno la scrittura del loro nome completo di prefisso nel codice.

Rimuovere I prefissi dei nomi dei namespace semplifica la scrittura del codice. Queste impostazioni sostituiscono la direttiva `$SET ILUSING` del compilatore.

Tipo dato DateTime

Nel programma esempio *CblDemol*, i campi *today* e *dob* nella definizione dei dati possono essere sostituiti dai tipi standard forniti in `System.DateTime`. Poiché lo spacename `System` è nella lista degli spacename importati per per default, la definizione per *today* e *dob* può essere sostituita con un semplice `DateTime`. I tipi derivati da `System.Object` devono essere dichiarati al livello 1. Poichè `DateTime` non ha sinonimi in COBOL, la parola chiave 'type' lo deve precedere. Nell'applicazione, la variabile *today* contiene la data corrente. Il suo valore è inizializzato usando il metodo interno a `DateTime Today()`.

Qui son riportate le nuove definizioni delle variabili

```
01 Today              typ ; DateTime      type DateTime::Today()
01 dob                typ ; DateTime.
01 test-date.
     03 test-month    PIC 99.
     03 test-day      PIC 99.
01 max-month          PIC 99.
01 max-days           PIC 99.
01 total-days         PIC 999.
01 full-name          PIC X(10)
01 display-output     PIC X(60)
01 greeting-output    PIC X(60)
01 txt1               PIC X(10) value  "Hello Mr
01 diff-output.
     03 txt2          PIC X(11) value   you have
     03 output-days   PIC ZZ9.
     03 txt3          PIC X(40) value   days until next birthday"
```

In questo esempio, una variabile derivata da `System.Object` fornisce il contenuto di default della variabile.

Puoi usare i tipi di dati di un namespace per sostituire la definizione dei dati senza dover modificare il codice scritto nella Procedure Division.

Poiché non è più necessario ricavare la data corrente per l'input, le line di codice dell'esempio precedente che definivano I campi *today* e *dob* possono essere rimosse.

Inizializzazione deglo Oggetti DateTime

Al campo *dob* è assegnato il valore dell'oggetto `DateTime` usando variabili convalidate come parte di un costruttore. Un costruttore è un metodo che è chiamato quando si inizializza un suo oggetto, ed è usato principalmente per inizializzare i parametri dell'oggetto.

Nell'applicazione di esempio il parametro anno è fornito dal campo *Year* dell'oggetto *today* come mostrato di seguito:

```
set dob to new type DateTime(today::Year, test-month, test-day)
```

> Nota: Le virgole che separano i parametri sono opzionali.

Confronto tra Oggetti DateTime

Gli oggetti `DateTime` forniscono una varietà di metodi per la manipolazione e il confronto. La differenza tra i campi *today* e *dob* può essere ottenuta in modo più semplice.

Per assicurarsi di non ottenere risultati negative, confrontiamo le date per assicurarci che il campo *dob* contenga un valore non minore del valore di *today.* L'oggetto `DateTime` fornisce un metodo `CompareTo()` che ritorna zero se le date sono uguali e +1 or -1 se sono differenti.

L'oggetto `DateTime` fornisce metodi che sommano o sottraggono singoli elementi di una data, sistemando automaticamente gli altri elementi, per fornire una data valida. (Ndt. Se il valore per il mese è maggiore di 12, viene incrementato l'anno di 1 e tolto 12 al valore del mese).

Nel programma d'esempio l'anno è incrementato di 1 se la data *dob* è minore della data odierna per assicurarsi che sia successiva. Il metodo da usare è `AddYears()` che accetta un intero come parametro.
Nella linea seguente l'istruzione `if` ritorna vero se *today* è maggiore di d*ob.* Se vero, l'elemento anno di *dob* è incrementato di uno.

```
if today::CompareTo(dob) > 0
      set dob to dob::AddYears(1)
end-if.
```

Usare il metodo `CompareTo()` come condizione per eseguire un'istruzione `perform` per scrivere usando una sola sezione per determinare il numero di giorni tra due oggetti `DateTime` come mostrato di seguito.

```
perform add-to-total until today::CompareTo(dob) = 0
```

La sezione *add-to-total* incrementa la variabile *total-days* e *today*. L'istruzione `add` incrementa la variabile *total-days*. Il metodo `AddDays()` incrementa l'elemento giorno della variabile *today*.

```
add-to-total SECTION.
     add 1 to total-days
     set today to today::AddDays(1).
```

Avendo così impostato la sezione *add-to-total,* le sezioni *add-day-difference, dd-mth-difference* e *increment- test-month* sono ridondanti e possono essere rimosse.

