RDz Tutorial 02
Local COBOL

Version 01, August 2012

Dieses Tutorial ist der zweite Teil einer Reihe. Es wird Sie durch die Schritte zur Verwendung des z/OS Application Development Komponente von WebSphere Developer for System z Version 7.0 mit lokalen Systemen führen. In diesem Tutorial werden wir mit einem Sample COBOL-Programm zu arbeiten, einschließlich der Bearbeitung, Kompilierung und Debugging eines COBOL-Programms. Nach Beendigung des Tutorials sollten Sie mit den grundlegenden RDz Workstation Einrichtungen vertraut sein, und sollte Kenntnisse erworbenen haben, um einfache COBOL-Anwendungen mit RDz entwickeln.

Dieses Tutorial (und die folgenden Tutorials) verwenden RDz Version 7.0. Es läuft in einer VMWare Player virtuellen Maschine auf unserem RDz Server. Es wird davon ausgegangen, dass Sie bereits eine Verbindung zu unserem RDz Server hergestellt haben (siehe Tutorial01).

Es existieren mehrere Cobol Dialekte. "IBM COBOL for Windows" (die Cobol Compiler Komponente von RDz) ist zu 100,00 % compatible mit dem „IBM Enterprise COBOL“ Compiler für z/OS (siehe http://www-01.ibm.com/software/awdtools/cobol/zos/). Es existieren zahlreiche Unterschiede zu den Micro Focus und Fujitsu (aka Alchamy) COBOL Compilern, die häufig in Windows Umgebungen eingesetzt werden.

Übersicht:

In dem Abschnitt 2 laden Sie in Ihr Projekt ein Cobol Beispiel Programm, welches das Schreiben eines eigenen Cobol Programms erspart.

Abschnitt 3 erläutert die vielen Verbesserungen und Erweiterungen, die RDz gegenüber dem ursprünglichen TSO ISPF Editor anbietet. Wenn Sie nicht wollen, müssen Sie diese nicht benutzen. Die Erweiterungen machen aus ISPF einen modernen, State of the Art Editor.

Abschnitt 5 erläutert die modernen Debug Funktionen, welche die TSO Funktionen deutlich erweitern und ergänzen.

Die ursprüngliche Version dieses Tutorials wurde von Frau Isabel Arnold in einer z / OS Sommer-Klasse an der Universität Hamburg vorgestellt und von Herrn Niels Michaelsen übernommen. Sie wurde geändert und angepasst durch Herrn Matthias Beyerle an der Fakultät für Informatik der Universität Tübingen.

Dieses Tutorium (und die folgenden Tutorials) sind gehostet auf dem z/OS 1.8 System leia.informatik.uni-leipzig.de oder 139.18.4.30 der Abteilung Technische Informatik der Universität Leipzig und hobbit.cs.informatik.uni-tuebingen.de oder 134.2.205.54 der Abteilung Technische Informatik der Universität Tübingen. Die RDz Installation auf diesem System wurde von Herrn Uwe Denneler, Elisabeth Puritscher und Herr Martin Benjamin Storz von IBM durchgeführt und unterstützt von Frau Martina Koederitz, Herb Kircher und Herr Andresa Hermelink, ebenfalls IBM. Lokale Unterstützung erfolgte durch Herrn Frank Güttler an der Universität Leipzig, sowie Andreas Nagel an der Universität Tübingen.

Dieses Tutorial verwendet die folgenden Konventionen

1k bedeutet 1 Klick mit der linken Maustaste
2k bedeutet 2 Klick mit der linken Maustaste
1kr bedeutet 1 Klick mit der rechten Maustaste

Selbst-Test

- Findet der Compile Vorgang in diesem Tutorialauf der Workstation, auf dem RDz Server, oder auf dem z/OS System statt ?
- Ist der erzeugte Maschinencode (Binaries) sowohl unter Windows als auch unter z/OS lauffähig ? Nur unter z/OS ? Nur unter Windows ?

Aufgabe: ***Erstellen Sie ein einfaches Cobol Beispiel Programm, übersetzen (compile sie es und führen es aus. Die Durcharbeitung von Abschnitt 3 ist eine Option, macht Sie aber mit moderner Software entwicklung vertraut.***

Aufgabe: ***Als Option machen Sie sich mit den in Abschnitt 5 beschriebenen RDz Test/Debug Funktionen vertraut.***

1. Starting WDz

Erstellen Sie eine Remote Desk Top Verbindung zu unserem RDz Server, wie bereits im letzten Tutorial dargestellt.

Abb. 1.1

Starten RDz, wenn es nicht bereits ausgeführt wird

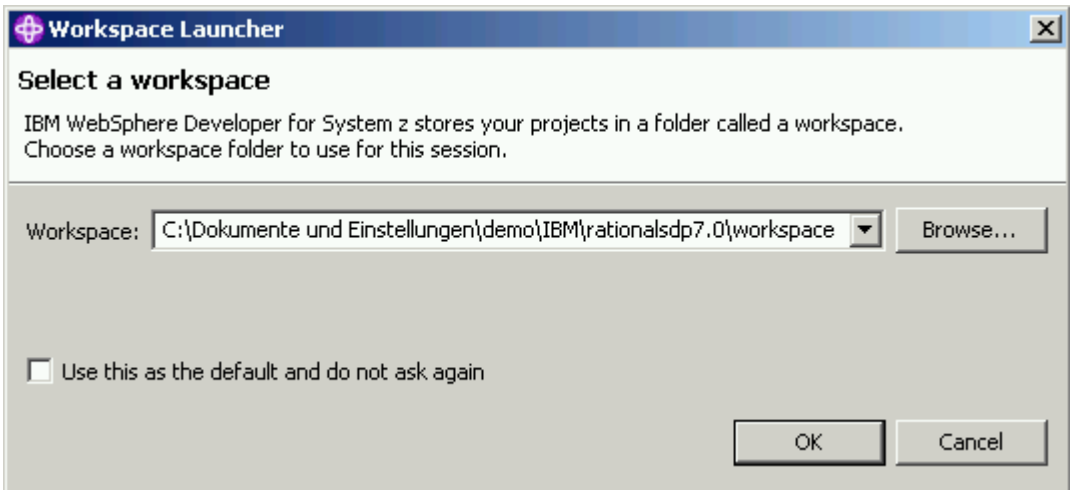


Abb. 1.2

Da auf dem remote Windows Server für jeden Benutzer eine eigene Festplatte z:\ eingerichtet wird, schlagen wir vor, dass sie statt des vorgeschlagenen Verzeichnisses den Wert „Z:\Workspace01“ benutzen.

1k auf OK, das Starten dauert ca. 1 Minute

Die Standard-Perspektive nach der Installation von RDz ist die z/OS Projects Perspektive. Um es zu öffnen machen Sie folgendes: Von der z/OS Projects Perspektive, wählen Sie in der Menüleiste Datei?-- New?--Beispiel ... Sie werden die z / OS Projects Perspektive zum Erstellen und Bearbeiten der Projekte und Projekt-Dateien auf Ihrer Workstation verwenden, auf Ihrem z/OS-System, oder in einer verteilten Umgebung. Beachten Sie, dass Perspektiven viele Ansichten (Views) haben. Um eine Ansicht zu schließen, klicken Sie auf die Schaltfläche Schließen mit einem X in der rechten oberen Ecke der Registrierkarte.

2. Laden eines Local COBOL Sample Programms

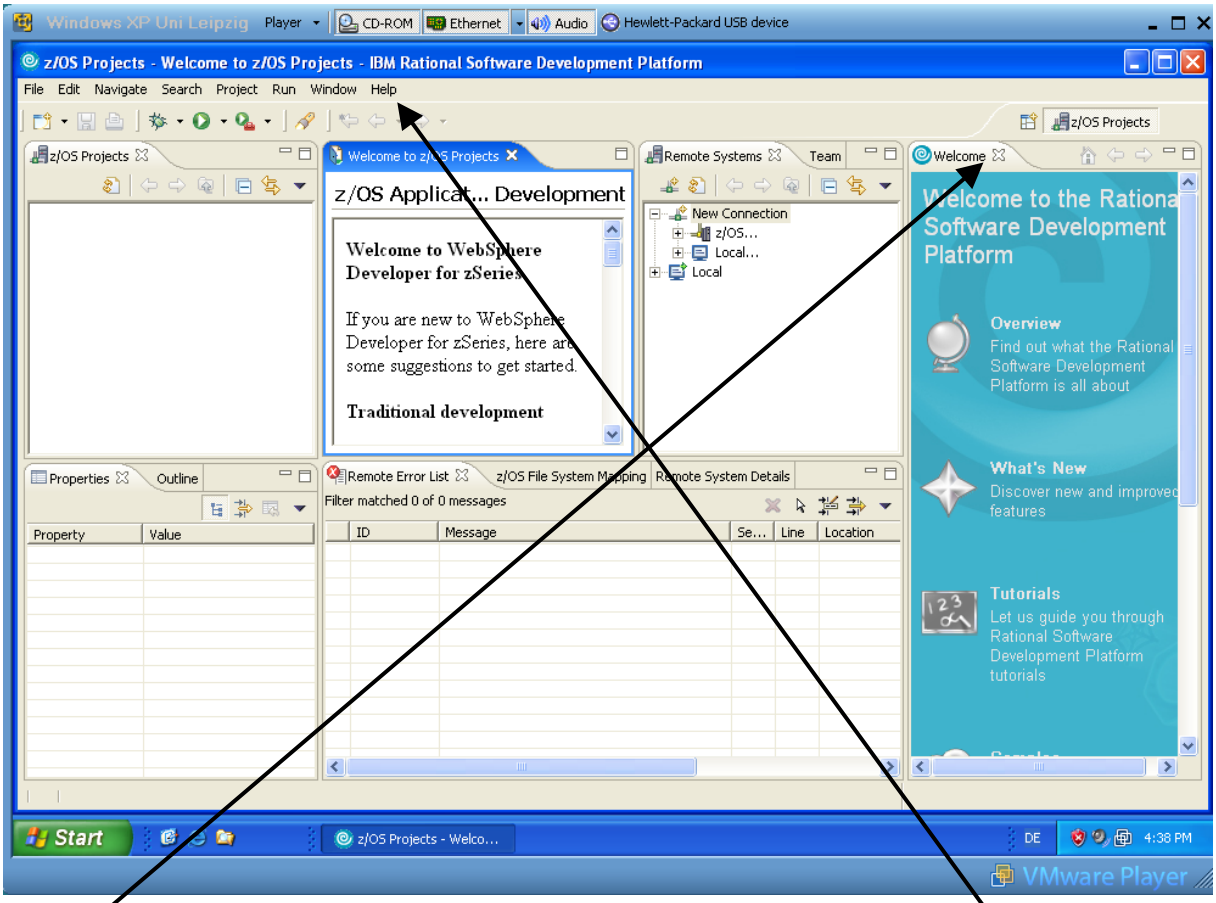


Abb. 2.1

Dies ist der RDz Welcome Screen, wie er beim erstmaligen Login erscheint.

1k auf dem Kreuz neben Willkommen auf das rechte Fenster zu schließen. Mit „Hilfe“ können Sie jederzeit neu zu erstellen.

Die Standard-Perspektive nach der Installation von Rdz ist die z/OS Projects Perspektive. Sie benutzen diese zum Erstellen un Bearbeiten von Projekten und Projekt-Dateien auf Ihrer Workstation, auf Ihrem z/OS-System, oder auch in einer verteilten Umgebung. Beachten Sie, dass Perspektiven viele Views haben.

Um einen View zu schließen, klicken Sie auf die Schaltfläche Schließen mit einem X in der rechten oberen Ecke der Registrierkarte. Um es zu öffnen, klicken Sie auf Window ? Show View? Andere, und wählen Sie den gewünschte View, den Sie öffnen möchten.

Als ersten Schritt laden Sie nun ein Beispiel COBOL-Programm in Ihrem lokalen Workspace.

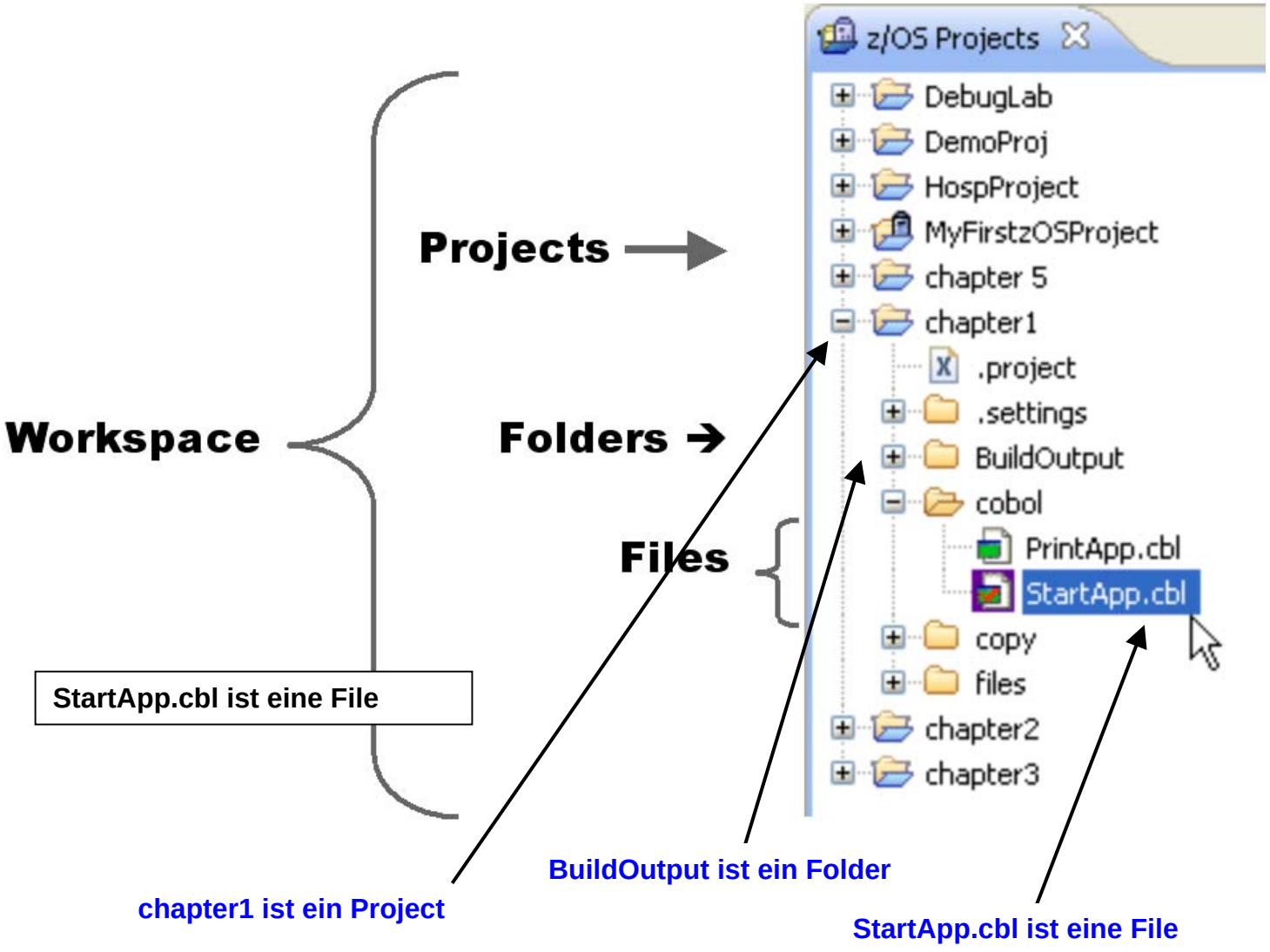
2.1 Workspace, Projects und Files

Wir werden mit unterschiedlichen COBOL Resourcen arbeiten: Pogramme, Copybooks, Compiler Listings, ausführbaren Programmen (executables), Daten Files usw.

RDz speichert, organisiert und managed die zu Ihren Projekten gehörigen Resourcen innerhalb eines Workspaces. Workspace Resourcen sind organisiert in:

- **Projekt(e)**
- **Folder**
- **Files.**

Nachdem Sie RDz gestartet und einen Workspace selektiert haben, haben Sie Zugriff auf alle Files und Folders innerhalb des Workspace. Hierzu ist der Workspace typischerweise in ein oder in mehrere Projekte organisiert:



Was ist ein Projekt, und was sind die Bestandteile eines Projektes ?

Workspace Projekte organisieren und verwalten zueinander gehörige Anwendungs-Resourcen. Projekte innerhalb eines Workspace können organisiert werden:

- **Batch Anwendung .. vs.. Online Anwendung**
- **Line of Business Anwendungens: Accounts Payable, Inventory, Claims, Manufacturing Part Assembly, usw.**
- **Common – oder shared Projekte, die Daten oder Record Definitionen enthalten können, welche von mehreren unterschiedlichen anwendungen gemeinsam genutzt werden.**

Es existieren eine Reihe von spezifischen RDz Projektarten, mit denen wir uns später noch beschäftigen werden. Beispiele sind z/OS Local Projects, MVS Subprojects, usw.

Projekte enthalten Build Configuration Files, auch als Property Files bezeichnet. Diese spezifizieren Generierungs-Optionen für Ihre Projekt Komponenten ihres Projektes.

Projekte können aus mehreren Foldern bestehen.

z/OS COBOL Projekte enthalten typischerweise die folgenden high-level Folder:

- **cobol,**
- **BuildOutput,**
- **copy**

Was befindet sich in dem Cobol Folder (Directory) ?

\cobol ist der Default Folder (der highest level folder) innerhalb eines Projektes, unter dem alle COBOL Ressourcen (Quell Programme) organisiert sind.

Diese COBOL Resourcen können z.B. Sub-Folders einschließen:

```
cobol
    batch
        program1.cbl
        program2.cbl
        …
    online
        program3.cbl
        …
```

oder enthalten alle COBOL Quellprogramm Files:

```
cobol
    program1.cbl
    program2.cbl
    …
```

Was befindet sich in dem BuildOutput Folder (Directory) ?

\BuildOutput\ ist der Default Folder (der highest level folder) innerhalb eines Projektes, unter dem alle übersetzten (compiled) COBOL Programme organisiert sind.

Diese Resourcen enthalten:

- **.OBJ – object modules**
- **.exe – COBOL executables – können ausgeführt oder debugged werden**
- **.adt – an internal-system file used by RDz when you do source-level debugging produced by a Compiler Directive)**
- **.lst – listing file**
  - **Shows highest error condition**
  - **Sorted XRef of COBOL variables**
  - **Other program info**

Der Folder copy enthält Copy Files, auch als Copybooks bezeichnet. Ein Copybook ist eine Bezeichnung für eine Dateibeschreibung oder eine Routine, die mehrfach in einem Cobol Programm und dessen Subroutines in identischer Form benötigt wird. Ein copy Statement bewirkt etwas ähnliches wie ein include Statement in C/C++. Ein Beispiel Programm finden Sie unter http://www.csis.ul.ie/cobol/course/Copy.htm. Es ist auf den beiden folgenden Seiten wiedergegeben.

Source Text

```
      $SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID. COPYEG1.
AUTHOR.  Michael Coughlan.

ENVIRONMENT DIVISION.
FILE-CONTROL.
    SELECT StudentFile ASSIGN TO "STUDENTS.DAT"
    ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD StudentFile.
COPY COPYFILE1.

PROCEDURE DIVISION.
BeginProg.
   OPEN INPUT StudentFile
   READ StudentFile
      AT END SET EndOfSF TO TRUE
   END-READ
   PERFORM UNTIL EndOfSF
      DISPLAY StudentNumber SPACE StudentName SPACE
              CourseCode SPACE FeesOwed SPACE AmountPaid
      READ StudentFile
         AT END SET EndOfSF TO TRUE
      END-READ
   END-PERFORM
   STOP RUN.
```

Text in COPYFILE1

```
01  StudentRec.
    88  EndOfSF      VALUE HIGH-VALUES.
    02  StudentNumber               PIC 9(7).
    02  StudentName                 PIC X(60).
    02  CourseCode                  PIC X(4).
    02  FeesOwed                    PIC 9(4).
    02  AmountPaid                  PIC 9(4)V99.
```

Text in COPYFILE1

```
01  StudentRec.
    88  EndOfSF      VALUE HIGH-VALUES.
    02  StudentNumber                PIC 9(7).
    02  StudentName                  PIC X(60).
    02  CourseCode                   PIC X(4).
    02  FeesOwed                     PIC 9(4).
    02  AmountPaid                   PIC 9(4)V99.
```

Source Text after processing

```
      $SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID. COPYEG1.
AUTHOR.  Michael Coughlan.

ENVIRONMENT DIVISION.
FILE-CONTROL.
    SELECT StudentFile ASSIGN TO "STUDENTS.DAT"
    ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD StudentFile.
COPY CopyFile1.
01  StudentRec.
    88  EndOfSF      VALUE HIGH-VALUES.
    02  StudentNumber                PIC 9(7).
    02  StudentName                  PIC X(60).
    02  CourseCode                   PIC X(4).
    02  FeesOwed                     PIC 9(4).
    02  AmountPaid                   PIC 9(4)V99.

PROCEDURE DIVISION.
BeginProg.
   OPEN INPUT StudentFile
   READ StudentFile
      AT END SET EndOfSF TO TRUE
   END-READ
   PERFORM UNTIL EndOfSF
      DISPLAY StudentNumber SPACE StudentName SPACE
              CourseCode SPACE FeesOwed SPACE AmountPaid
      READ StudentFile
         AT END SET EndOfSF TO TRUE
      END-READ
   END-PERFORM
   STOP RUN.
```

Selbst-Test

- Muss ein cobol Programm unbedingt Copybooks enthalten ?