Tipo Dato String

Cambiamo la definizione della variabile `display-output` cambiando il `PIC X` con il tipo dato `string` per migliorare la manipolazione delle stringhe.

Integrazioni Tipo Dato String

Nell'esempio *CblDemo3* puoi cambiare la definizione della variabile *display-output* con il tipo dato `System.String` per ottenere le funzioni per la manipolazione di testo. Puoi anche convertire le costanti testo per mostrare che il loro valore iniziale non è cambiato. Il tipo derivato da `System.Object` deve essere dichiarato a livello `01`, in modo tale che il gruppo di variabili *diff-output* può essere rimosso. Puoi anche rimuovere la variabile *output-days* creata per visualizzare il numero di giorni soppressi, poichè non è più richiesto.

Il tipo `String` elimina la necessità di conscere la lunghezza delle costanti. L'oggetto è automaticamente dimensionato alla lunghezza del dato memorizzato

Nota: `String` ha un sinonimo in COBOL e quindi non è necessario farlo precedere dalla parola chiave `type`.
Qui vi sono le nuove definizioni delle variabili con *temp1* e *greeting-output* aggiunte per usi futuri:

```
01 today              type DateTimevalue type DateTime::Today().
01 dob                type DateTime
01 test-date.
     03 test-month    PIC 99.
     03 test-day      PIC 99.
01 max-month          PIC 99.
01 max-days           PIC 99.
01 total-days         PIC 999.
01 temp1              PIC 99.
01 full-name          PIC X(30).
01 display-output     String.
01 greeting-output    String.
01 txt1               String value "Hello Mr ".
01 txt2               String value " you have ".
01 txt3               String value " days until next birthday".
```

La definizione di una variabile di tipo `String` non richiede cambiamenti al codice della procedure division. Questo perchè una variabile proveniente da S`ystem.Object` fornisce il contenuto della variabile per default.

L'applicazione d'esempio copia il testo preso dalla console nel campo `display-output` di tipo `String` per poter usare i metodi disponibili

Funzioni String

Definire le variabili di tipo `String` non richiede modifiche al codice della procedure division. Il comportamento di default di una variabile derivate da `System.Object` fornisce il contenuto della variabile.

Visualizzare il Cognome

L'applicazione visualizza un saluto. Questo esercizio mostra I cambiamenti da fare per visualizzare un cognome.

L'applicazione visualizza il contenuto del campo *txtl* seguito dal cognome e una virgola (,). Ipotizzando che il cognomen sia l'ultima parte del nome complete, preceduto da uno spazio, dobbiamo trovare la posizione di questo spazio.

L'oggetto `String` permette di poter usare un metodo per trovare una sotto-stringa. Per impostare il codice necessario per questo metodo, per prima cosa impostare la variabile *display-output* per ricevere la sotto-sctringa. Il metodo `Trim()` rimuove tutti gli spazi iniziali e finali dalla stringa. Usare l'istruzione sotto ripoprtata per inserire la sotto-stringa nella variabile *greeting- output*.

```
set display-output to full-name
set greeting-output to display-output(???:)::Trim()
```

Dobbiamo ancora determinare la posizione iniziale del Cognome. Usare l'oggetto `String` per trovare l'ultima occorrenza di un carattere. Per ottenere ciò, usare il metodo `LastIndexOf()` con la prima posizione della stringa impostata a zero. Il metodo prende due parametric separate da due punti (:). Il primo parametro è la posizione iniziale della sottostringa, dove uno indica la prima posizione nella stringa originale. Il secondo parametron è la lunghezza della stringa da cercare. Se non trovato, viene ritornato il resto della stringa.

Usare quanto segue per impostare la variabile *tempi* in modo tale che la posizione dello spazio finale sia giusto prima del cognome. Incrementare il valore della variabile *tempi* di uno per dare la posizione iniziale del cognome.

```
Set display-output to full-name
set          temp1          to          display-  ')
output::Trim()::LastIndexOf('  set  greeting-output:Trim()
```

Per visualizzare le informazioni, usare la parola chiave `display` seguita dalla variabile:

```
set display-output to full-name
set          temp1          to          display-  ')
output::Trim()::LastIndexOf('  set  greeting-output:Trim()
to   display-output(temp1   +   1:):   display   txt1
```

Visualizzare in Nome

Ora che abbiamo modificato il codice per mostrare il cognome, vediamo le modifiche da apportare per visualizzare le seguenti informazioni:

- Il nome
- Il valore della variabile *txt2*
- Il numero dei giorni che mancano per il prossimo compleanno
- Il valore della variabile *txt3*.

Si assume che non vi siano spazi eccedenti.

L'esercizio precedente usa il metodo substring per visualizzare il cognome come parte di un saluto. Questo esercizio lo usa per trovare il nome dall'inizio della stringa. Si assume che vi sia un solo spazio tra il nome e il cognome.

Poichè dobbiamo determinare la lunghezza della stringa nome, l'istruzione sarà:

```
display display-output(1:???) txt2 total-days txt3
```

Usare il metodo `indexOf`() per determinare la posizione del primo carattere, sapendo che la posizione del primo carattere della stringa è zero. La lunghezza della stringa sarà la posizione del primo spazio dopo il nome. Quanto segue inizializza la variabile *tempi* con il primo spazio nella stringa *display-output*:

```
set temp1 to display-output::IndexOf(' ')
```

Mettendo queste due istruzioni insieme il risultato è la visualizzazione del nome con il contenuto delle variabili *txt2, total-days* e *txt3* come mostrato di seguito:

```
set temp1 to display-output::IndexOf(' ')
display display-output(1:temp1) txt2 total-days txt3
```

La concatenazione dell'istruzione d`isplay` accetta valori sia numerici che alfabetici.

Catturare l'Errore

Questa sezione si occupa di come gli errori nell'applicazione d'esempio tramite l'uso dell'istruzione `Try`. Decidere cosa fare quando si incontra un errore, dipende dall'applicazione. Questo esercizio visualizza un messaggio in caso di errore.

Nota: *CblDemo4* contiene il codice complete riportato in questa sezione. Per eseguire il programma, vedere l'appendice Cambiare l'oggetto di inizio.

La sezione *validate-test-date* nell'esempio, valuta che il giorno ed il mese della data di nascita siano nei limiti standard di una data. Essa controlla che che il giorno sia diverso da zero ed il mese compreso tra 1 e 12. La sezione *set-max-days* trova l'ultimo giorno del mese della data di nascita e ne controlla la validità.

```
validate-test-date SECTION.
     IF (test-day = 0)
         STOP "Invalid day field"
         STOP RUN END-IF
     IF ((test-month = 0) OR (test-month > 12))
         STOP "Invalid month field"
         STOP RUN END-IF
     MOVE test-month TO max-month.
     PERFORM set-max-days IF (test-day > max-days)
         STOP "Invalid day field"
         STOP RUN END-IF
```

La sezione *set-max-days* contiene un semplice codice che non consente di di gestire l'anno:

```
 set-max-daysSECTION
     EVALUATEmax-month
         WHEN  1 MOVE 31 TO max-days
         WHEN  2 MOVE 28 TO max-days
         WHEN  3 MOVE 31 TO max-days
         WHEN  4 MOVE 30 TO max-days
         WHEN  5 MOVE 31 TO max-days
         WHEN  6 MOVE 30 TO max-days
         WHEN  7 MOVE 31 TO max-days
         WHEN  8 MOVE 31 TO max-days
         WHEN  9 MOVE 30 TO max-days
         WHEN10 MOVE 31 TO max-days
         WHEN11 MOVE 30 TO max-days
         WHEN 12 MOVE 31 TO max-days
     END-EVALUATE
```

Questo è un modo accettabile per gestire gli errori. Comunque i dati di tipo `DateTime` sono controllati al momento della digitazione, anno compreso e viene generato un messaggio (exception) in caso di errore.