2.2 Laden eines Beispielprogramms

Wenn erforderlich: "z/OS-Projekte"-Perspektive öffnen
Von der z/OS Projects Perspective, wählen Sie File → New → Example at the menu bar.

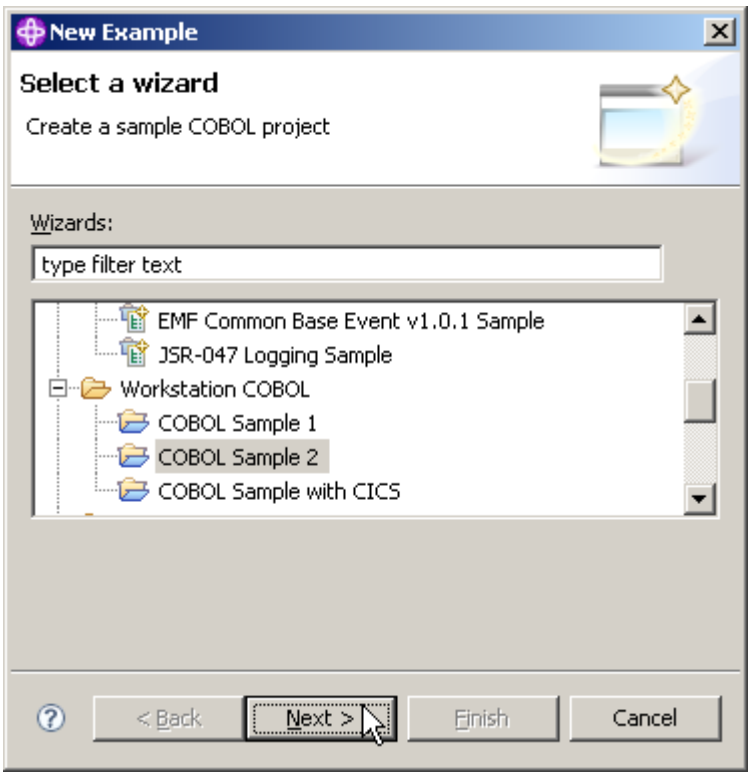


Abb. 2.2

"Workstation COBOL" erweitern

In dem Select a wizard panel, selektieren Sie Workstation COBOL - COBOL Sample 2.
1k auf Next.

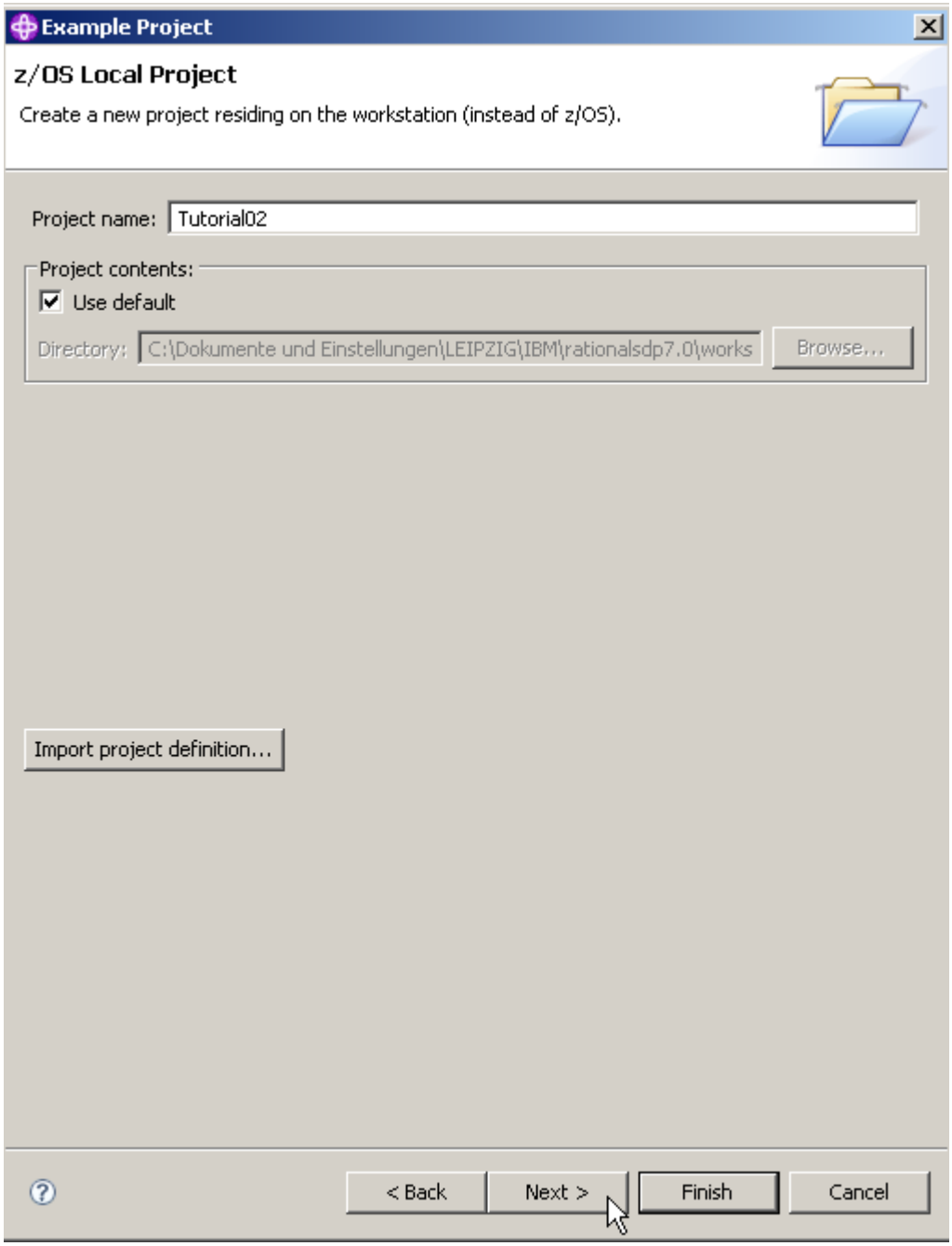


Abb. 2.3

Im z/OS Local Project Panel, geben Sie Tutorial02 als den Projekt Namen ein, setzen Sie ein Häkchen auf Use default, und click Next.

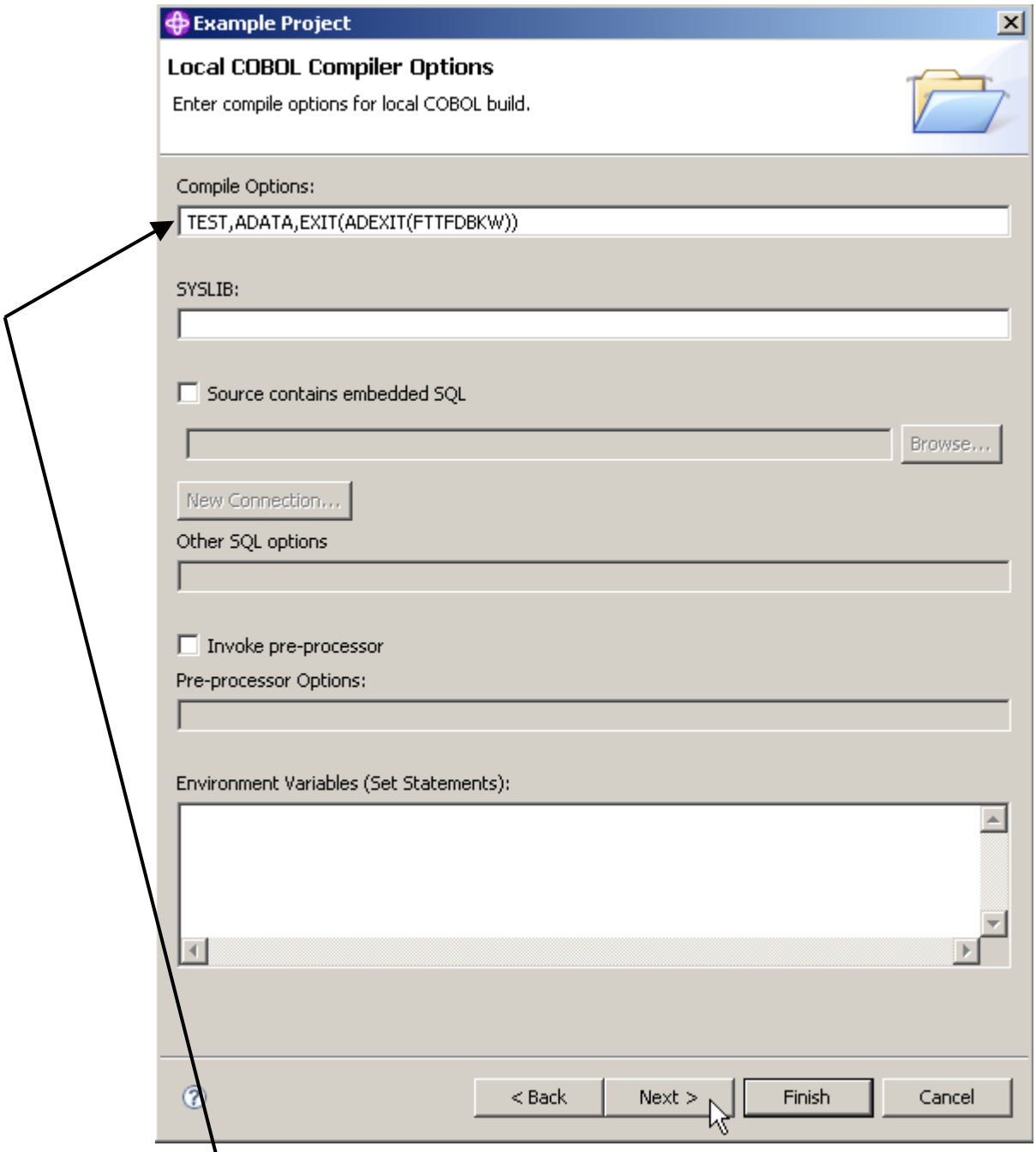


Abb. 2.4

Im Local COBOL Compiler Options Panel verifizieren Sie, dass TEST unter Compile Options spezifiziert ist. Lassen Sie die Defaults für den Rest unverändert.

Beachten Sie die default Local COBOL Compiler Options für Ihr COBOL Programm. Die TEST Compiler Option beinhaltet Sub-Optionen für die Definition von debugging Funktionen.

Verifizieren Sie, dass TEST spezifiziert ist unter Compile Options. Der Screen sollte aussehen wie dargestellt. Click Next.

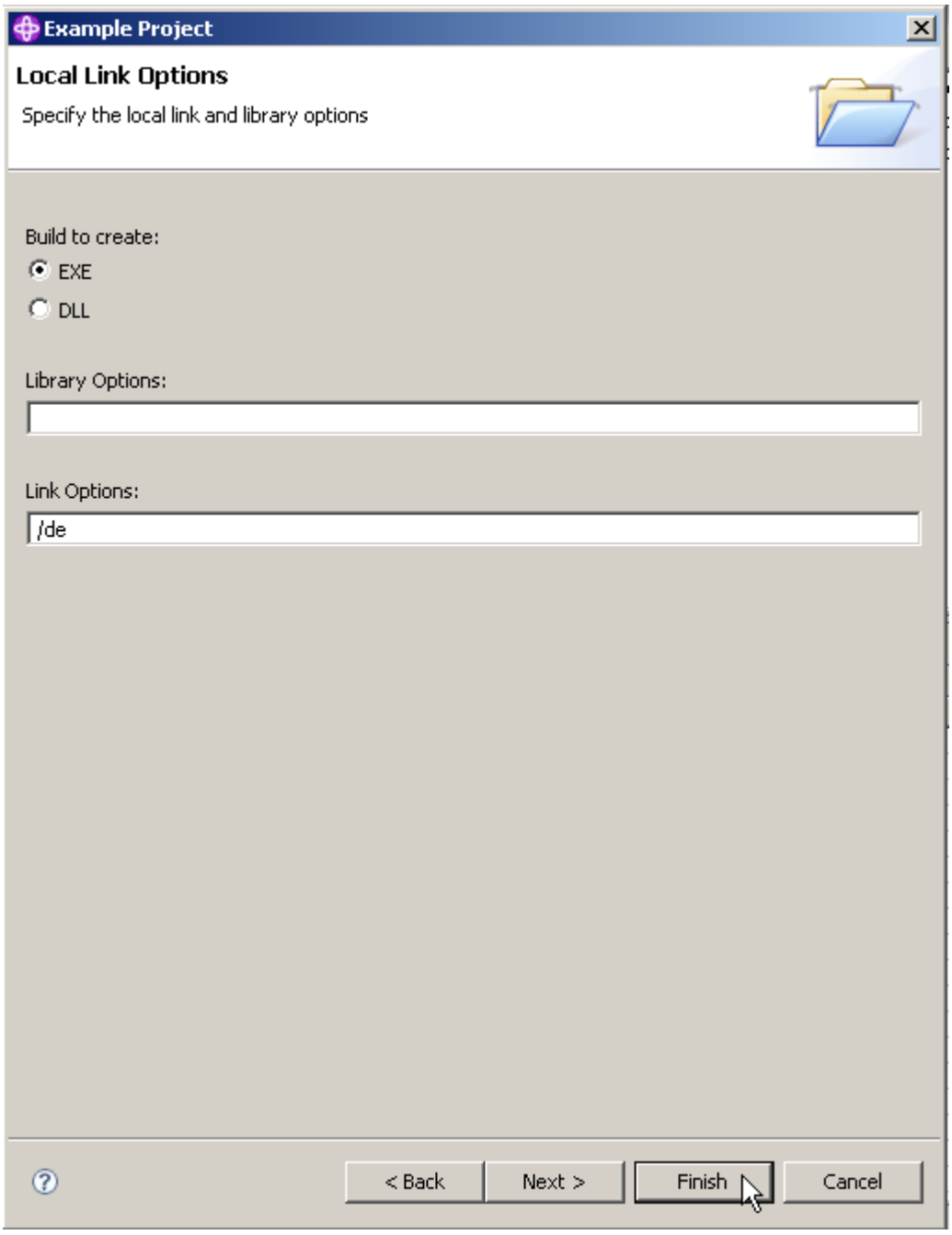


Abb. 2.5

Auf dem lokalen Link-Optionen-Panel sind die Link-Parameter festzulegen. Die Option für Build sollte EXE sein. Dies bedeutet, für ein COBOL-Programm aus diesem Projekt wird eine exe-Datei anstelle einer dll-Datei generiert.

Beachten Sie auch die Option /de. Dies bedeutet, Ihr Programm wird Debugging-Informationen enthalten. Klicken Sie auf Finish, um den Vorgang abzuschließen.

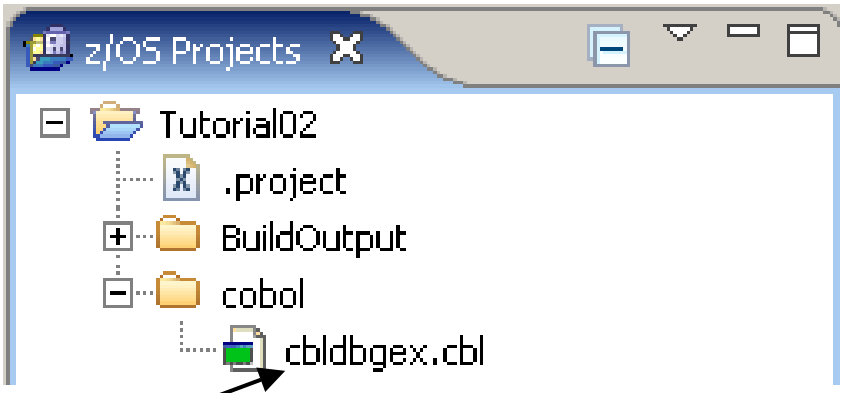


Abb. 2.6

Nach einigen Sekunden ist das Ergebnis links oben in der z/OS-Perspektive sichtbar Das cbldbgex.cbl Cobol source Program wurde geladen.

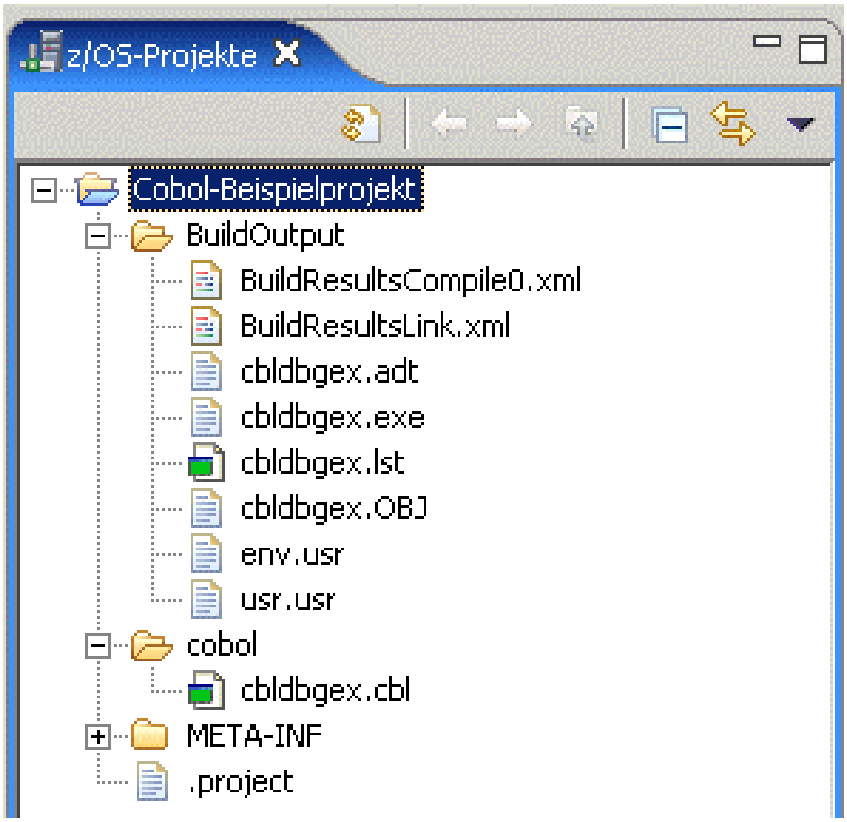


Abb. 2.7

Nach Erweiterung von "BuildOutput" und "cobol" sieht das Ergebnis wie folgt aus:

Beispiel-Cobol-Programm-Code: "cbldbgex.cbl"
Daraus erzeugte ausführbare EXE-Datei: "cbldbgex.exe"

3. Arbeiten mit einem localen COBOL Pprogramm

In diesem Kapitel erfahren Sie, wie Sie Quellcode bearbeiten und wie Sie eine Syntaxprüfung mit dem RDz COBOL-Editor durchführen.

3.1 Bearbeiten eines COBOL-Quellprogramms

Stellen Sie sicher, dass die z / OS Projects Perspektive geöffnet ist (Window → open Perspective → z/OS Projects).

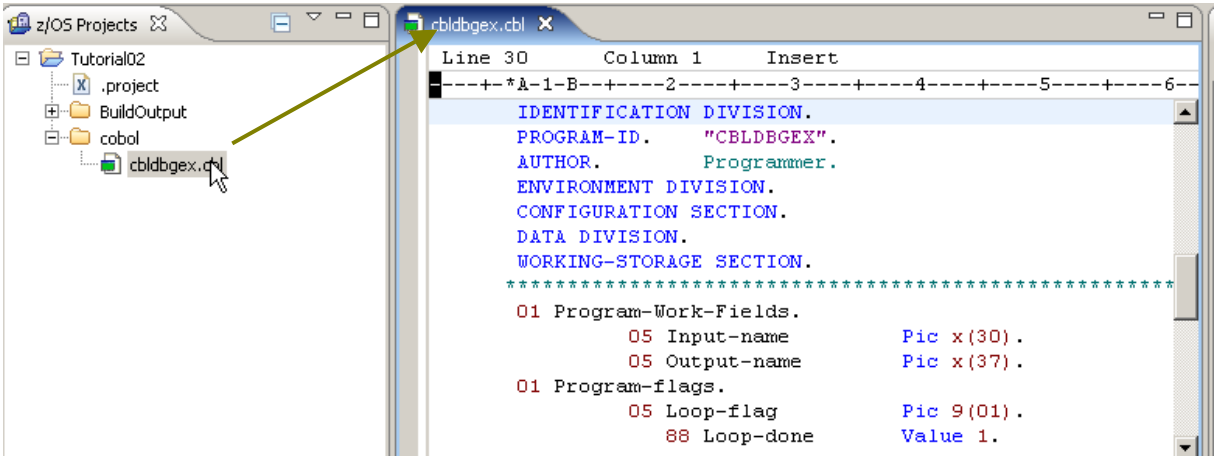


Abb. 3.1.1

Expandieren Sie den Cobol Folder. 2k auf cbldbgex.cbl. Sie sollten den Cobol Quell Code in dem mitleren Fenster sehen.