Gestione delle Exception con i dati di tipo DateTime

La seguente linea di codice nell'applicazione di esempio crea una istanza `DateTime` dalla informazioni fornite dall'utente:

```
set dob to new type DateTime(today::Year test-month test-day)
```

Nella linea di codice precedente, *validate-test-date* controlla che le informazioni fornite siano nel range corretto. Sostituire questa linea usando i dati forniti dal tipo dati `DateTime`. Il controllo è memorizzato come un tipo di dato `ArgumentException` con zero se è corretto. Il codice include una linea che imposta il tipo dato `ArgumentException` ad una variabile che accede al risultato del controllo.

```
01 argException      type ArgumentException.
```

Racchiudendo il costruttore in una istruzione `try` indichiamo all'applicazione di controllare il risultato della validazione. Se vi è qualsiasi exceptions viene eseguita la prima istruzione `catch`. L'istruzione `catch` qui sotto usa la variabile result per dire all'applicazione cosa fare quando l'utente fornisce dati che causano una exception.

```
try
     set dob to new type DateTime(today::Year test-month test-day)
catch argException
     display "I couldn't understand your date entry"
display argException::Message stop run end-try
```

Il codice di sopra sostituisce le funzionalità fornite da *validate-test-date* e *set-max-days* quindi quel codice può essere eliminato. Si possono eliminare anche le variabili *max-month* e *max-days*.

Lavorare con le Classi

L'applicazione d'esempio contiene funzioni ed informazioni utili per altri programmi. Ma piuttosto che copiare il codice ovunque possano servire le sue funzionalità, è preferibile creare una classe che contenga quelle funzionalità. La classe poi potrà essere usata dale alter applicazioni.

Nota: Nella soluzione d'esempio *CblDemo5* e la classe *Person* contengono il codice completo basato su esercizi completati in questa sezione. Vedere l'appendice Cambiare l'oggetto di inizio per le informazioni sul programma di partenza.

Creare una Classe

Una classe è un tipo di definizione che contiene campi e metodi che generalmente rappresentano una cosa o un'azione. Poichè l'applicazione d'esempio include codice che crea, memorizza e visualizza informazioni riguardanti un individuo, creiamo una classe *Person* per queste informazioni.

1. Click destroy sul nome del progetto nella finestra di Solution Explorer e selezionare Add > New Item.... Appare la finestra Add New Item.
2. Click su COBOL Items > Pannello Managed dei Template installati, quindi selezionare COBOL Class dalla lista.
3. Digitare `Person.cbl` nel campo Name. Il Name indica sia il nome del file sia il nome della classe da creare.
4. Click Add per creare un file chiamato *Person.cbl* contenente il template vuoto della classe *Person*. Il template di classe è molto simile al template di programma e contiene un metodo vuoto.

```
class-id CblSolution.Person.

working-storage section.

method-id InstanceMethod.
local-storage section.
procedure division.

   goback. end
       method.

end class.
```

5. Eliminare *CblSolution* dall'identificatore di classe sulla prima linea del template della classe.
   Il prefisso del nome della classe di default serve ad assicurare nomi di classe unici nell'applicazione Questo non è richiesto nella nostra applicazione d'esempio, quindi possiamo eliminarlo.

Riempire la Classe

Copiare I campi importanti e le funzioni dal codice esistente nella classe *Person*. Assicurarsi di copiare solo I campi che servono per evitare confusioni quando si usa la classe in altre applicazioni.

1. Copiare le variabili che servono nella classe.
   Nel codice COBOL *dob* e *full-name* si riferiscono direttamente alla persona. Copiare queste variabili nella classe convertente il nome nel tipo dato `String` per assicurare il corretto dimensionameto, come mostrato di seguito.
   ```
   working-storage section.
   01 full-name   string.
   01 dob         type DateTime.
   ```
2. Aggiornare il codice per permettere l'accesso agli attributi.
   Le variabili della classe (detti anche attribute) includono una proprietà che indicate l'accessibilità dall'esterno della classe. La proprietà è impostata su `private` per default. Questo indica nessun accesso dall'esterno. Per permettere l'accesso dall'esterno della classe, impostare la proprietà come `public` e chiamarla per referenza come mostrato di seguito.
   ```
   working-storage section.
   01 full-name   string publicproperty  as"FullName".
   01 dob         type DateTime publicproperty  as "DateOfBirth".
   ```
3. Creare un costruttore che permatta l'uso dei campi della classe.
   Un costruttore è un metodo che istanzia ed inizializza I campi definiti nella classe. Per assicurare che la classe *Person* contenga sempre un nome e una data di nascita, fornire questi valori come parametric nel costruttore, come mostrato di seguito. Senza I parametric il metodo genera un errore che può essere catturato tramite l'istruzione `Try`.
   ```
   method-id New public.
   procedure division using by value fullname as string
                                     birthday as type DateTime.
       set full-name to fullname
   set dob to birthday goback. end
   method.
   ```
4. Eliminare il metodo di default.
   Quando si crea una classe, si crea anche un template del metodo. Il template del metodo non è richiesto, quindi eliminiamo il seguente codice.
   ```
   method-id InstanceMethod. local-storage section. procedure division.
      goback. end
          method.
   ```

La classe *Person* contiene il nome completo e la data di nascita di una persona potrà essere create successivamente solo se sono forniti il nome complete e la data di nascita.

Aggiungere Metodi alla Classe

Ora che la classe *Person* è disponibile, creiamo un metodo per le funzioni che si riferiscono alla persona. Le funzioni che si riferiscono alla persona sono le seguenti:

1. Numero di giorni per il prossimo compleanno.
   Attualmente l'applicazione prende la data di nascita e somma l'anno di nascita per ottenere il prossimo compleanno.. Poi è incrementata la data corrente fino a quando le due date non coincidono, incrementando parallelamente un contatore. Creare un metodo che fornisce il prossimo compleanno, quindi usare quella data per determinare il numero di giorni che mancano al prossimo compleanno.

   Quanto segue mostra un metodo che fornisce il prossimo compleanno. Esso definisce una variabile locale chiamata *today* che contiene la data odierna. E' definita anche una variabile temporanea chiamata *return-value.* Poichè questo è parte del metodo e include il tipo di variabile, il run-time crea automaticamente la variabile. Cambiare l'istruzione `set` dal codice originale per usare *return-value:*

```
method-id NextBirthday.
01 today             type DateTime value type DateTime::Today().
procedure division returning return-value as type DateTime.
    set return-value to new DateTime(today::Year, dob::Month, dob::Day)
end method.
```

Questo metodo fornisce la data del prossimo compleanno. Creare un secondo metodo per confrontarla con la data odierna e ritornare il numero di giorni.

Per ottenere ciò, creare una variabile per fornire la data odierna. E' create una variabile temporanea per valorizzare il numero di giorni. Per usdare il metodo `NextBirthday()` definite nella classe corrente, mettere il prefisso `self`. Questo fornisce un oggetto `DateTime` contenenet la data del prossimo compleanno. `DateTime` ha il metodo `Subtract` che fornisce la differenza tra il suo contenuto e un oggetto di tipo `DateTime` chiamato `TimeSpan`. L'oggetto `TimeSpan` ha un metodo per fornire il numero di giorni. Il codice risultante è il seguente.

```
method-id DaysUntilBirthday.
01 today             type DateTime value type DateTime::Today().
procedure division returning return-value as binary-long.
    set return-value to self::NextBirthday()::Subtract(today)::Days
end method.
```

Poichè vi sono due metodi che hanno *today* come variabile locale, possiamo provare a rendere questa variabile un campo della classe per poterlo riusare. Al moment oil valore della variabile è la data e orario del momento dell'uso del metodo. Se la spostiamo, il valore deve essere reimpostato prima di usarlo. Un'alternativa potrebbe essere definire *DateTime* come parte della chiamata al metodo come riportato nel codice seguente.

```
method-id DaysUntilBirthday.
procedure division returning return-value as binary-long. set return-value to
        self::NextBirthday()::Subtract(type DateTime::Today())::Days
end method.
```

2. Funzione Cognome.

   L'applicazione estrae il cognomen e lo visualizza come parte di una frase. Il cognomen fa sempre parte del nome complete di una persona. Creare un metodo che ritorna il cognome come una stringa. Definiamo una variabile temporanea che contenga la posizione del cognome. Quindi usiamo la posizione per individuare il dato da ritornare. Usare il seguente codice per aggiornare l'applicazione in modo che inizializzi la variabile temporanea.

```
method-id FamilyName.
01 temp1             PIC 99.
           procedure division returning return-value as type String.
                set temp1 TO full-name::Trim()::LastIndexOf(' ')
        set return-value to full-name(temp1 + 1:)::Trim()
    end method.
```