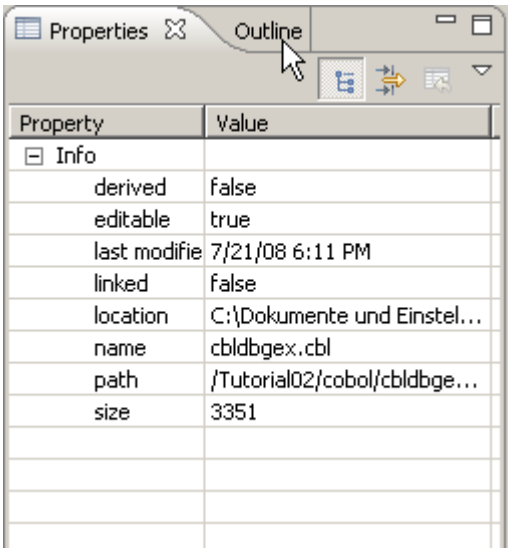


Abb. 3.1.2

Im linken unteren Fenster (unterer linker Teil des RDz Fensters) befindet sich der View (Registrierkarte) „Properties" (Eigenschaften), unter ihm verdeckt der View "Outline“ (Gliederung). Um letzteren View anzuzeigen, Klick auf die Registrierkarte „Outline“ (Gliederung).

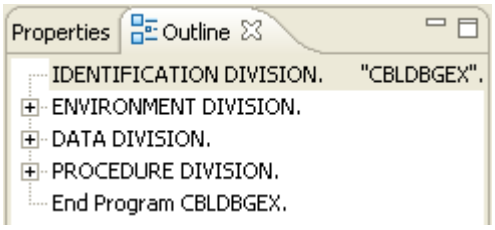


Abb. 3.1.3

Die Gliederungsansicht zeigt einen Überblick über die Strukturierung der Datei, die derzeit in dem Editor-Bereich geöffnet ist. Der Inhalt der Gliederungsansicht ist Editor-spezifisch.

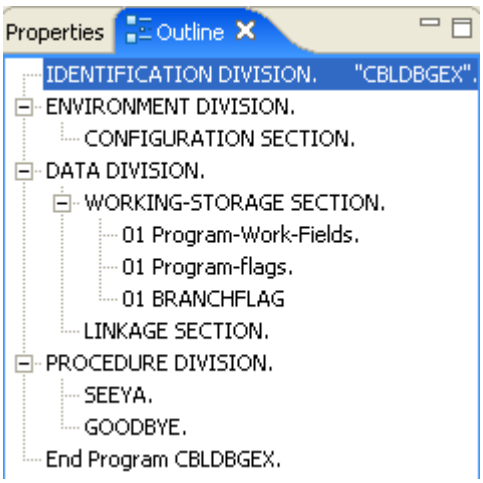


Abb. 3.1.4

Expandiere die + Zeichen und betrachte die COBOL Struktur.

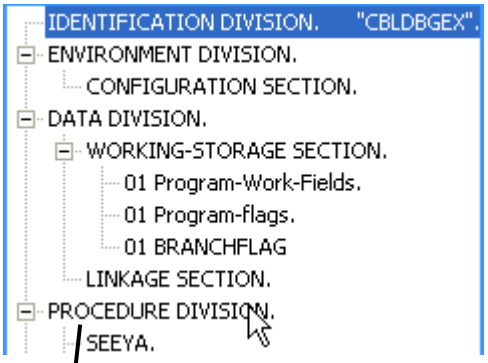


Abb. 3.1.5

Klick auf " PROCEDURE DIVISION" in der Gliederung bewirkt, dass genau dieser Teil des Cobol-Programm-Codes in der zentralen View angezeigt wird.

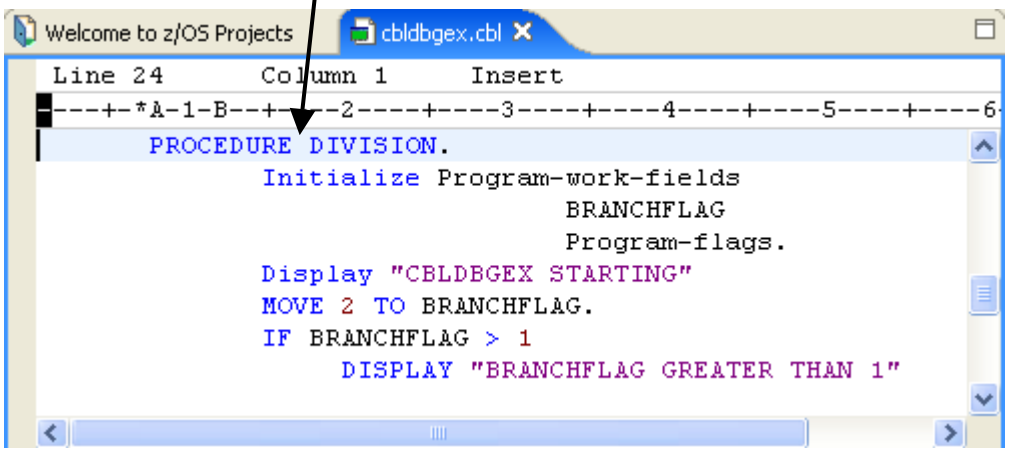


Abb. 3.1.6

Der Editor zeigt die PROCEDURE DIVISION des COBOL Programms. Klick auf "ENVIRONMENT DIVISION" in der Gliederung würde bewirken, dass genau dieser Teil des Cobol-Programm-Codes in dem zentralen View angezeigt wird.

3.2 LPEX und ISPF Editor

Der in RDz integrierte Editor heißt LPEX-Editor. LPEX (Live Parsing Extensible Editor) ist ein erweiterbarer und leistungsfähiger Texteditor. Zu den Funktionen zählen Syntaxhervorhebung und Programmierbarkeit.

Die Tastenkombination Strg+F ermöglicht nun das Suchen (und Ersetzen) von Strings im Cobol-Programm-Code.

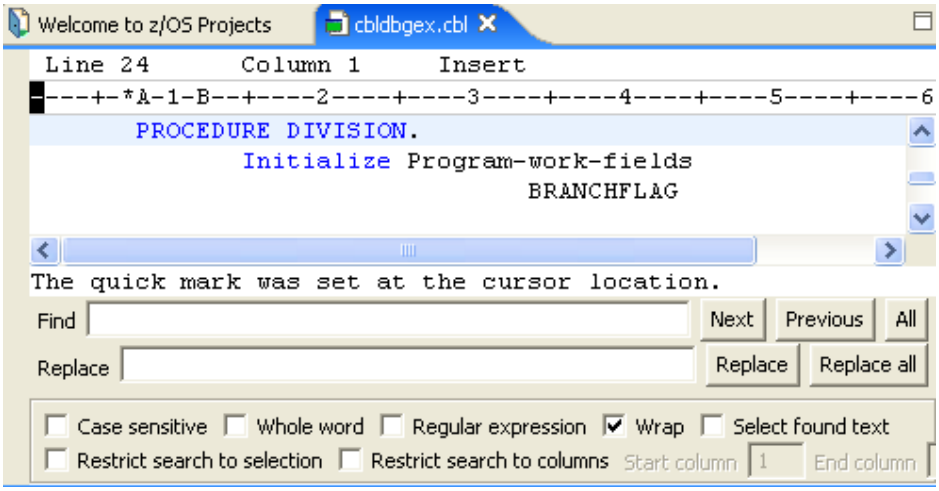


Abb. 3.2.1

Dies aktiviert die Search Facility. Soll zum Beispiel nach "Branchflag" gesucht werden:
In die Zeile "Find (Suchen)" den String "BRANCHFLAG" eingeben. (Ein Abschließen mit der Eingabetaste ist nicht erforderlich).

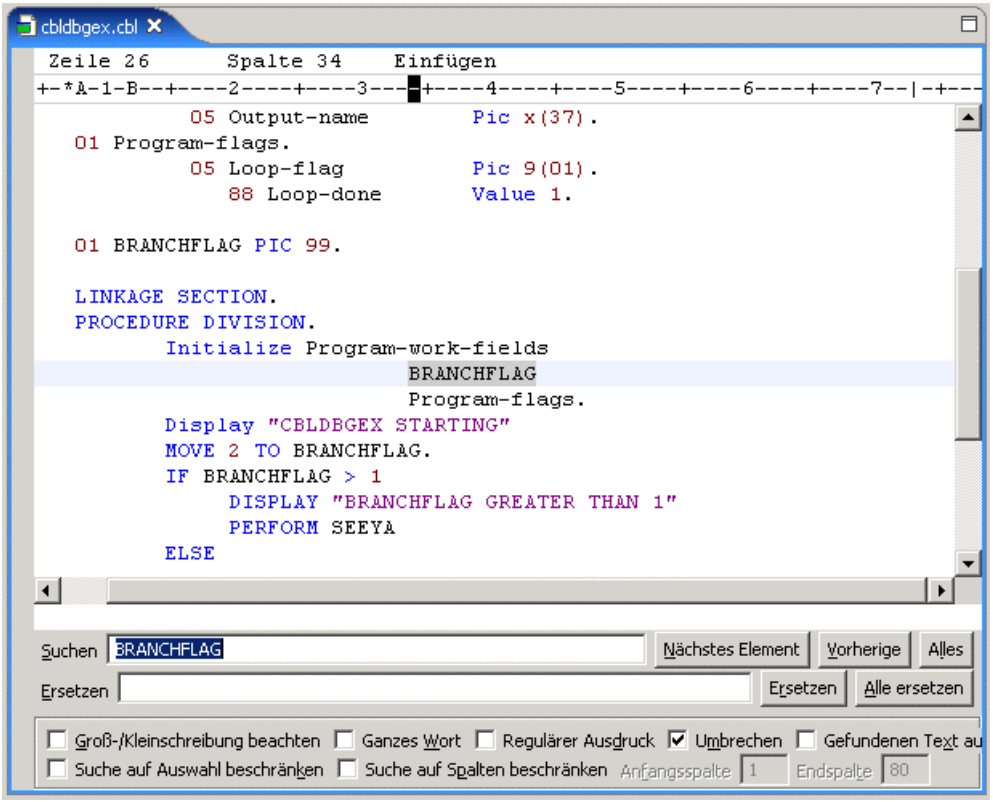


Abb. 3.2.2

Es ist intuitiv sehr einfach, diese Suchen- und Ersetzen-Funktionalität zu verwenden.

Probieren Sie doch mal:

- **Nach dem nächsten String "BRANCHFLAG" suchen,**
- **nach dem vorigen String "BRANCHFLAG" suchen,**
- **Alle vorhandenen "BRANCHFLAG" z.B. durch "BRANCHF" ersetzen, etc.**

Das vom LPEX Editor benutzte Default Profil ist lpex. Es definiert die Tasten Belegung. Der LPEX Editor kann die Tastenbelegung von anderen Editoren emulieren, u.a. dem

- **ISPF-Editor,**
- **Xedit-Editor (Standard CMS Editor für das z/VM Betriebssystem),**
- **Emacs-Editor,**
- **vi-Editor (von POSIX standardisierter Texteditor für Unix und Linux),**
- **etc.**

Standardmäßig ist das lpex Profil eingestellt. Ist man die Funktionsweise des ISPF-Editors gewöhnt, so lässt sich beispielsweise die ISPF-Editor-Emulation wie folgt aktivieren:

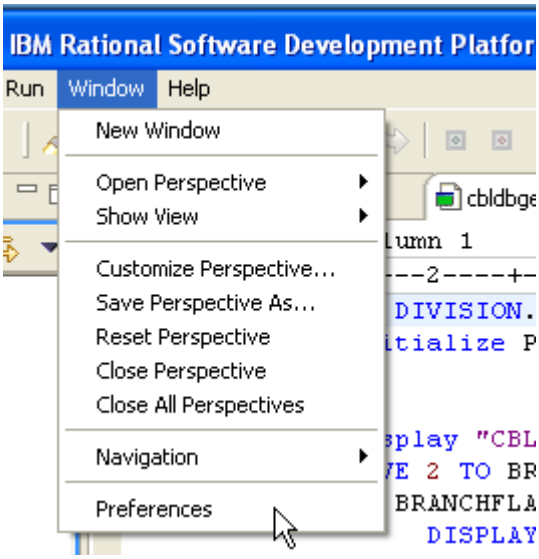


Abb. 3.2.3

Open the Preferences Dialog. 1k auf Window → Preferences.

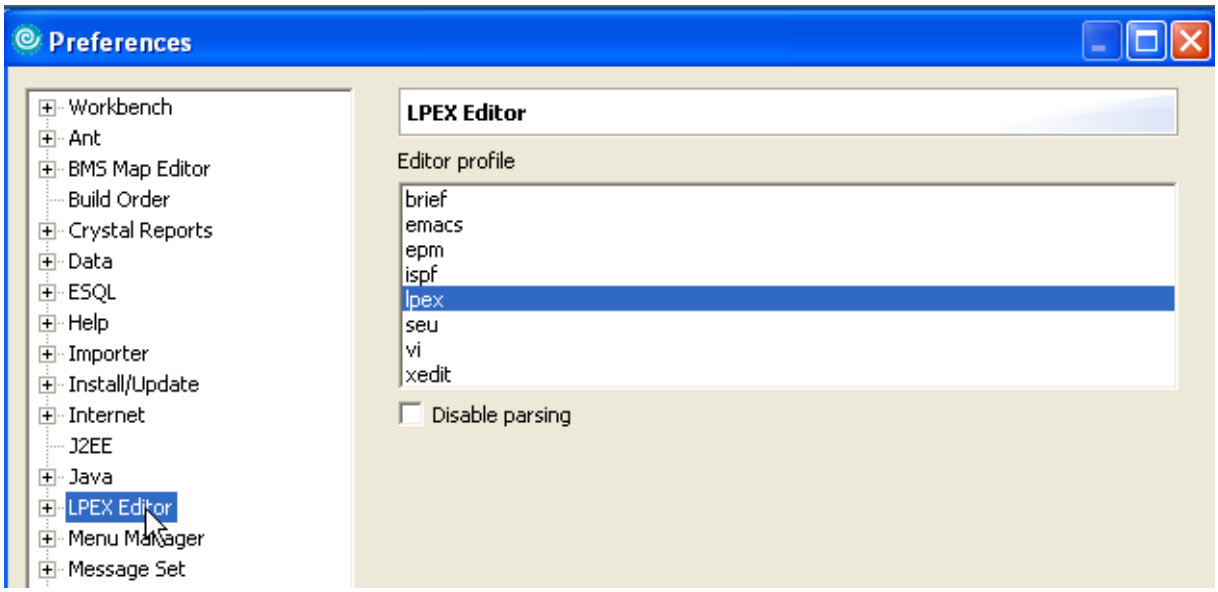


Abb. 3.2.4

1k auf LPEX Editor, und

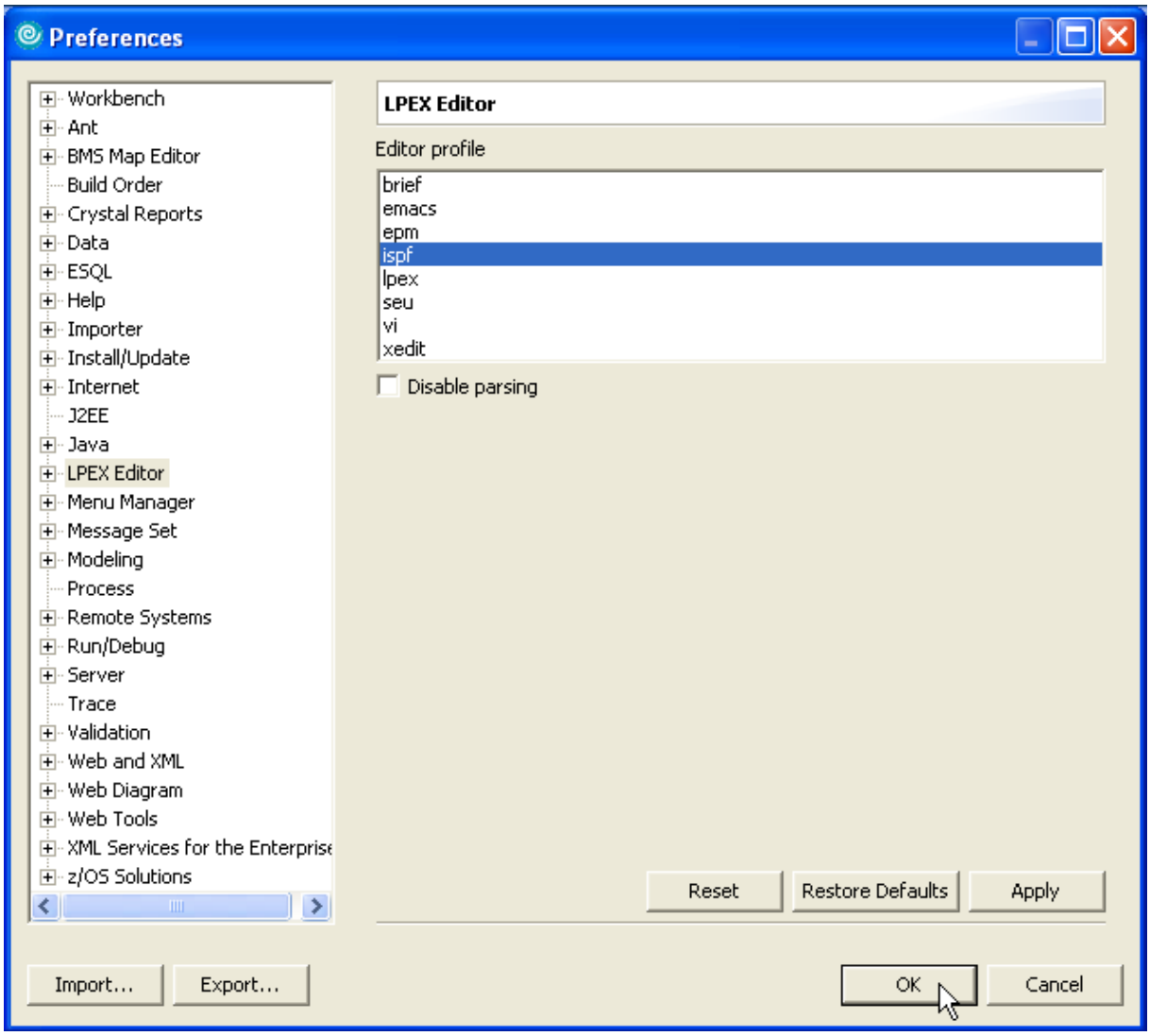


Abb. 3.2.5

1k on ispf, 1k on OK. Dies schließt den Preferences Dialog.

Selbst-Test

- Ist es unbedingt erforderlich, das Editor Profile von lpex auf ispf zu ändern, oder könnte man Cobol Programme auch mit dem lpex Profile erstellen ?

3.3 Erstellen von Tabs für den Editor

Da wir mit COBOL arbeiten, ist es eine gute Idee, einige TABS für eine einfache Navigation im Editor zu erstellen. Öffnen Sie den Dialog Preferences. 1k auf Window → Preferences.

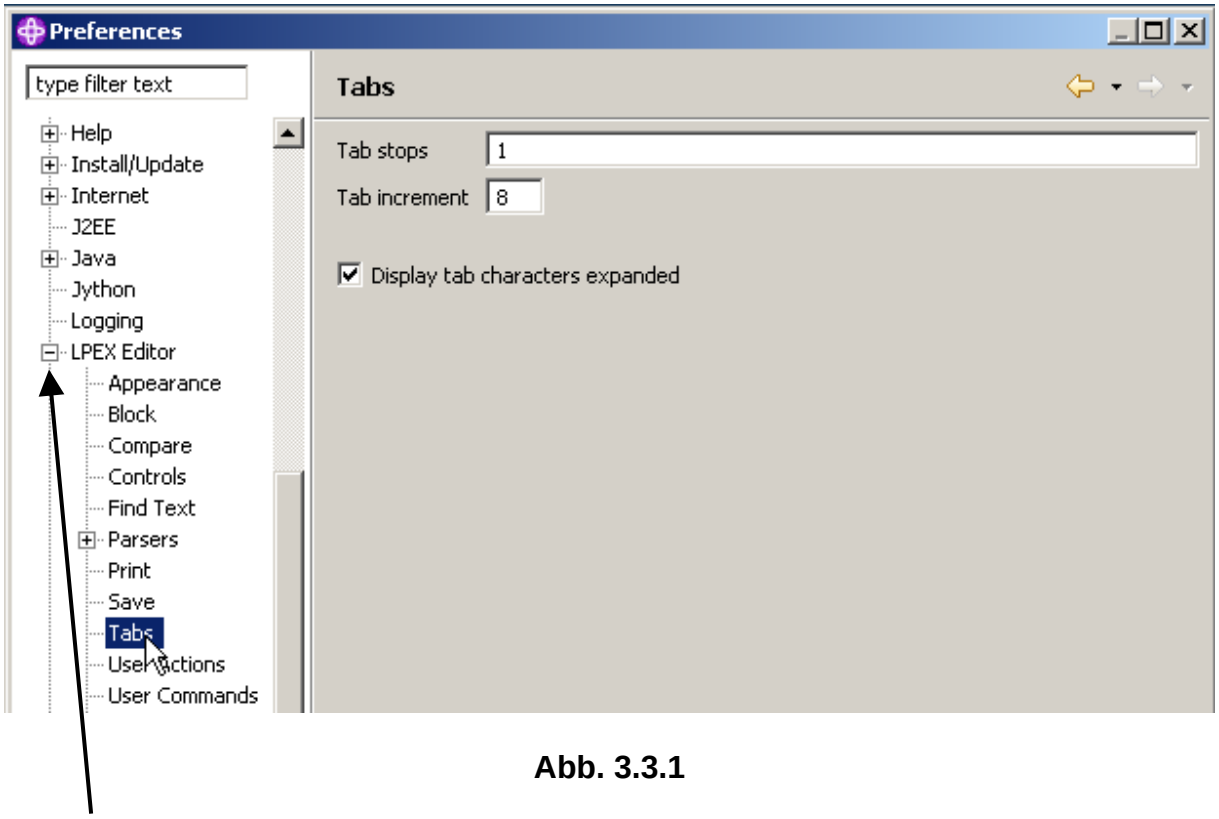


Abb. 3.3.1

Expand LPEX Editor, click on Tabs

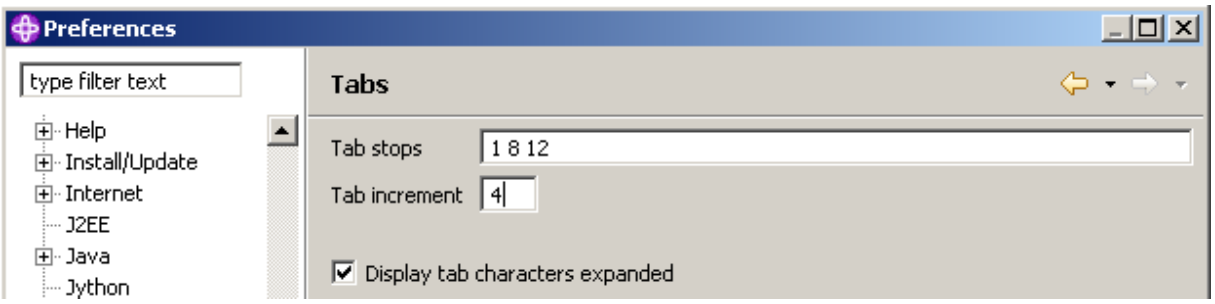


Abb. 3.3.2

Click on Tab stops und enter 1 8 12 als Tab Stops. Ändere das Tab Iincrement zu 4 (statt 8). Jedes Mal wenn die Tab Taste gedrückt wird, wird der Cursor auf Positions 1, 8 und 12 bewegt. Das Increment 4 bewirkt weiterhin Stops auf 16, 20, 24 usw. Click OK.

3.4 Using Prefix Commands

Sie sind jetzt in der Lage, mit Präfixbefehlen im Präfix Bereich zu arbeiten, wie d zu löschen, m zu bewegen, etc. Sie können auch Tastenkombinationen für Befehle, zB (Strg + L) für Locate, (Strg + F) für Find, etc. eingeben. Siehe Tutorial 1c für Einzelheiten.

Beispiel, um nach Zeile 25 zu bewegen, hit Ctrl + L,

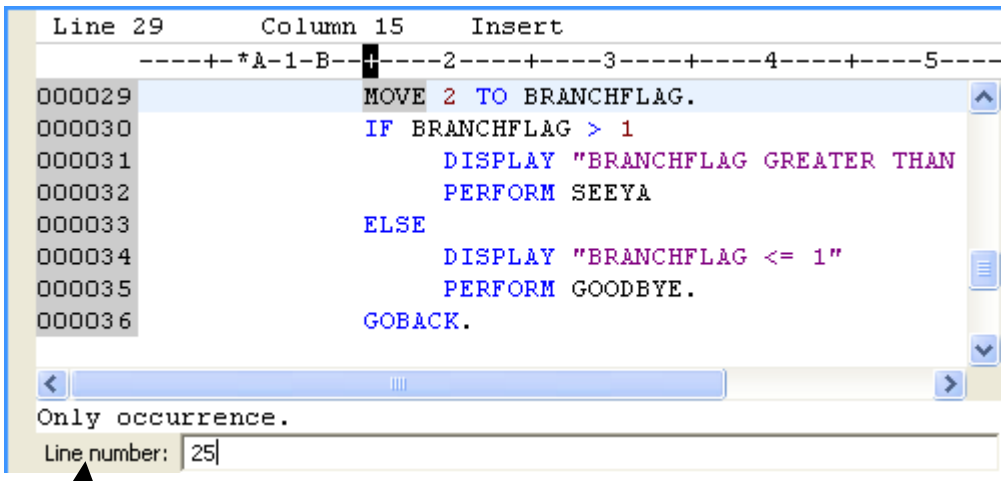


Abb. 3.4.1

25 in in das Zeilennummerfeld eingeben, Enter.

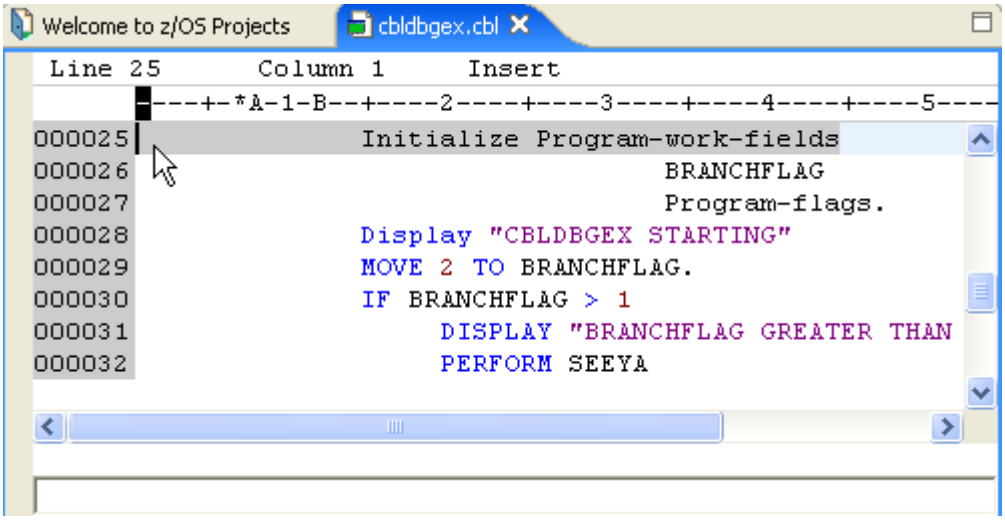


Abb. 3.4.2

Dies ist das Ergebnis

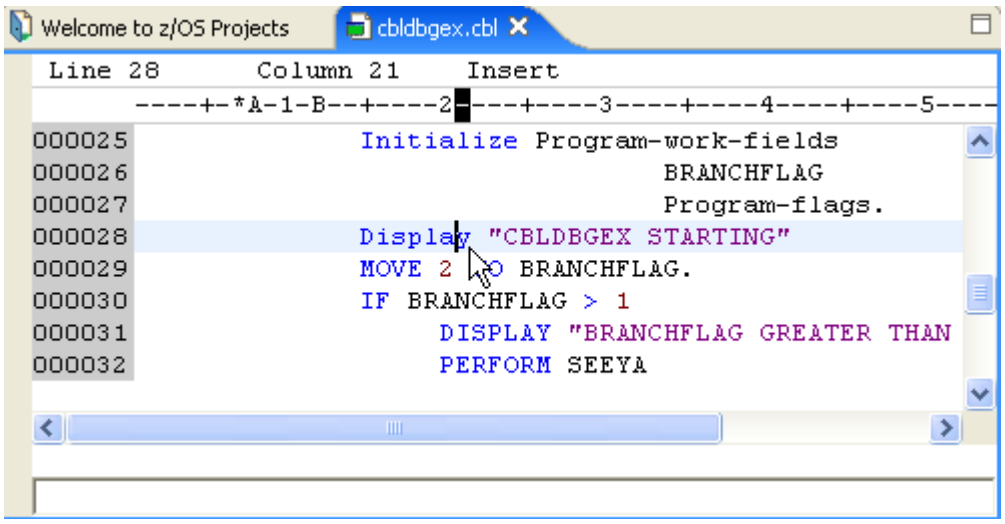


Abb. 3.4.3

Nach Zeile 28 bewegen, den Tab Key benutzen um den Cursor auf das Wort “Display” zu setzen. (Anmerkung: Shift und Tab bewegt den Cursor zurück).

Selbst-Test

- Können Prefix Commands auch mit dem lpex Profile benutzt werden ?

Tipp: Wenn die Tabs beim ersten Mal nicht arbeiten, wechseln sie von ISPF nach LPEX zurück, und dann wieder nach ISPF.

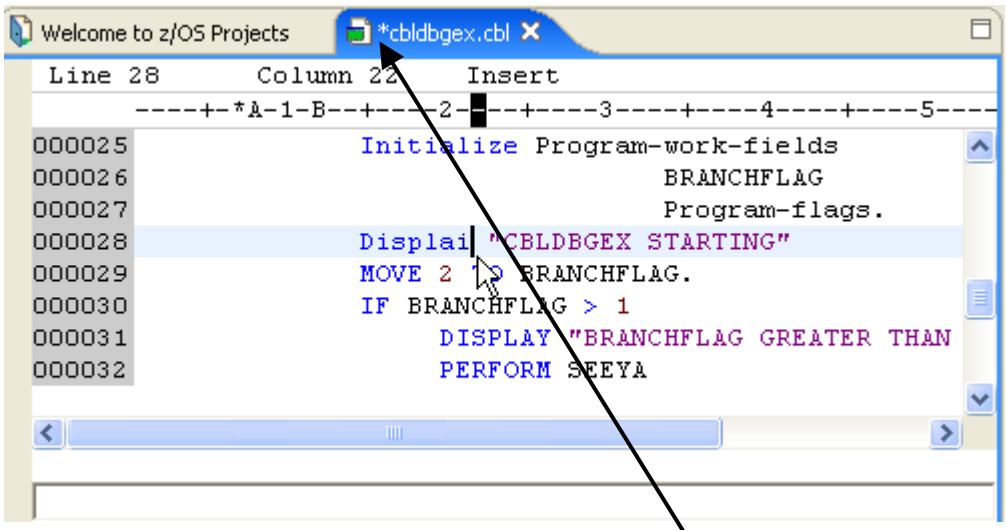


Abb. 3.4.4

Ändern Sie das Wort Display in Displai. Beachten Sie: da wir über einen Smart Editor verfügen, und Displai kein gültige COBOL Statement ist, wechselt die Farbe von blau auf schwarz .

Achten Sie auf den Stern * auf der linken Seite des Dateinamens 

Er zeigt an, dass die Datei geändert wurde. Das Sternchen verschwindet, sobald die Datei gespeichert wurde.

Drücken Sie STRG + S, um die Änderung zu speichern. Zu diesem Zeitpunkt wird kein Fehler festgestellt, da das Programm compiliert werden muss, um Fehler zu finden.

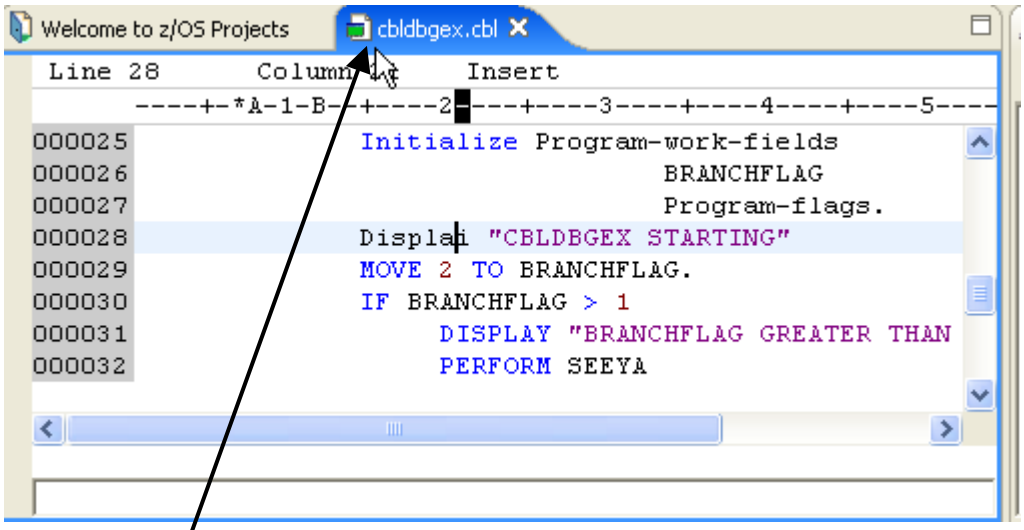


Abb. 3.4.5

Beachten Sie, dass der * links vom Dateinamen verschwunden ist.

Machen Sie eine weitere Änderung. Gehen Sie nach Zeile 33 , geben Sie ein “d” in linken grauen Area (der „Line Command Area“, siehe Tutorial 1c) ein.

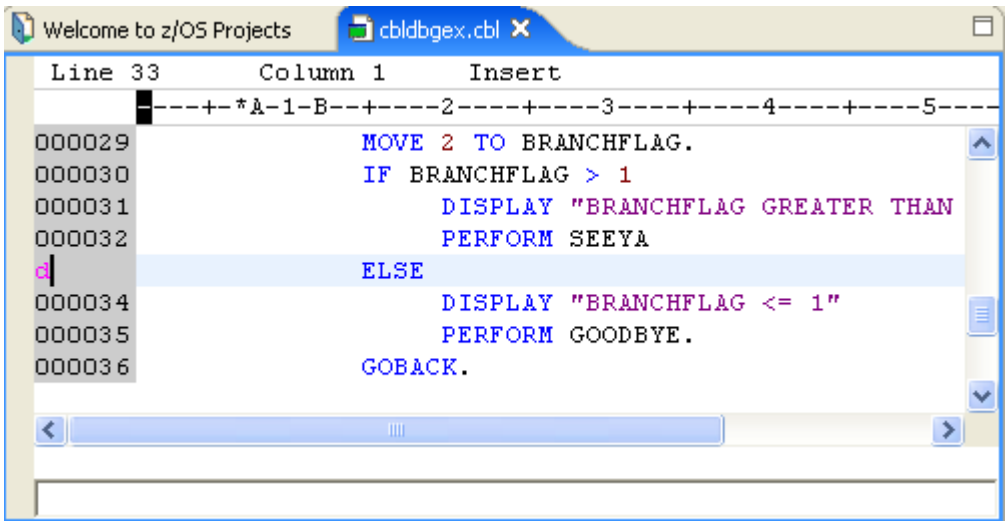


Abb. 3.4.6

Drücken Sie Enter, um die Zeile zu löschen.

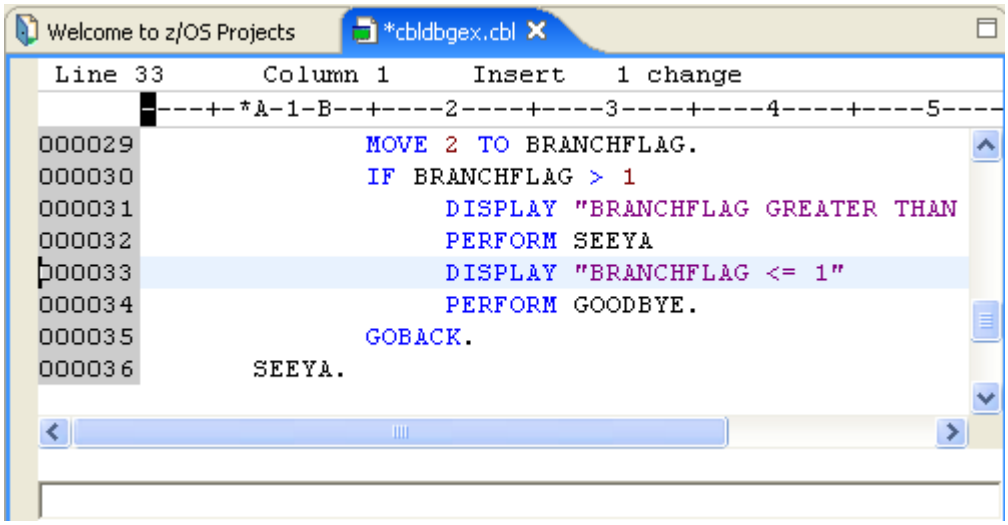


Abb. 3.4.7

Drücken Sie CTRL + S um es zu retten (to save it).

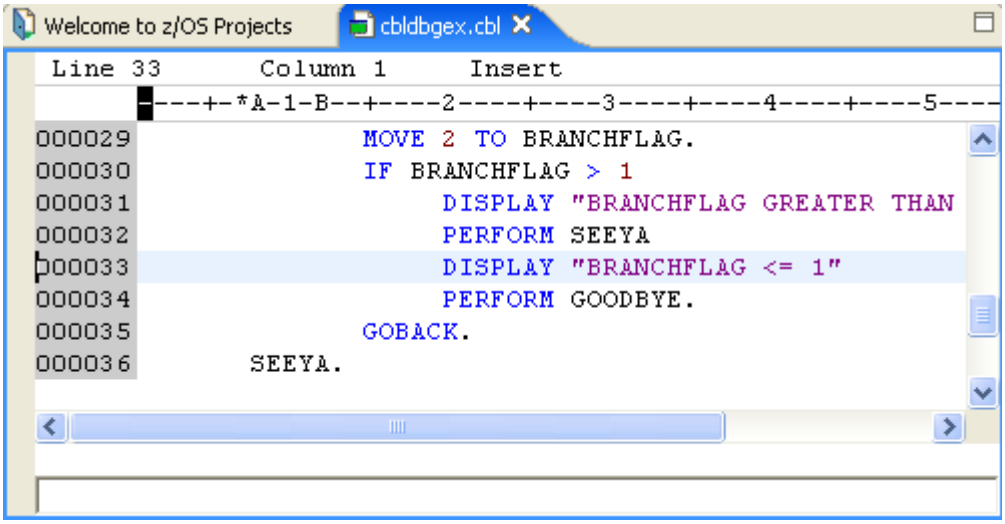


Abb. 3.4.8

Dies ist das Ergebnis.

Sie können jederzeit das Editor Fenster vergrößern oder verkleinern. Gelegentlich werden sie mehr Raum brauchen um das ganze Programm zu sehen.

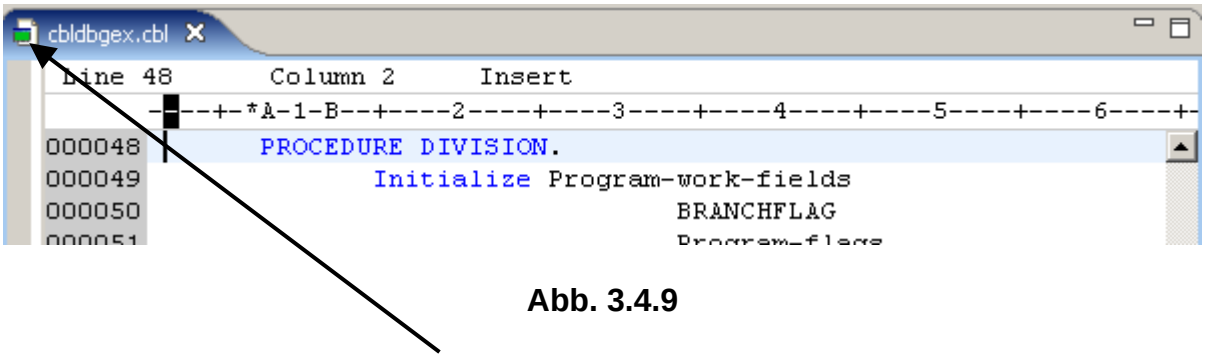


Abb. 3.4.9

Am Einfachsten geht dies mit einem 2k (Double Click) an dieser Stelle. Dies maximiert die Größe des Fensters. Ein weiteres 2k stellt die ursprüngliche Fenstergröße wieder her.

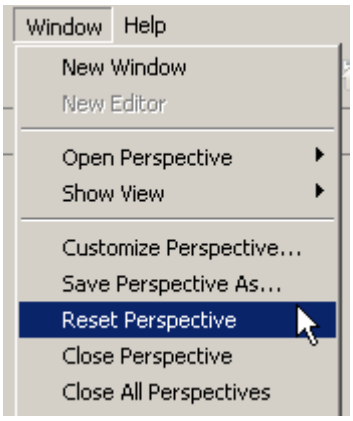


Abb. 3.4.10

Wenn Sie die Größe des Editor Fensters geändert haben, irgendwelche Views geschlossen haben, usw. und Sie wollen zum Originalzustand der Perspective zurückkehren, selektieren Sie Window → Reset Perspective.