3. Funzione Nome.

   Come per il cognome, il programma estrae Nome e lo visualizza come parte di una frase. Creare un metodo per fornire il Nome come una S`tring`. Usare il codice esistente dell'applicazione d'esempio come base:

```
method-id FirstName.
01 temp1             PIC 99.
procedure division returning return-value as type String. set temp1 TO
full-name::IndexOf(' ') set return-value to full-name(1:temp1) end
method.
```

   Puoi combinare le due line di codice qui sopra per motivi di efficienza come mostrato di seguito. Considera il codice come due linee per motivi di leggibilità:

```
method-id FirstName.
procedure division returning return-value as type String.
    set return-value to full-name(1:full-name::IndexOf(' '))
end method.
```

Usare la Classe Persona

Aggiornare l'applicazione con la classe *Person* creata.

1. Creare una variabile per memorizzare l'istanza della classe.

```
01 person1           type Person.
```

2. Cambiare la variabile *test-date* per accettare l'anno di nascita.
   Dobbiamo prendere il input la data di nascita.
3. Istanziare la variabile nel codice.
   Questo lo si fa nello stesso modo della variabile *dob* e, in modo simile, possiaamo gestire il codice per gli errori. Il costruttore della classe *Person* richiede un oggetto `DateTime` come secondo argomento, quindi passiamo una istanza di `DateTime` creata usando le informazioni fornite dall'utente come mostrato di seguito:

```
try
    set person1 to new type Person(full-name
                   new DateTime(test-year test-month test-day))
catch argException
    display "Invalid arguments provided" display
argException::Message stop run end-try
```

4. Sostituire il codice dell'applicazione con questo metodo di classe.
   Tutto il codice che determina il Nome, il Cognome, e Il numero di giorni fino al prossimo compleanno, può essere sostituito con i metodi appropriati della classe *Person* e la sezione *add-to- total* può ora essere rimossa.
5. Riordiniamo le variabili definite.
   Con alcune delle funzionalità spostate nella classe *Person* class ed il codice risultante semplificato, le variabili definite nell'applicazione d'esempio possono essere riviste e quelle ridondanti rimosse.

Ulteriori Miglioramenti

Ora che il codice per l'applicazione di esempio è stato semplificato e le funzionalità spostate nella classe, le migliorie al codice sono molto più semplici da realizzare. Codice e variabili ridondanti sono stati rimossi.
Con l'introduzione delle classi, che forniscono metodi standard, le nuove applicazioni sono più semplici e più veloci da scrivere. Puoi aggiornare le tue applicazioni esistenti per usare le nuove classi spostando il codice, ottenendo maggior efficienza nello sviluppo del codice.

Con l'aggiornamento dell'applicazione d'esempio, le costanti testo sono contenute in tre variabili. Queste variabili possono essere eliminate, sostituendole con direttamente con stringhe costati, oppure possono essere fornite tramite una seconda classe. L'applicazione console può essere completamente sostituita con l'applicazione Windows or ache le funzionalità sono state spostate in una classe.

Il messaggio di benvenuto "Hello Mr " presuppone che che l'uente sia Sempre un uomo; comunque l'applicazione potrebbe chiedere di inserire il sesso dell'utente. La classe *Person* potrebbe quindi avere un nuovo metodo che fornisca il messaggio adeguato, per esempio semplicemente "Hello".

Puoi modificare ulteriormente l'applicazione con questi suggerimenti o altri personali. Di seguito c'è l'applicazione console finale:

```
                                        next birthday".
```

```
PROGRAM-ID.
CblDemo5.



01 person1            type Person.
01 test-date.
    03 test-year      PIC 9999.
    03 test-month     PIC 99.
    03 test-day       PIC 99.
01 full-name          PIC X(30).
01 txt1               String value "Hello Mr ".
01 txt2               String value " you have "
01 txt3               String value " days until
01 argException       type ArgumentException.

    PROCEDURE DIVISION.
        display "Enter your full name" accept full-name
        display "Enter your date of birth in yyyymmdd format"
        accept test-date
        try
            set person1 to new type Person(full-name
                              new DateTime(test-year test-month test-day))
        catch argException
            display "Invalid arguments provided"
        display argException::Message stop run end-
        try
        display txt1 & person1::familyName() & ","
        display person1::firstName() & txt2 & person1::daysUntilBirthday() & txt3
        stop run


    END PROGRAM CblDemo5.
```