3.5 Benutzung lokaler Datei Versionen

Die lokale History einer Datei wird automatisch zwischengespeichert, wenn Sie diese erstellen oder ändern. Jedes Mal, wenn Sie eine Datei bearbeiten und speichern, wird eine Kopie der alten Version zwischengespeichert. Damit können Sie jederzeit Ihren aktuellen Datei Zustand mit einen früheren Zustand vergleichen, oder die Datei durch einem früheren Zustand ersetzen. Jeder Status in der lokalen History wird durch das Datum und die Uhrzeit der Datei-Speicherung identifiziert. Um eine aktuelle Version durch eine vorherige zu ersetzen (z. B.die Zeile, die Sie vorher gelöscht haben, wiederherzustellen) gehen Sie so vor:

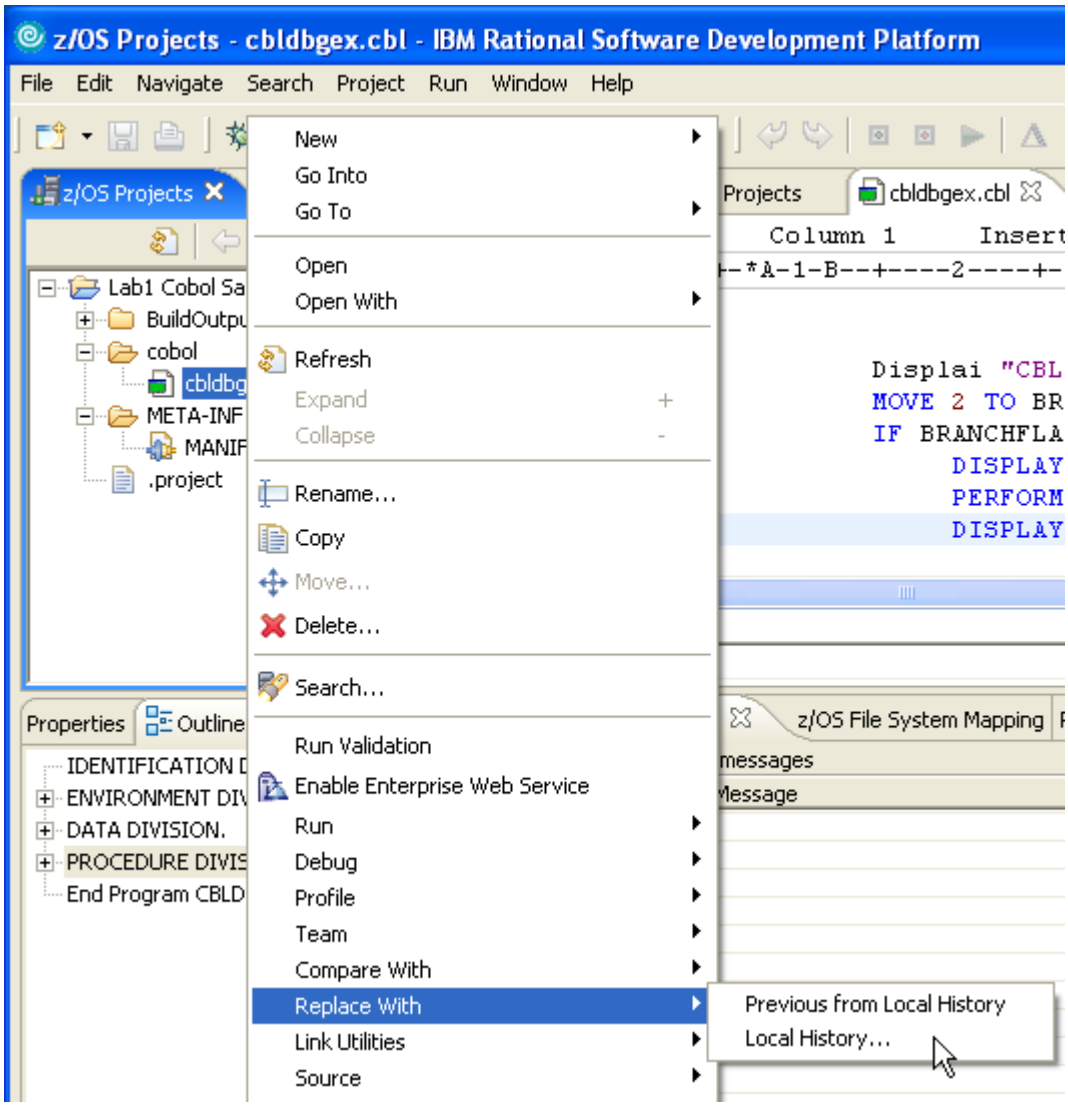


Abb. 3.5.1

Im z/OS Projects view, 1kr auf cbldbgex.cbl. Von dem pop-up Menu, selektiere “Replace with! → Local History

1k

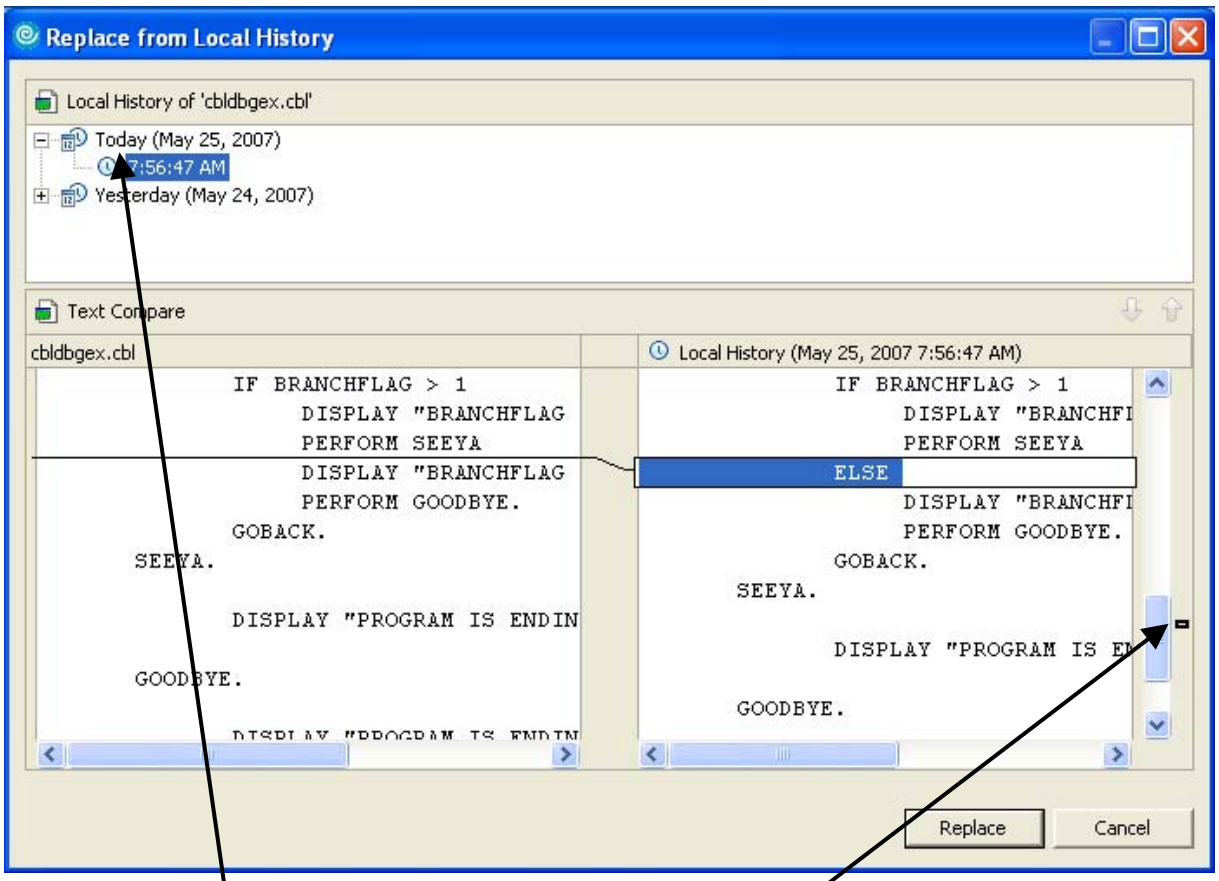


Abb. 3.5.2

In diesem Fall sind Datum und Uhrzeit unterschiedlich.

Das „Replace from Local History" Fenster wird geöffnet. Auf der linken Seite sehen Sie die aktuelle Version unseres Programms. Auf der rechten Seite befindet sich die vorherige Version.

Beachten Sie das kleine Zeichen auf der rechten Seite. Dies ist wichtig, wenn wir vielfache Änderungen vorgenommen hatten.

Nur auf diese Marke klicken, um zur nächsten Änderung zu gelangen. Beachten Sie, dass in diesem Beispiel nichts geschieht, da Sie nur eine Veränderung vorgenommen hatten.

Von der Local History, wählen Sie die Zeit, die Sie ersetzen möchten. In diesem Beispiel, wählen Sie den jüngsten Eintrag.

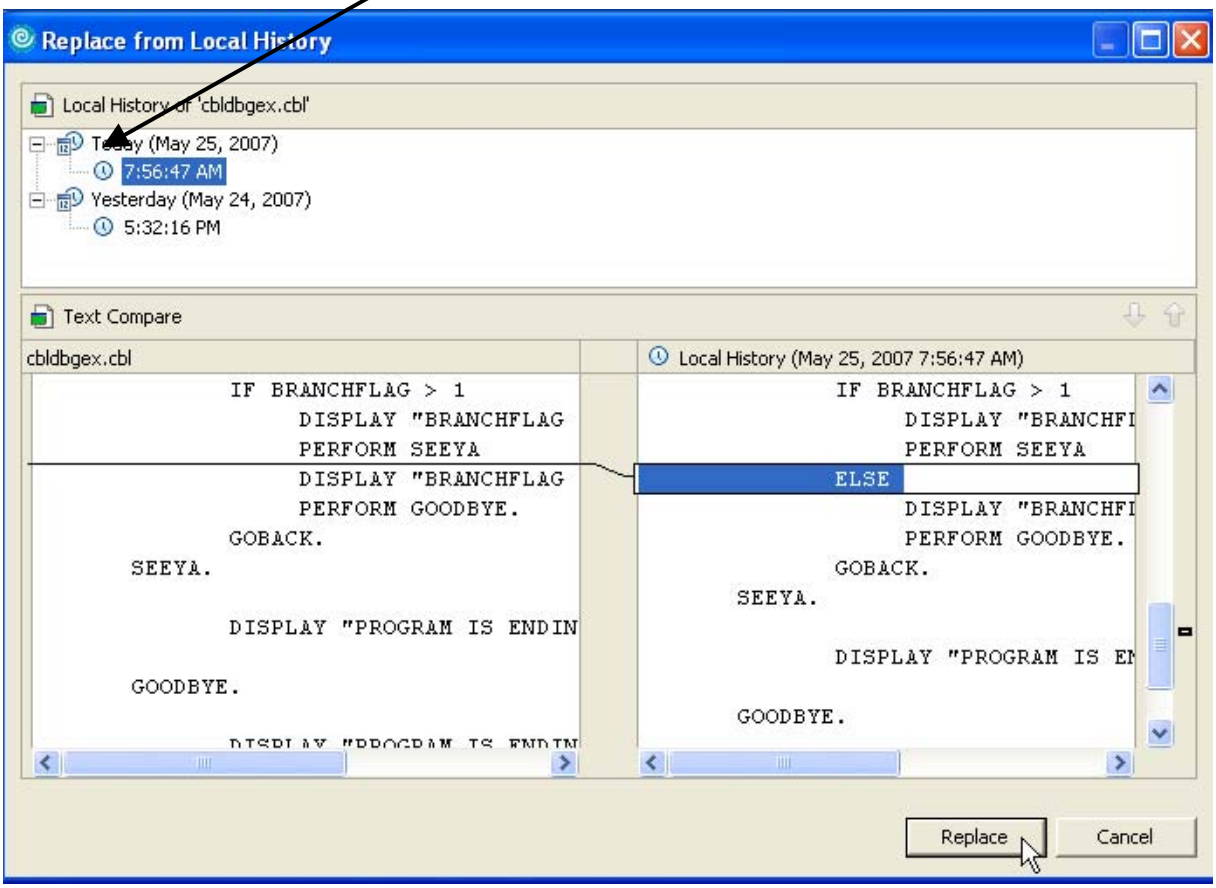


Abb. 3.5.3

Dann click Replace.

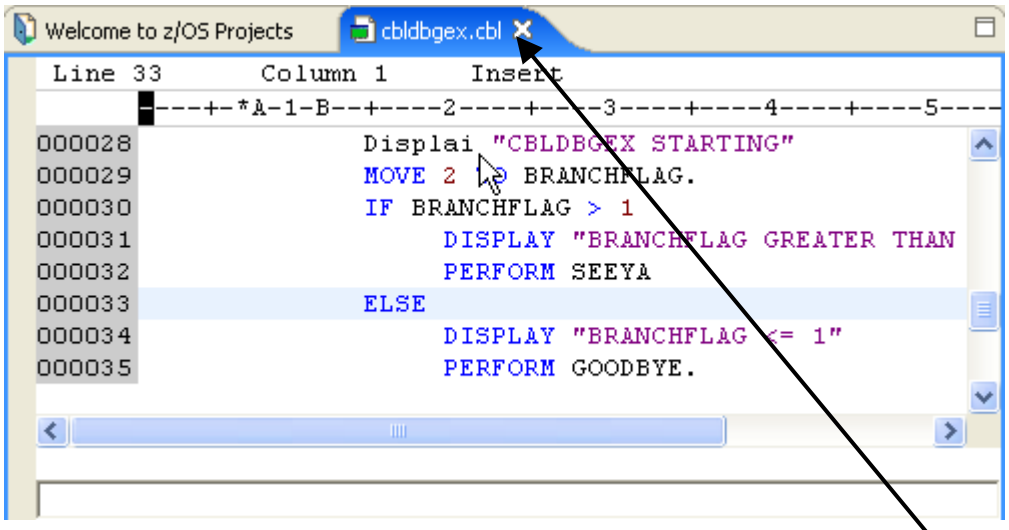
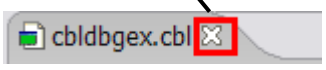


Abb. 3.5.4

Zeile 33 ist wiederhergestellt, die Änderung “Displai“ ist noch vorhanden.

Den Editor schließen durch ein Klick auf das X neben dem File Namen.



3.6 Überprüfung der COBOL-Quelldatei Syntax

In dem z/OS-Projekte View, highlight cbldbgex.cbl, 1kr (klicken Sie die rechte Maustaste), und wählen Sie Local Syntax Check Check aus dem Pop-Up-Menü:

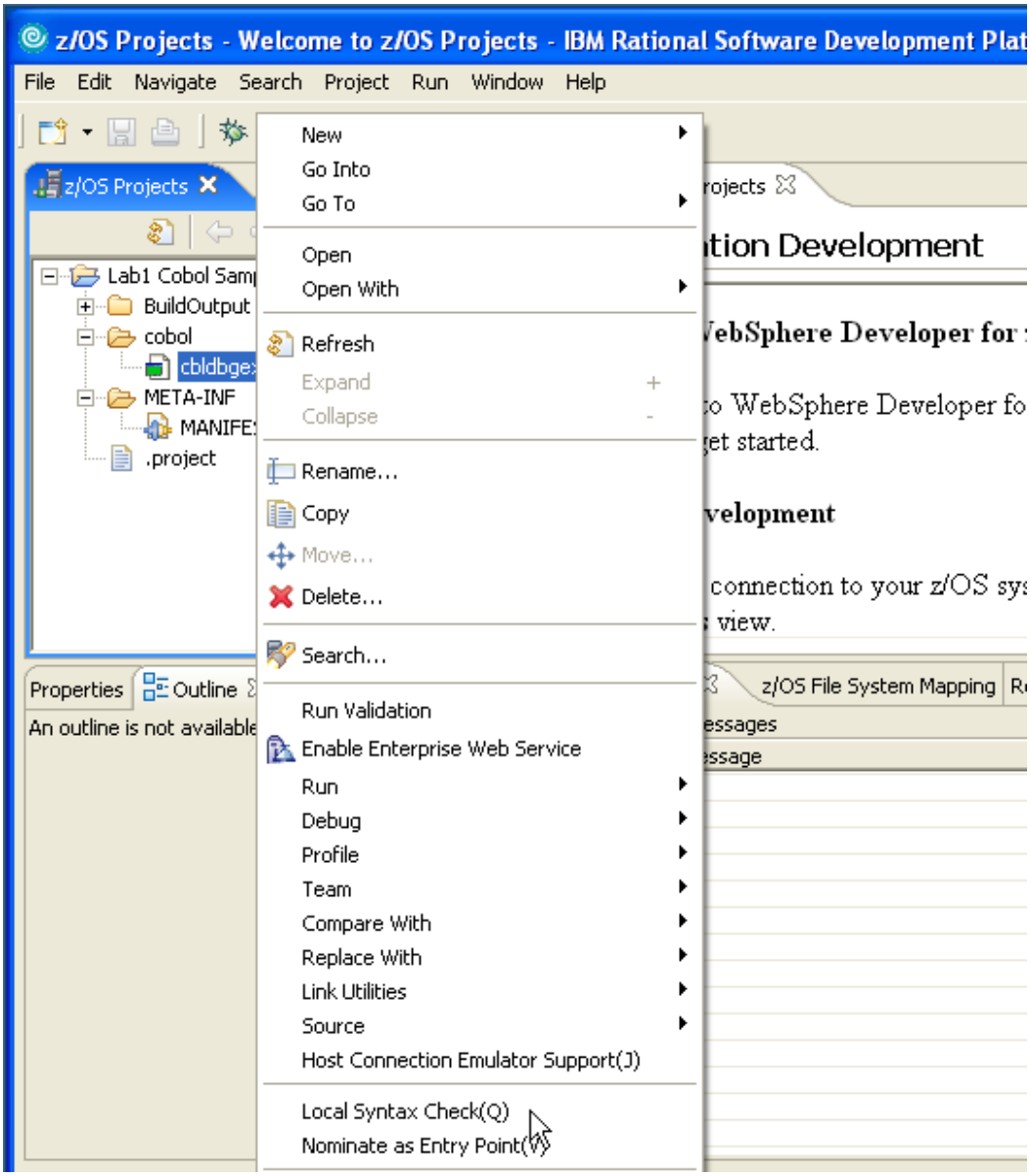


Abb. 3.6.1

Click auf Local Syntax Check.

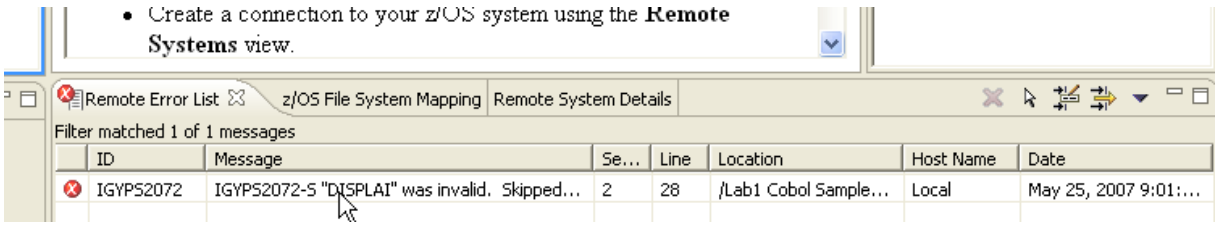


Abb. 3.6.2

Schauen Sie nach unten. Das Tab “Remote Error List” zeigt die Compile Fehler an.

Doppelklicken Sie auf einen Fehler. Das bringt Sie zu dem Editor an die Stelle, wo das fehlerhafte Statement entdeckt wurde.

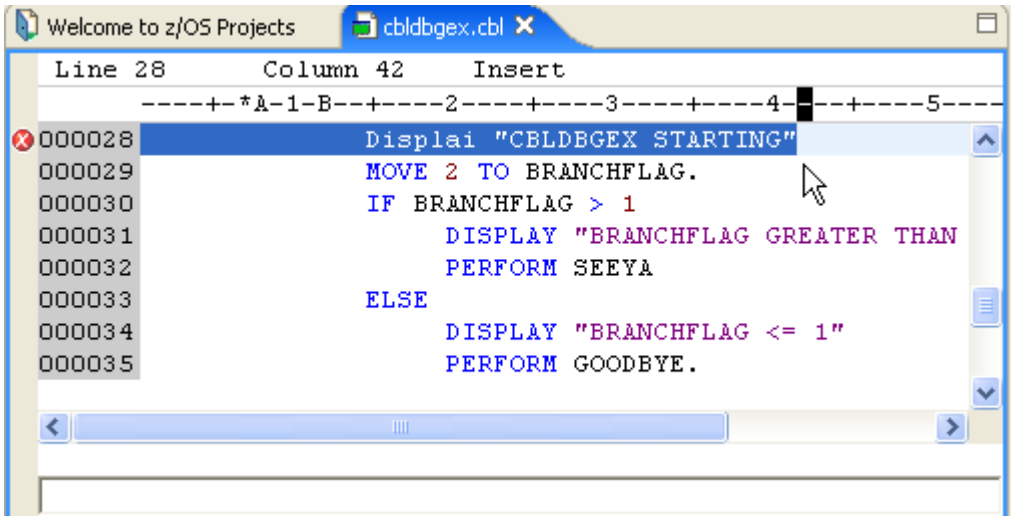


Abb. 3.6.3

Die fehlerhafte Zeile ist highlighted.

Ändern Sie Displai in Display.

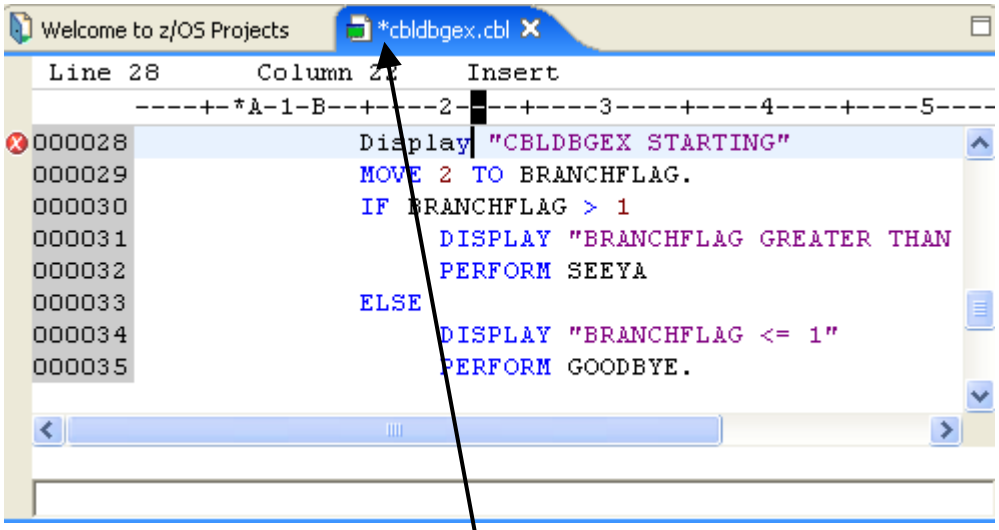


Abb. 3.6.4

Save Änderungen mit (Ctrl + S), Das Sternchen * verschwindet.

Führen Sie einen weiteren Lokalen Syntax Check durch. Sie sollten keine Fehler in der Remote Error List nach dem Ausführen des Local Syntax Check mehr sehen.

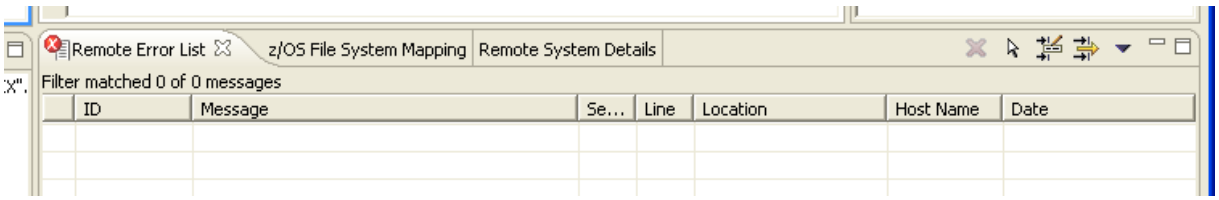


Abb. 3.6.5

Die Compilierung ist jetzt sauber.

3.7 Fenster vergrößern/verkleinern

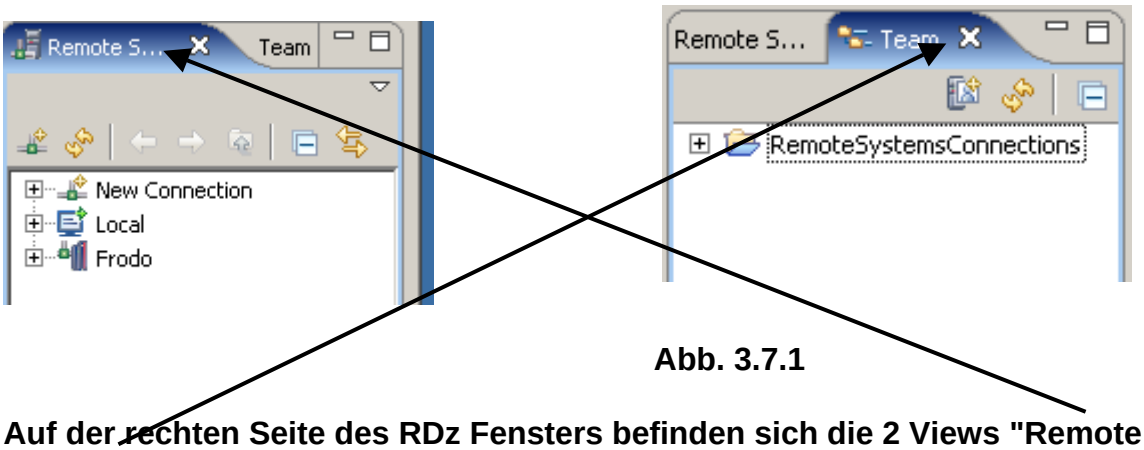


Abb. 3.7.1

Auf der rechten Seite des RDz Fensters befinden sich die 2 Views "Remote Systems" und "Team" jeweils überlagert. Sie können jeden der beiden Views durch einen Klick auf die entsprechende Registrierkarte (Tab) öffnen.

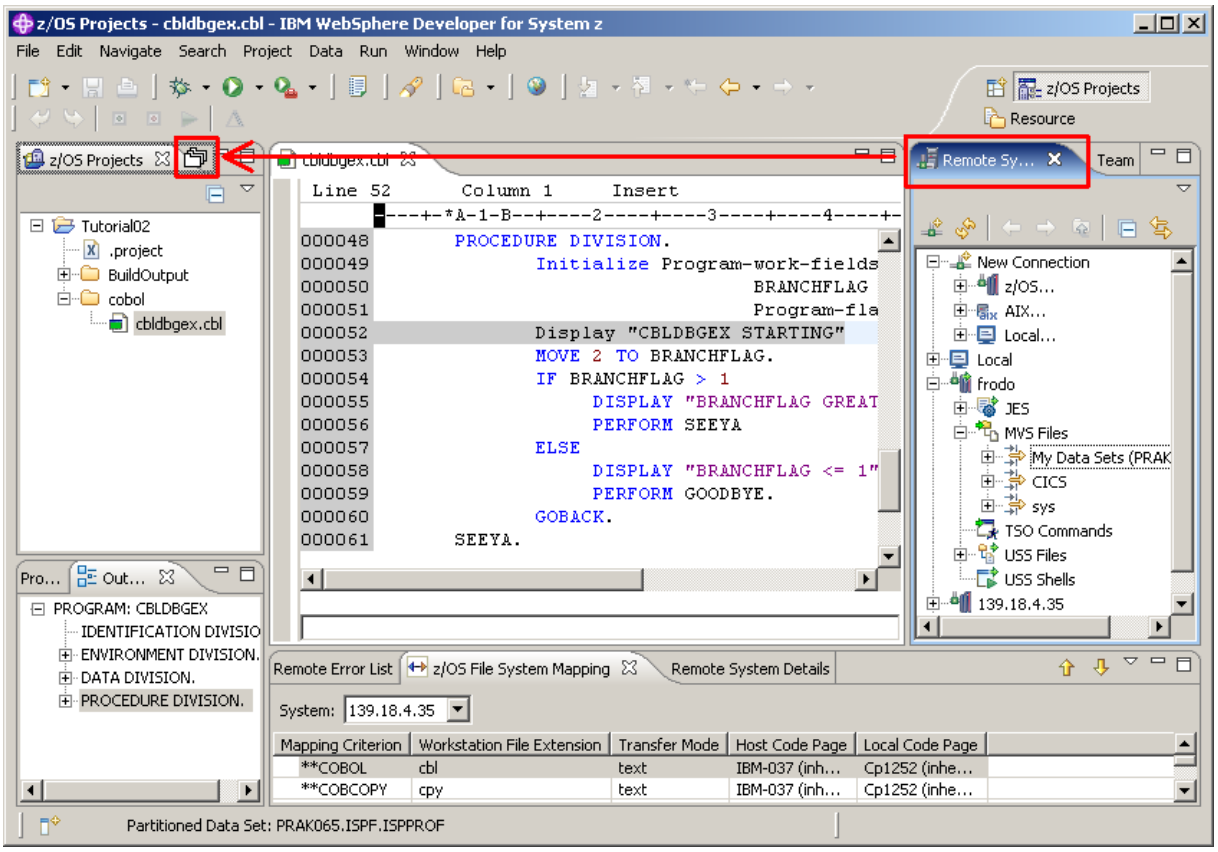


Abb. 3.7.2

In diesem Tutorial werden Sie die Ansichten Remote Systems und Team nicht benutzen. Sie können beide Views auf der linken Seite des RDz Fensters bewegen, um mehr Platz für die COBOL-Bearbeitung zu gewinnen. Hierzu Drag & Drop beide Views auf der Oberseite des z/OS Projects View, wie unten dargestellt.

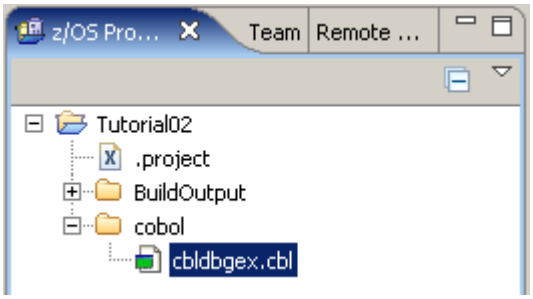


Abb. 3.7.3

Beachten Sie, dass der Maus Zeiger sich zu ändert, ehe Sie den Drop durchführen. Sie müssen dies für jeden der beiden Views getrennt tun.

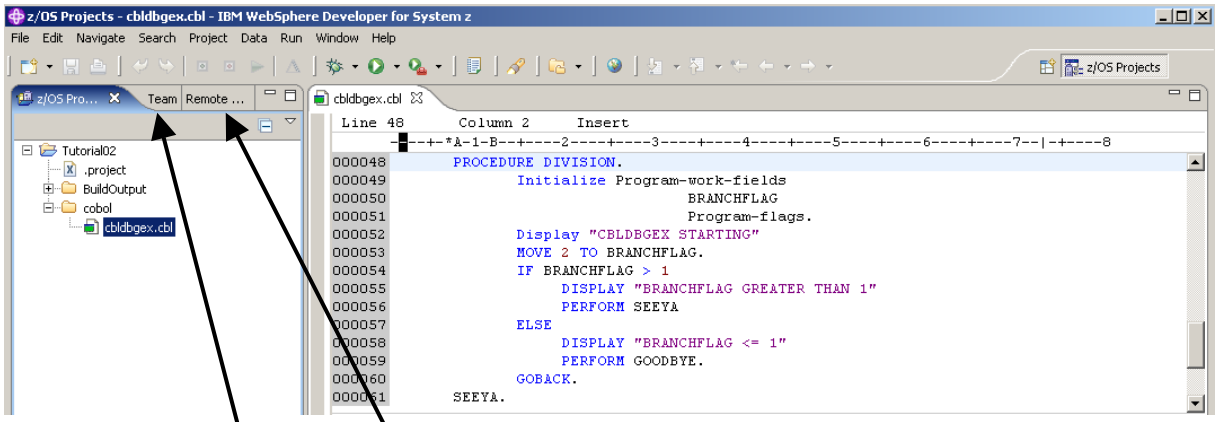


Abb. 3.7.4

Dies sind die Tabs für Team und für Remote System. Das Editor Fenster hat jetzt mehr Platz.

Je nach Bildschirmauflösung können Sie jetzt 1k auf und dann den z/OS Projects tab selektieren.

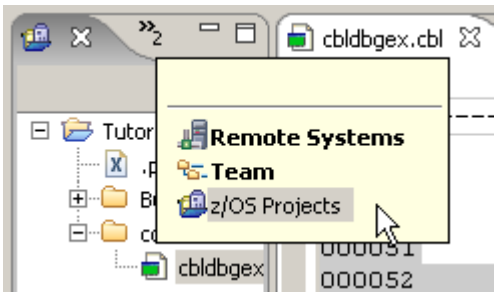


Abb. 3.7.5

Klick auf z/OS Projects. Ansonsten 1k auf das z/OS Projects Tab.

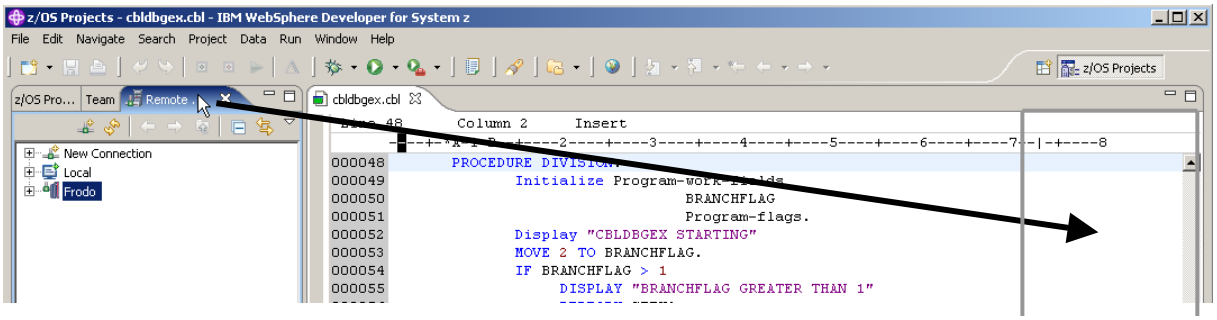


Abb. 3.7.6

Um das Fenster wiederherzustellen, einfach per Drag & Drop die Registrierkarte an den ungefähren Ort, wo Sie wollen, dass das Fenster wiedergegeben wird. Beachten Sie, ein Rechteck erscheint, um die Position anzugeben.

3.8 Das Cobol Quellprogramm ändern

Einer der attraktiven Eigenschaften des COBOL-Editor ist die Fähigkeit, Fehler vor der Kompilierung zu finden und Entwicklern zu helfen, Code zu schreiben. Lassen Sie uns diese Funktionen hier näher ansehen. Angenommen, wir wollen etwas Logik zu unserem Programm hinzufügen.

Ein Beispielist, wenn wir zu dem COBOL Quell-Programm Statements wie: DISPLAY, ACCEPT, MOVE und IF hinzufügen möchten. Hier sind einige Beispiele:

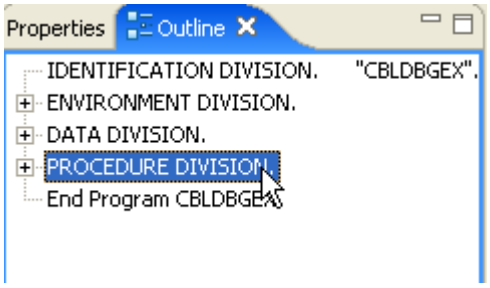


Abb. 3.8.1

Um COBOL Anweisungen unmittelbar nachdem der PROCEDURE DIVISION hinzuzufügen, verwenden Sie den Outline View. Klicken Sie auf die PROCEDURE DIVISION, um den Cursor auf dieser Zeile zu positionieren.

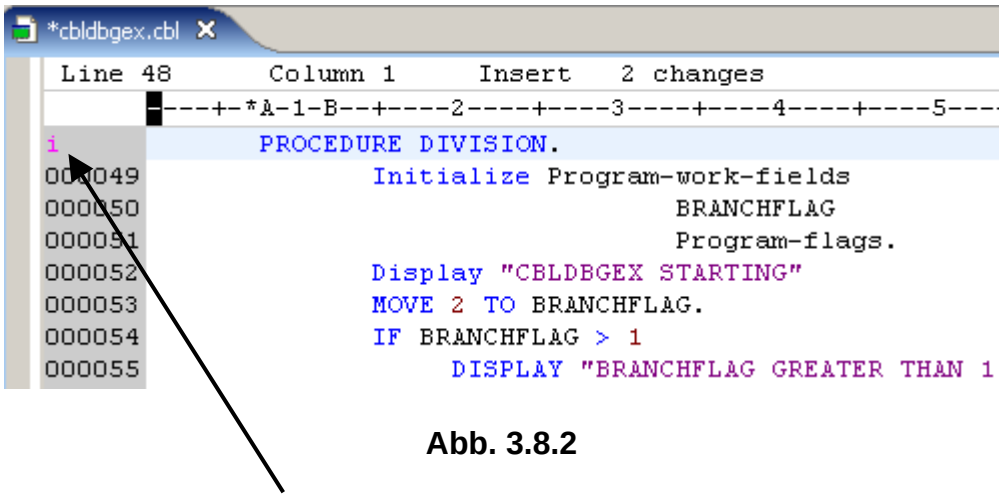


Abb. 3.8.2

Dann das Command “i” in den grauen Bereich (Line Command Area) für diese Zeile eingeben.

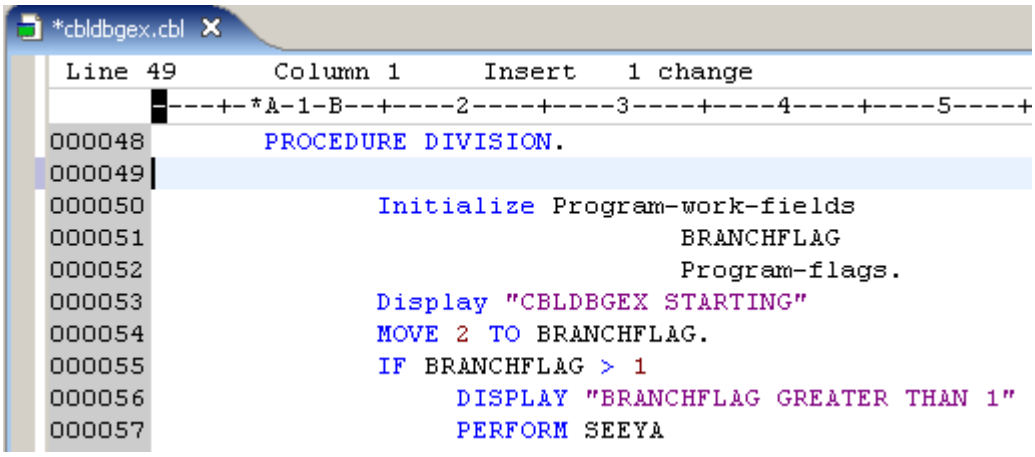


Abb. 3.8.3

Enter, um eine Leerzeile zu erzeugen. Alternativ könnten Sie CTRL + ENTER benutzen um eine Leerzeile zu erzeugen.

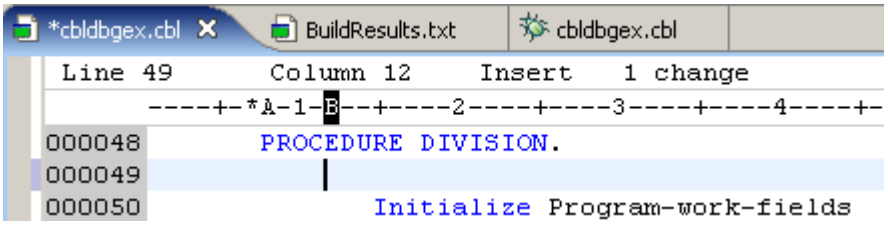


Abb. 3.8.4

Zweimal Tab Taste bewegt den Cursor auf Position 12.

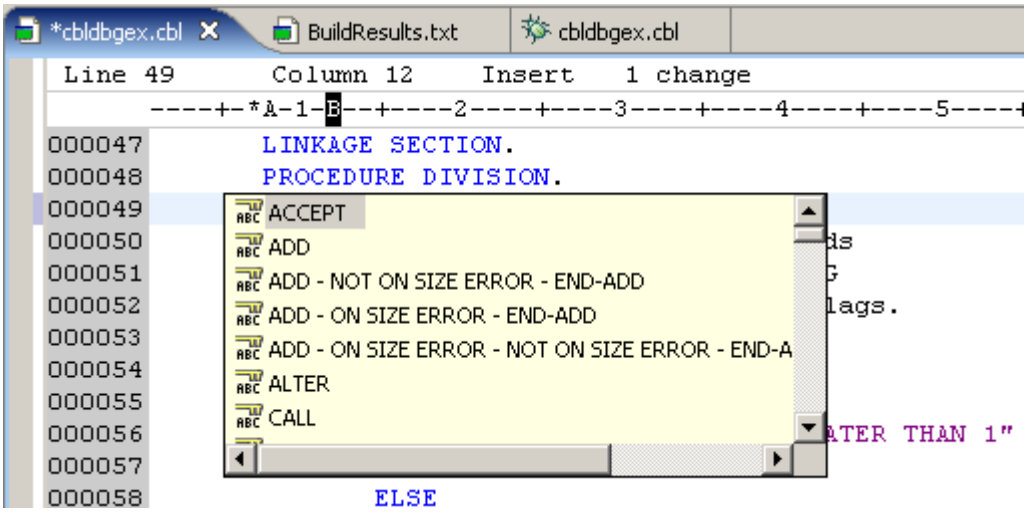


Abb. 3.8.5

Benutzen wir eine nette Eigenschaft: „Code Assist“. Mit zweimal Tab den Cursor nach Spalte 12 bewegen. CTRL + SPACE eingeben. Sie sehen eine mögliche Liste aller COBOL Statements.

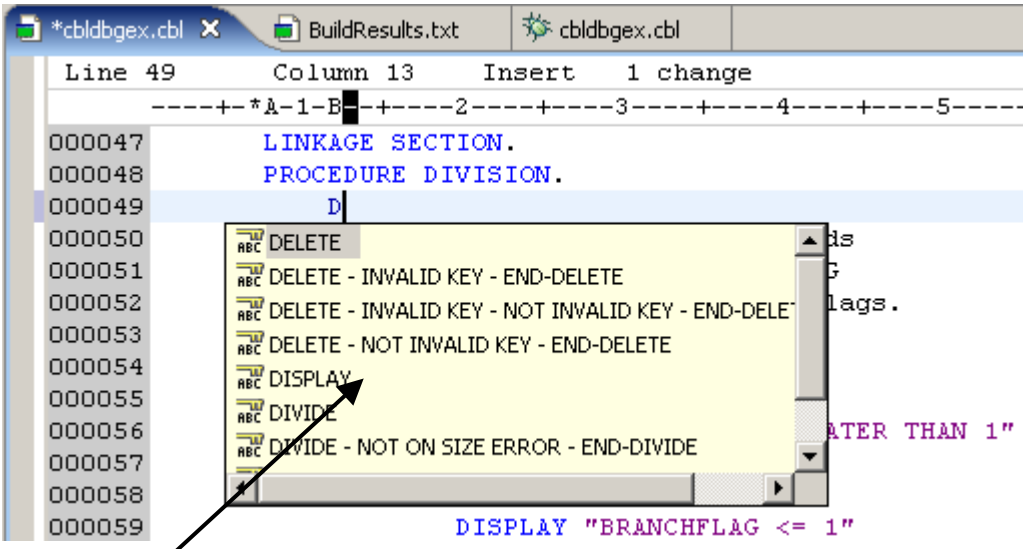


Abb. 3.8.6

Buchstaben D eingeben. Angezeigt wird eine Liste aller Cobol Statement, die mit D anfangen. 2k auf DISPLAY.