Qui il codice finale della classe *Person*:

```
    class-id Person.
    01 full-name             string public property as "FullName".
    01 dob               type DateTime public property as "DateOfBirth".

    method-id New public.
    procedure division using by value fullname as string
                                      birthday as type DateTime.
        set full-name to fullname set dob to birthday goback. end method.

    method-id NextBirthday.
    01 today             type DateTime value type DateTime::Today().
```

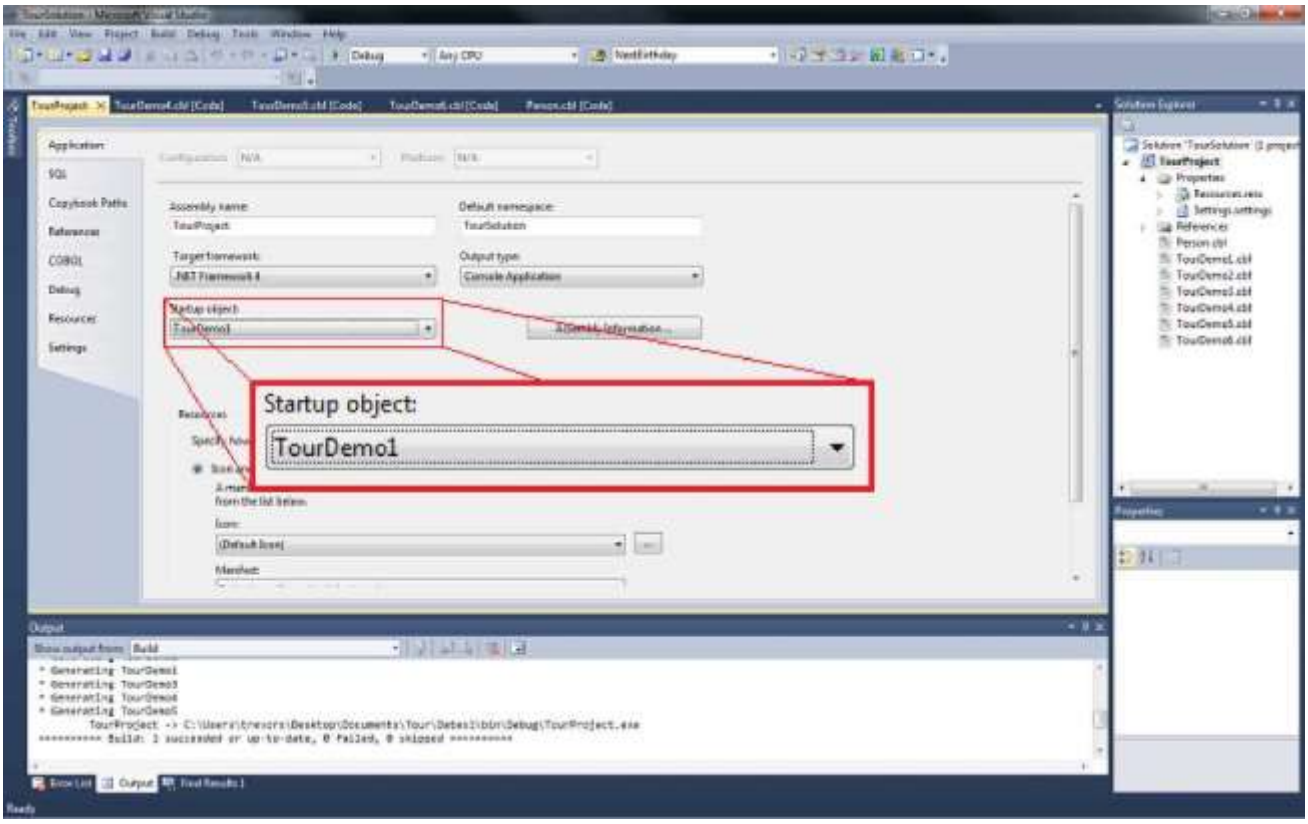
```
procedure division returning return-value as type DateTime.
    set return-value to new DateTime(today::Year, dob::Month,
dob::Day) end method.

method-id DaysUntilBirthday.
procedure division returning return-value as binary-long.
    set return-value to
self::NextBirthday()::Subtract(type
DateTime::Today())::Days end method.

method-id FamilyName.
01 temp1                   PIC 99.
procedure division returning return-value as type
String. set temp1 TO full-name::Trim()::LastIndexOf(' ')
set return-value to full-name(temp1 + 1:)::Trim() end
method.

method-id FirstName.
procedure division returning return-value as type String.
    set return-value to full-name(1:full-name::IndexOf('
')) end method. end class.
```

Appendice

Cambiare l'oggetto di inizio

L'oggetto iniziale di un progetto è il punto di ingresso quando inizia un'applicazione. Quando crei un progetto è impostato inizialmente al template del programma creato in quel momento. Puoi voler cambiare questo poichè, ad esempio, stai sostituendo l'interfaccia utente da console a Windows. Scambiando l'oggetto iniziale tra I due programmi, puoi verificare se le funzionalità sono mantenute in entrambe le version. I passi seguenti illustrano come cambiare l'oggetto iniziale.

1. Click destroy sul progetto nella Solution Explorer.
   Viene visualizzata la pagina delle Proprietà.
2. Selezionare l'Applicazione dalla lista a sinistra.
   Le proprietà dell'applicazione vengono portate in primo piano nella pagina Properties.
3. Selezionare l'oggetto dalla list box contrassegnata Startup object.
   La prossima volta che verrà eseguita l'applicazione, inizierà con l'oggetto selezionato.



Riferimenti ai Tipi di Dati

La tavola sottostante mostra i tipi di dati equivalenti tra il Cobol tradizionale e quello attuale. Sono stati inseriti anche I tipi di dati relative all'ambiente .NET. Raccomandiamo fortemente di usare I tipi del COBOL attuale poichè sono validi sia in ambiente Java Virtual Machine (JVM) che in .NET.

Attuale COBOL	**Tradizionale COBOL**	**.NET Class**
binary-char	pic s9(2) comp-5	System.SByte
binary-char unsigned	pic 9(2) comp-5	System.Byte
binary-short	pic s9(4) comp-5	System.Int16
binary-short unsigned	pic 9(4) comp-5	System.UInt16
binary-long	pic s9(9) comp-5	System.Int32
binary-long unsigned	pic 9(9) comp-5	System.UInt32
binary-double	pic s9(18) comp-5	System.Int64
binary-double unsigned character	pic 9(18) comp-5	System.UInt64 System.Char
float-short	comp-1	System.Single
float-long	comp-2	System.Double
condition-value		System.Boolean
decimal		System.Decimal
object		System.Object
string		*System.String*

Lavorare con IntelliSense

IntelliSense® è il termine generale che indica il prompts che appare quando il cursore passa sul codice e per le indicazioni che assistono lo sviluppo del codice. Visual COBOL fa uso intensive di IntelliSense, fornendo feedback istantaneo ed incrementando la produttività. Le caratteristiche principali di IntelliSense includono le seguenti.

Informazioni sulle Variabili	Mettendo il cursore sulla variabile, mostra un tooltip con le informazioni riguardanti la variabile, incluso il numero di volte che è stata referenziata, il tipo e la sua lunghezza.
Evidenziazione dell'Errore	Il codice è costantemente compilato in background,in modo da trovare qualsiasi cosa possa generare errori. E' controllata la sintassi e vengono forniti suggerimenti. La validazione conferma se la variabile è stata definita.
Colori	Mentre scrivi il codice, esso è validato e colorato. Istruzioni, variabili, costanti e nomi di sezione hanno colori differenti che possono essere personalizzati.
Completamento	Mentre scrivi le istruzioni, è visualizzata la lista delle variabili per assisterti nel completamento dell'istruzione Per accedere alla lista, click destro nell'editor del codice per selezionare l'opzione Suggest Word.
Frammenti di Codice	Questi sono template di sezioni comuni di codice che sono generate e copiate direttamente nel codice. Per inserire un frammento di codice, click destroy nel punto in cui si deve inserire il frammento e selezionare Insert Snippet. Doppio click per selezionare il frammento richiesto dalla lista fornita.