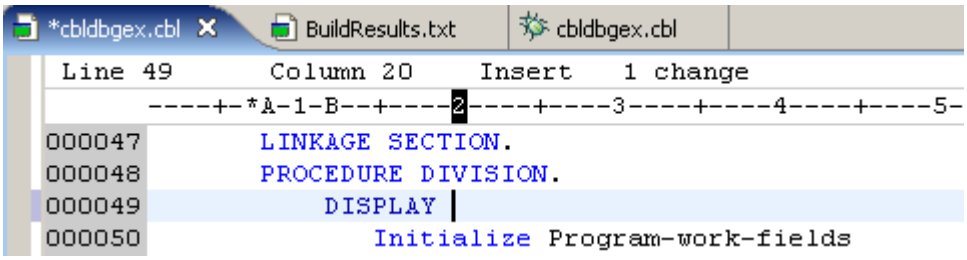


Abb. 3.8.7

Eingeben “Enter name or Q to quit”, Leerzeichen eingeben,

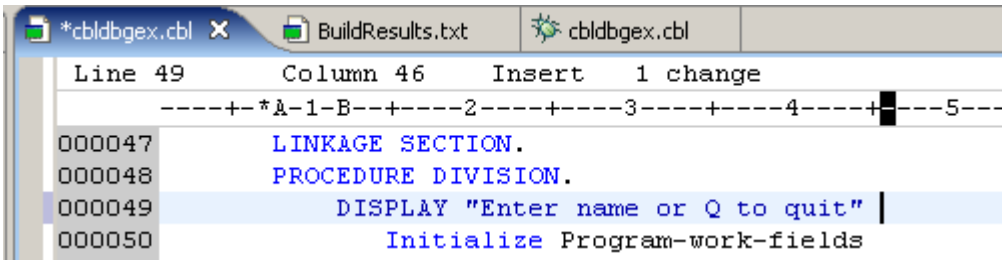


Abb. 3.8.8

wieder CTRL + SPACE,

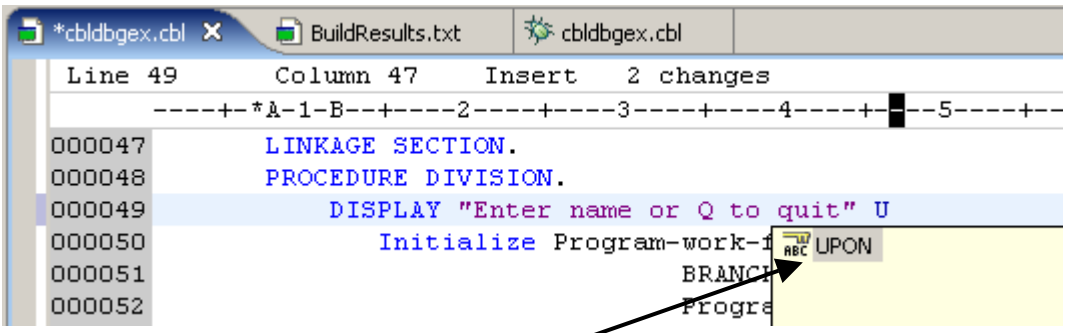


Abb. 3.8.9

ein „U“ eingeben, und 2k auf UPON

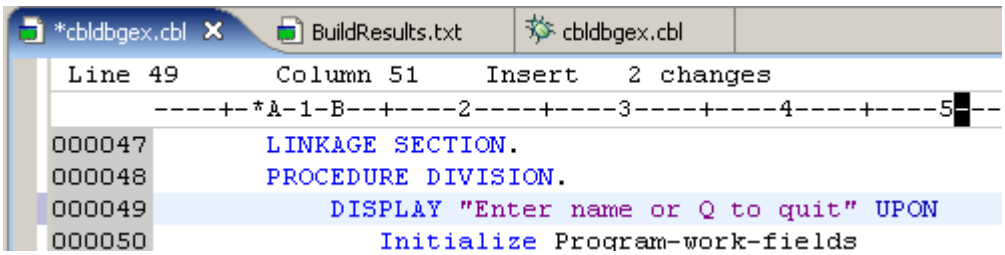


Abb. 3.8.10

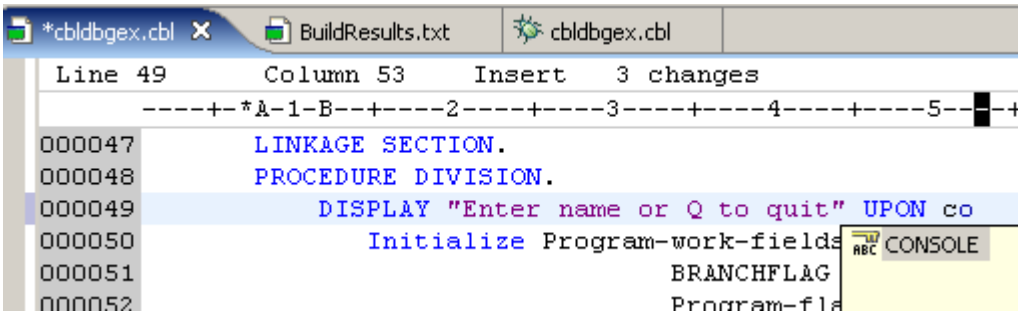


Abb. 3.8.11

nochmals CTRL + SPACE, , Eingabe “co”, und 2k auf CONSOLE

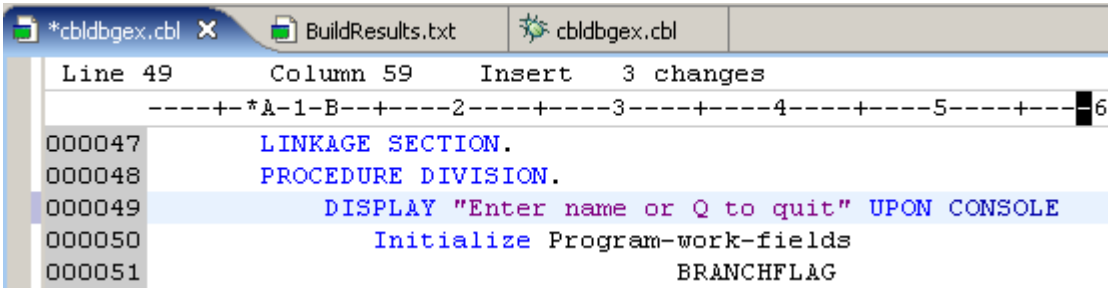


Abb. 3.8.12

Die Verwendung von STRG + SPACE ist sehr hilfreich, wenn wir Daten-Namen verwenden möchten. Zum Beispiel, wenn Sie den Befehl ACCEPT verwenden, um Daten von der Konsole in einen Daten-Namen zu kopieren, wird eine Liste der möglichen Daten-Namen gezeigt, welche diesen Wert aufnehmen könnten. In einem kleinen Programm mag diese Funktion nicht so wichtig sein, aber in einem großen Programm, mit vielen Daten Namen, kann diese Fähigkeit sehr nützlich sein.

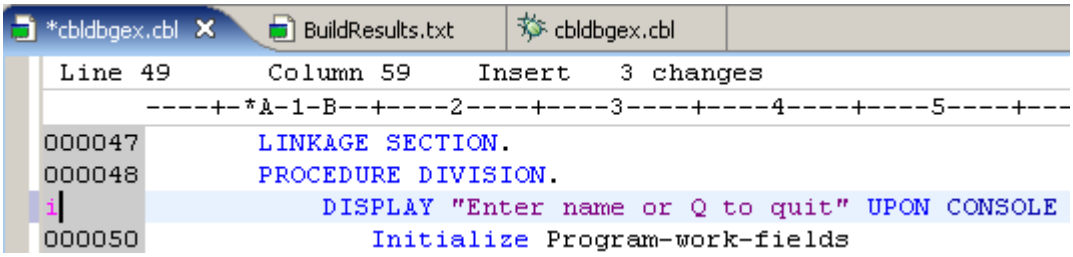


Abb. 3.8.13

Type “i “in in den grauen Bereich auf der linken Seite.

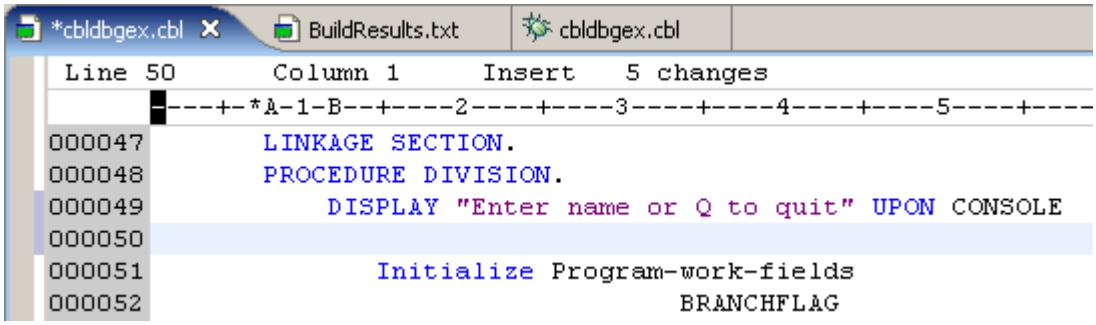


Abb. 3.8.14

Enter, um eine Leerzeile zu erzeugen. (Dies könnte mit CTRL + ENTER geschehen, wenn wir den LPEX _Esitor an Stelle des ISPF Editors benutzen würden).

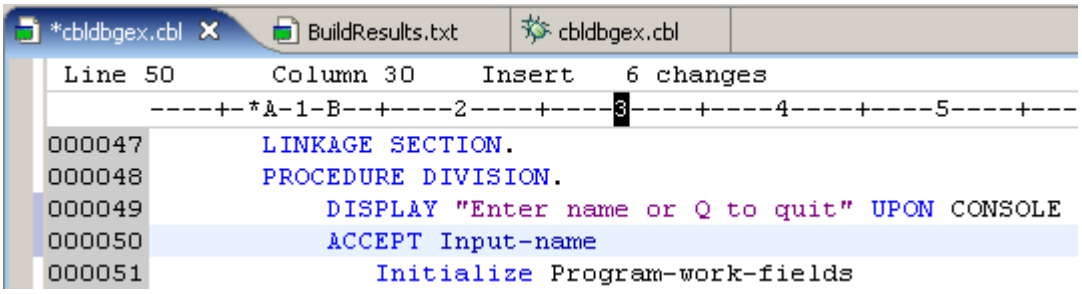


Abb. 3.8.15

2 x Tab, Type ein ACCEPT Statement. Benutzen Sie CTRL + Space, um den Input-Namen einer Variablen hinzuzufügen.

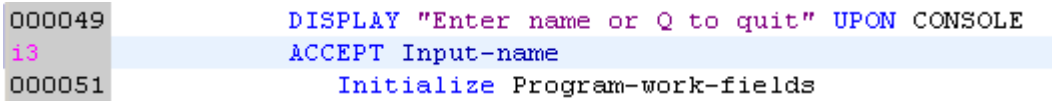


Abb. 3.8.16

Type i3 auf Zeile 50

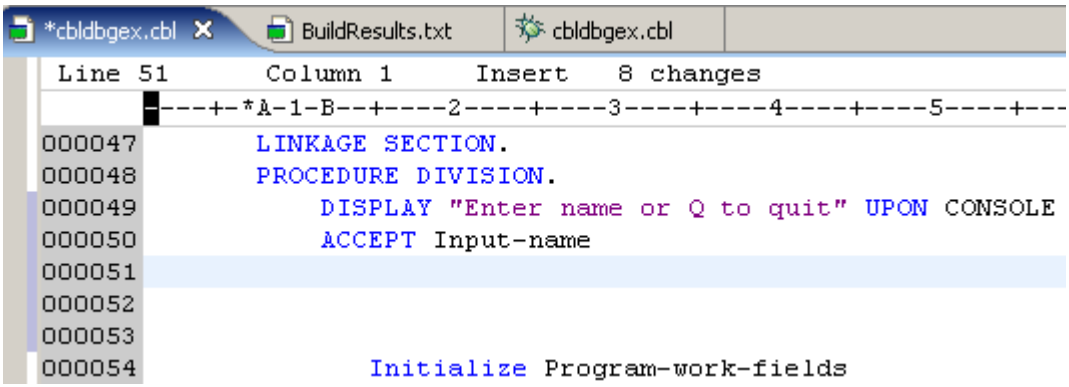


Abb. 3.8.17

Enter drücken um 3 Leerzeilen einzufügen.

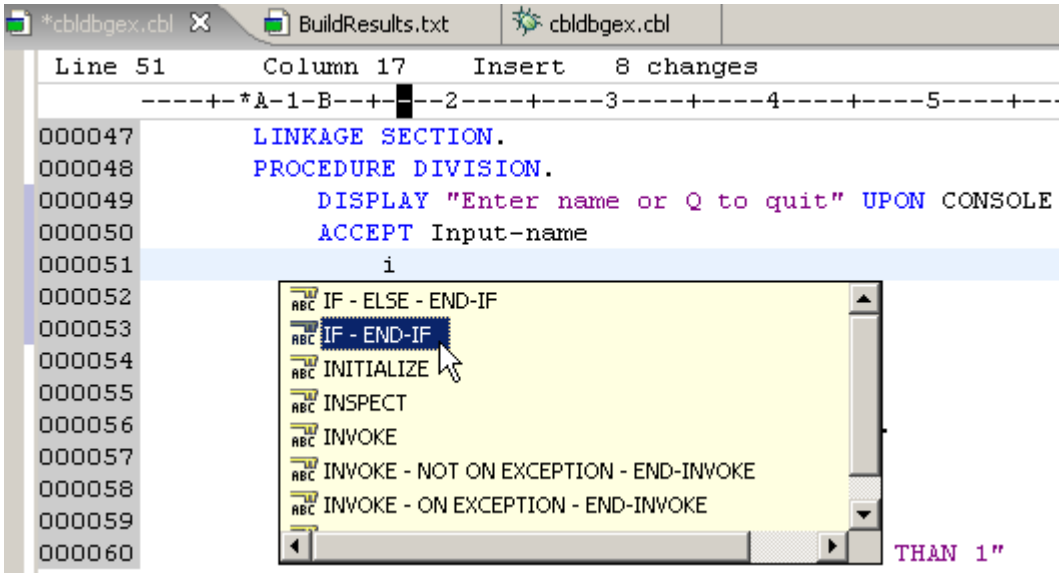


Abb. 3.8.18

Ein IF – END-IF Statement einfügen durch eingabe eines “i”; anschließend “IF - END-IF” von dem Content Assist Dropdown Menu selektieren.

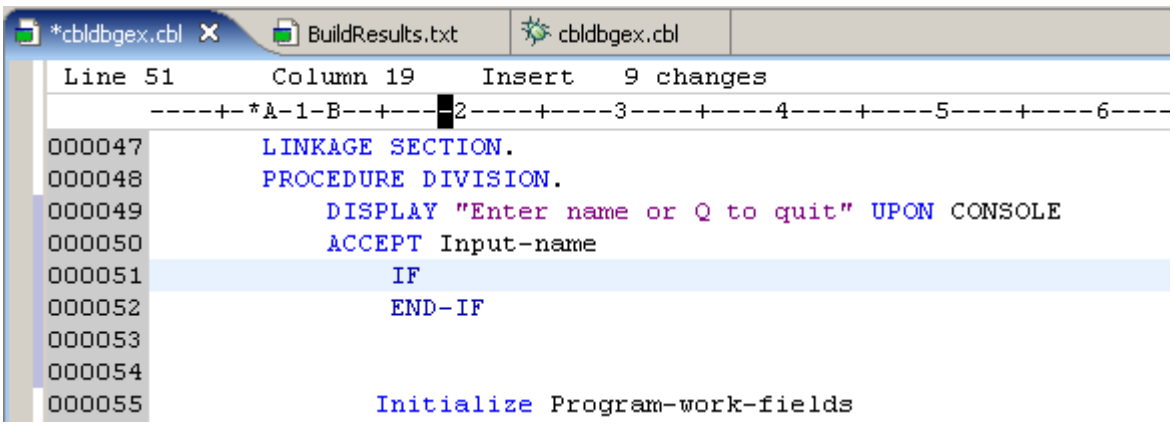


Abb. 3.8.19

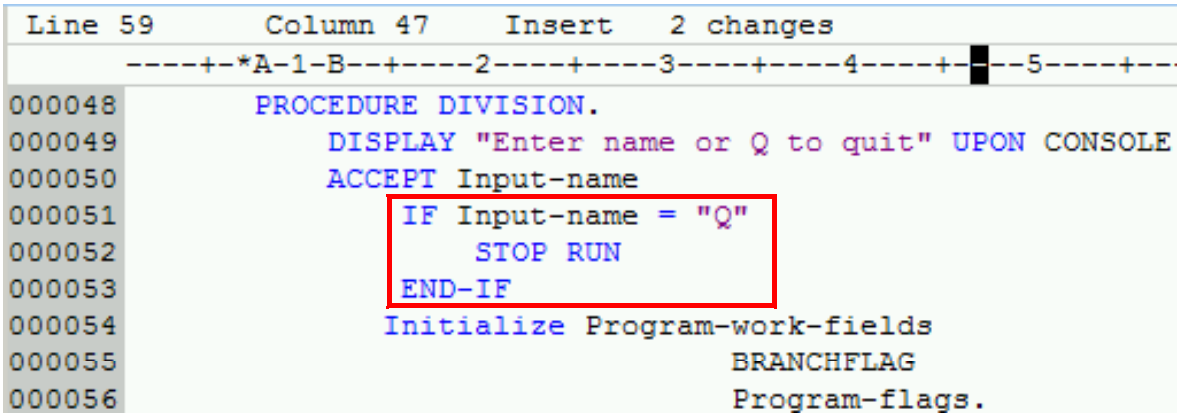


Abb. 3.8.20

Statement vervollständigen. Neuen Text in Zeilen 51 - 53 eingeben, wie hier dargestellt.

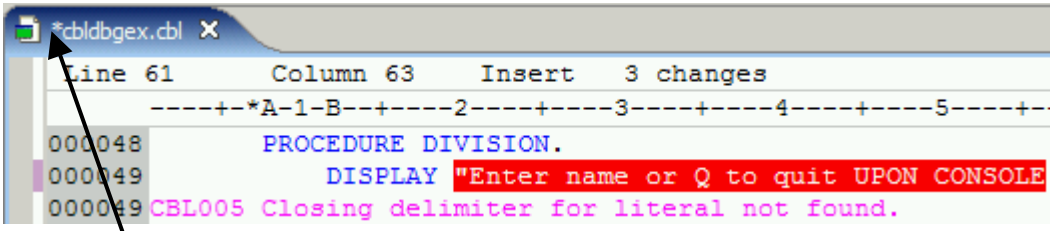


Abb. 3.8.21

Weil der COBOL-Editor ein Smart-Editor ist, können einige Fehler beim Kodieren gefunden werden. Zum Beispiel, wenn wir die ersten Anführungszeichen (") Im Statement DISPLAY beseitigen und drücken die Eingabetaste (oder bewegen den Cursor auf einer andere Zeile), erhalten wir eine Fehler Meldung wie oben angezeigt.

Korrigieren Sie den Fehler, und speichern Sie mit CTRL + S.

Beachten Sie, das Sternchen * auf der linken Seite des Titels verschwindet nach der Speicherung. 

Machen Sie noch einen lokale Syntax Check, um die Syntax wieder zu überprüfen.

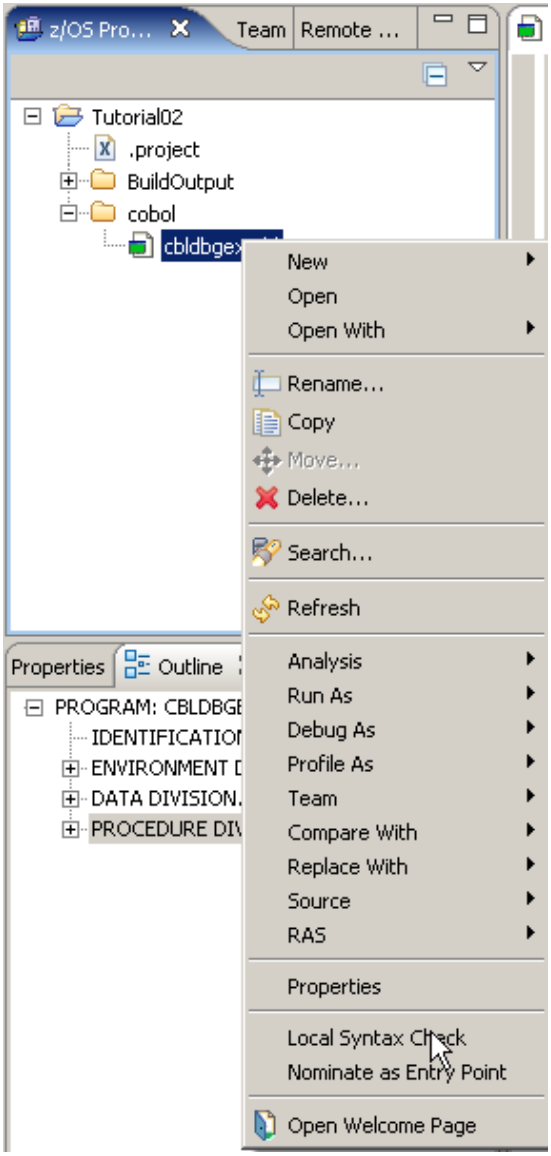


Abb. 3.8.22

1kr auf cbldbgex.cbl. Selektiere Local Syntax Check, 1k.

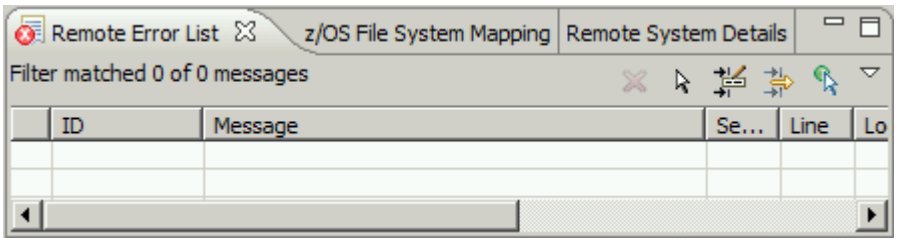


Abb. 3.8.23

Klicken Sie in die Ansicht Remote Error List. Es sollten keine Fehler in dieser Ansicht aufgeführt sein. Wenn Sie Fehler haben, korrigieren Sie diese und überprüfen Sie die Syntax wieder.

Hinweis: Falls Sie Fehler haben und Zeit für die Korrektur nicht verlieren wollen, laden Sie einfach den Code mit den genannten Änderungen aus dem Ordner c: \ wdz Tutorials \ Ressourcen \ Tut02solution. Verwenden Sie einfach File → Import → File system → Next → und selektieren sie die Datei cbldbgex.cbl

Ein weitere nette Hilfe bei der Codierung des Programms ist die Help Built-in Funktion.

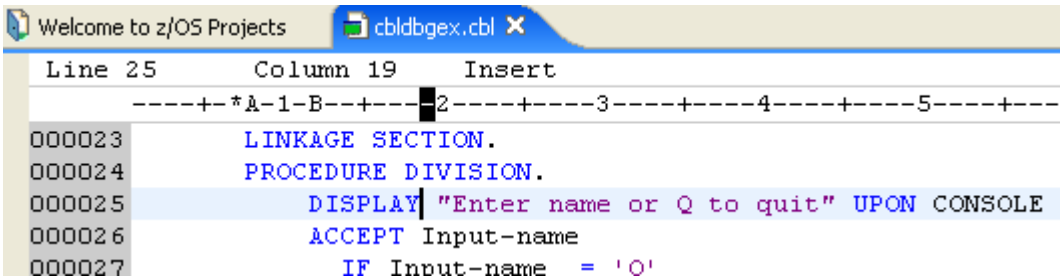


Abb. 3.8.24

Bewegen sie den Cursor zu em Statement DISPLAY und drücken Sie F1. Ein Help Fenster wird geöffnet und zeigt das selektierte Statement. Dies kann für jedes beliebige Statement benutzt werden.

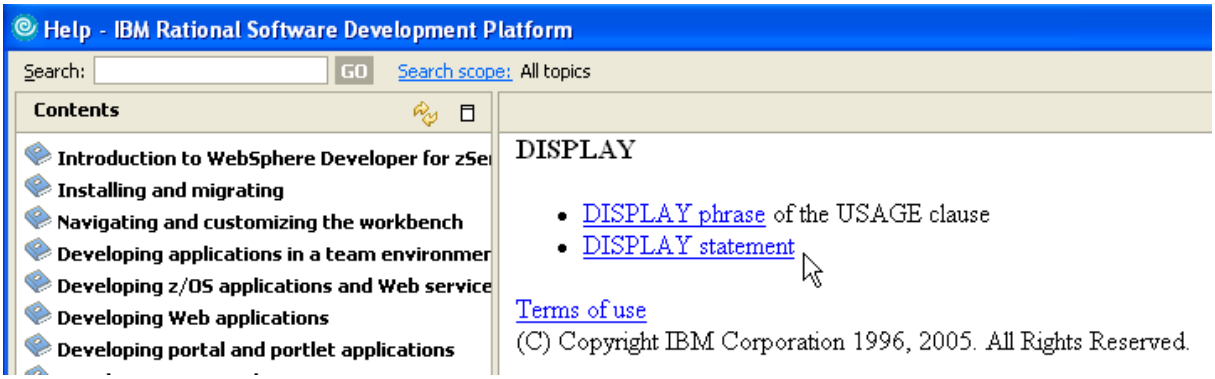


Abb. 3.8.25

Nach einiger Zeit wird das Help Fenster geöffnet. In unserer VMWare Umgebung kann dies etwas länger dauern.

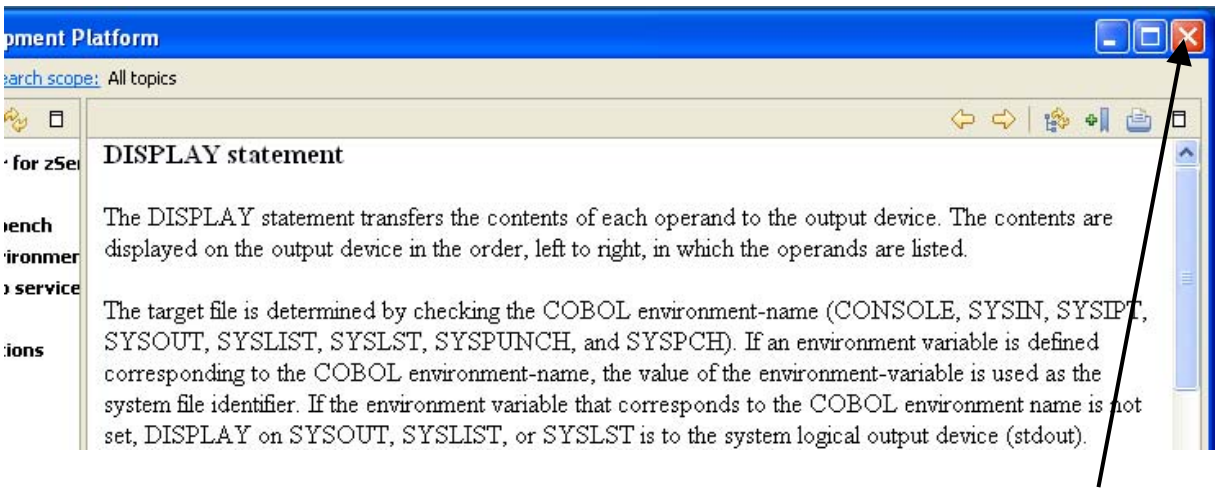


Abb. 3.8.26

1k auf das DISPLAY Statement produziert mehr Information. Das Help Window wieder schließen

4. Kompilieren, Linken und Ausführen des lokalen COBOL-Programms

Nachdem Sie Ihre Quellcode fertigt gestellt haben, führen Sie nun Ihr Programm aus. Es wird in der lokalen COBOL-Laufzeitumgebung (d.h. unter Windows) compiled und ausgeführt. Darüber hinaus werden Sie den Debugger verwenden, um den Code zur Laufzeit Step für Step auszuführen.

4.1. Erstellen eines ausführbaren COBOL-Programms

Vor der Übersetzung und dem Erstellen der ausführbaren Datei, löschen Sie zunächst evtl. von früher her vorhandenen ausführbaren Code.

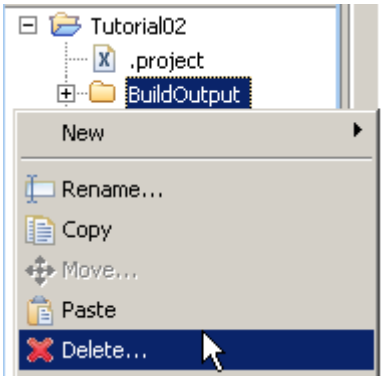


Abb. 4.1.1

Der Folder „BuildOutput“ enthält allen übersetzten und ausführbaren Code. BuildOutput selektieren, 1kr. Das Context Menu zum Löschen benutzen.

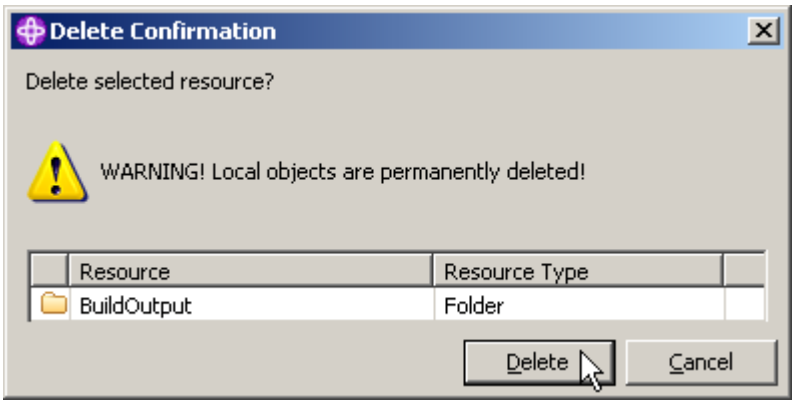


Abb. 4.1.2

Das Delete Confirmation Window erscheint. 1k Click auf Delete

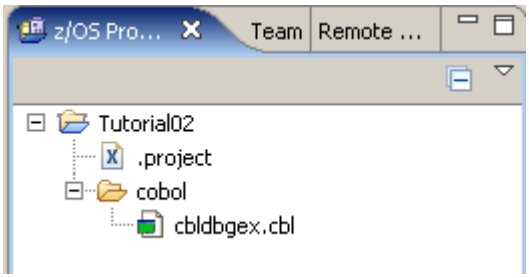


Abb. 4.1.3

Hinweis: Wir löschen den BuildOutput Ordner nur um zu zeigen, dass er neu erstellt wird. Im Augenblick wäre dieser Schritt vermutlich nicht wirklich notwendig.

To Build the program:

Wir benutzen den z/OS Projects View. Expandiere COBOL. 1kr auf cbldbgex.cbl, und

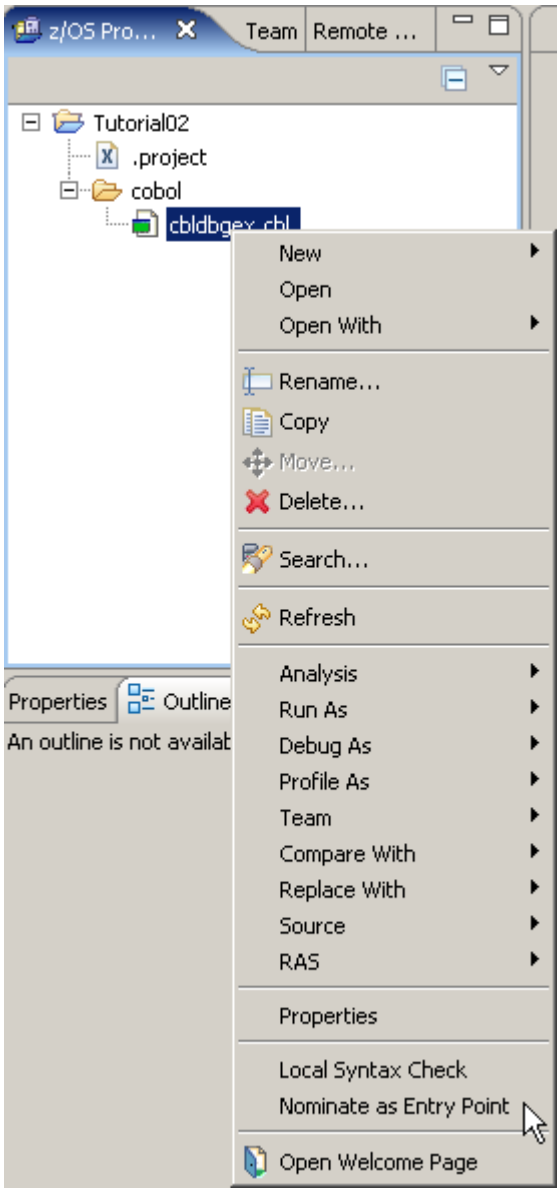


Abb. 4.1.4

selektiere “Nominate as Entry Point”. 1k.

Wenn Sie das nicht tun, und haben mehrere Programme im selben Projekt, wird eine DLL mit dem Projekt-Namen erzeugt. Das ist hier nicht der Fall, aber es ist besser, diesen Schritt immer auszuführen.

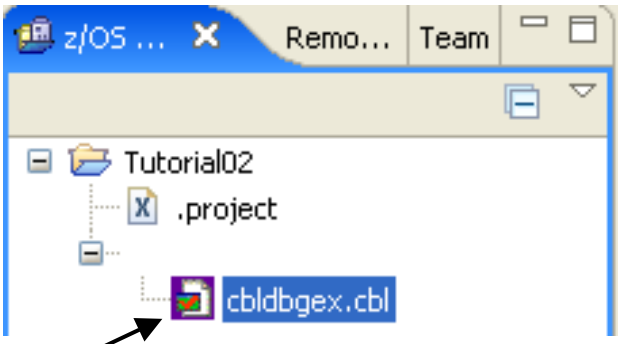


Abb. 4.1.5

Ein rotes Häkchen erscheint in dem Icon für das Programm, sobald dieses als Einstiegspunkt (Entry Point) nominiert worden ist.

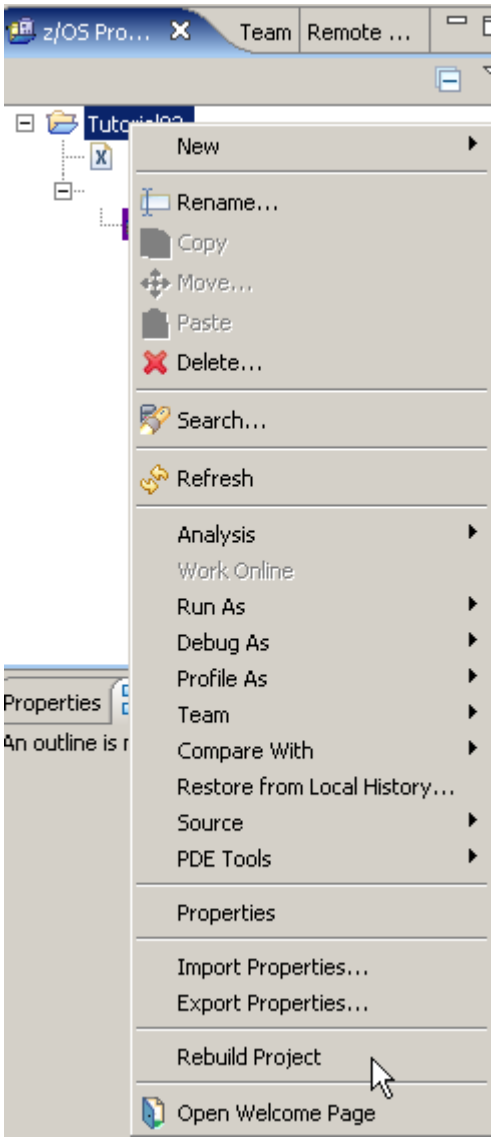


Abb. 4.1.6

Um das Programm zu erstellen, 1kr auf Projekt Tutorial02. Selektiere Rebuild Project, 1k.

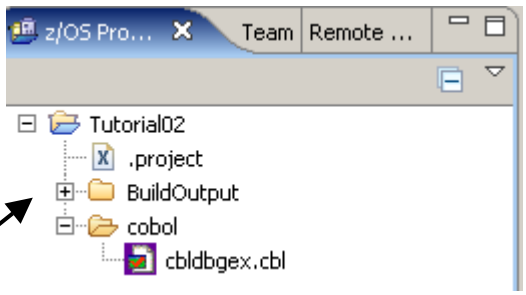


Abb. 4.1.7

Beachten Sie, BuildOutput wird neu erstellt.

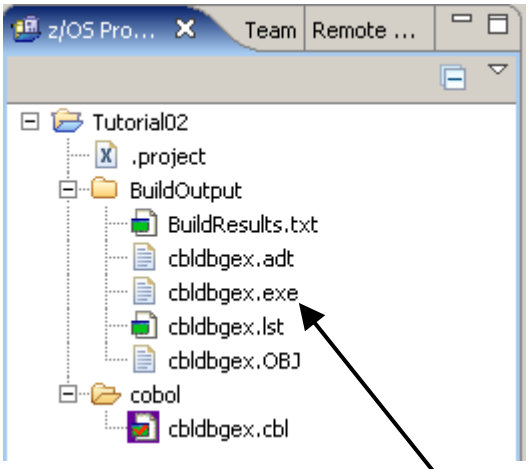


Abb. 4.1.8

Expandiere BuildOutput um den Ordner Inhalt anzuzeigen. Eine exe-Datei wurde erstellt. Optional kann eine DLL erstellt werden (dies ist z.B. für CICS TS Programme erforderlich).

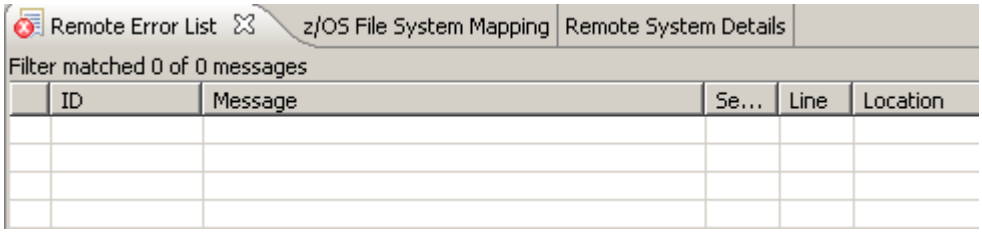


Abb. 4.1.9

Es sollten keine Fehler im Remote Error List View aufgeführt werden.

4.2. Überprüfen Sie die lokale COBOL Compilation und Link Edit

Der COBOL-Compiler speichert die Ausgabe im BuildOutput Ordner.

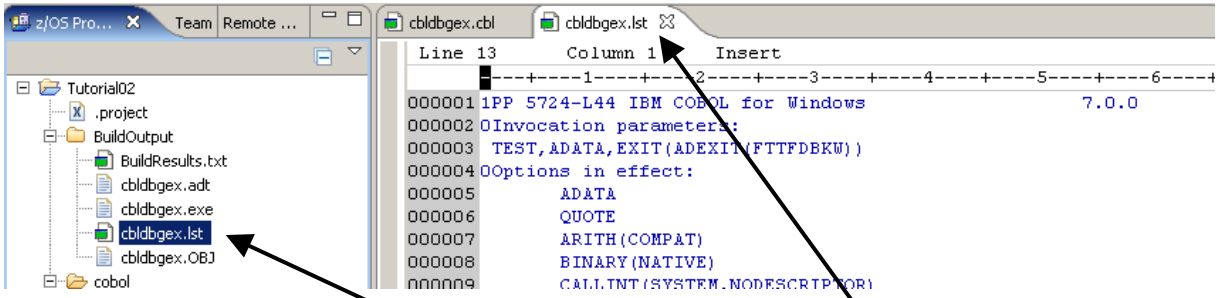


Abb. 4.2.1

Um das compile Listing azusehen, 2k auf cbldbgex.lst. Ein weiterer Tab erscheint.

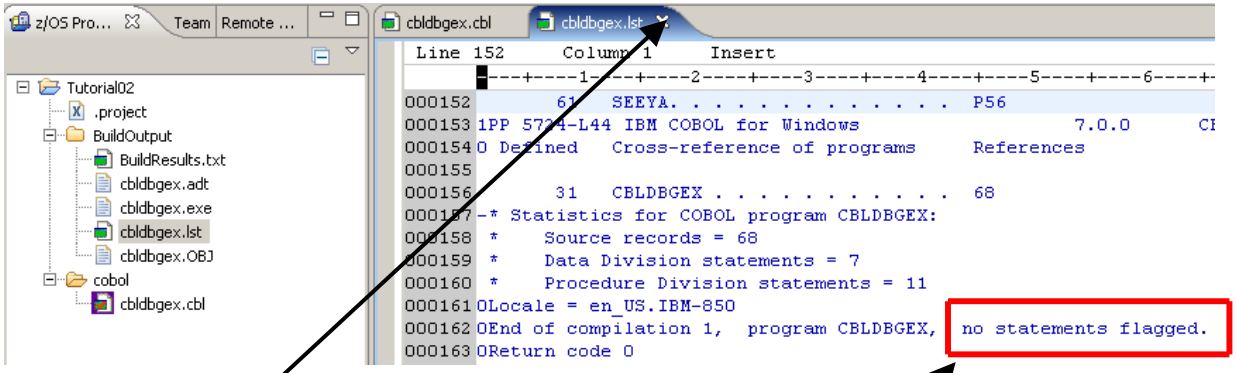


Abb. 4.2.2

Scroll down zum Ende des Listing. Es werden keine Compile Fehler angezeigt. Benutzen Sie die Ctl + End Tasten-Kombination “to go to the end”.

Schließen Sie den Editor.

4.3. Erstellen einer Run Launch Konfiguration, und Ausführen des COBOL-Programms.

Da wir dieses Programms viele Male ausführen wollen, es ist eine gute Idee, eine Debug-Launch-Konfiguration zu erstellen. Dies erleichtert das Ausführen und Debuggen eines spezifischen Programms. Der folgende Schritt erstellt eine Launch Konfiguration, welche das kompilierte COBOL-Programm lädt:

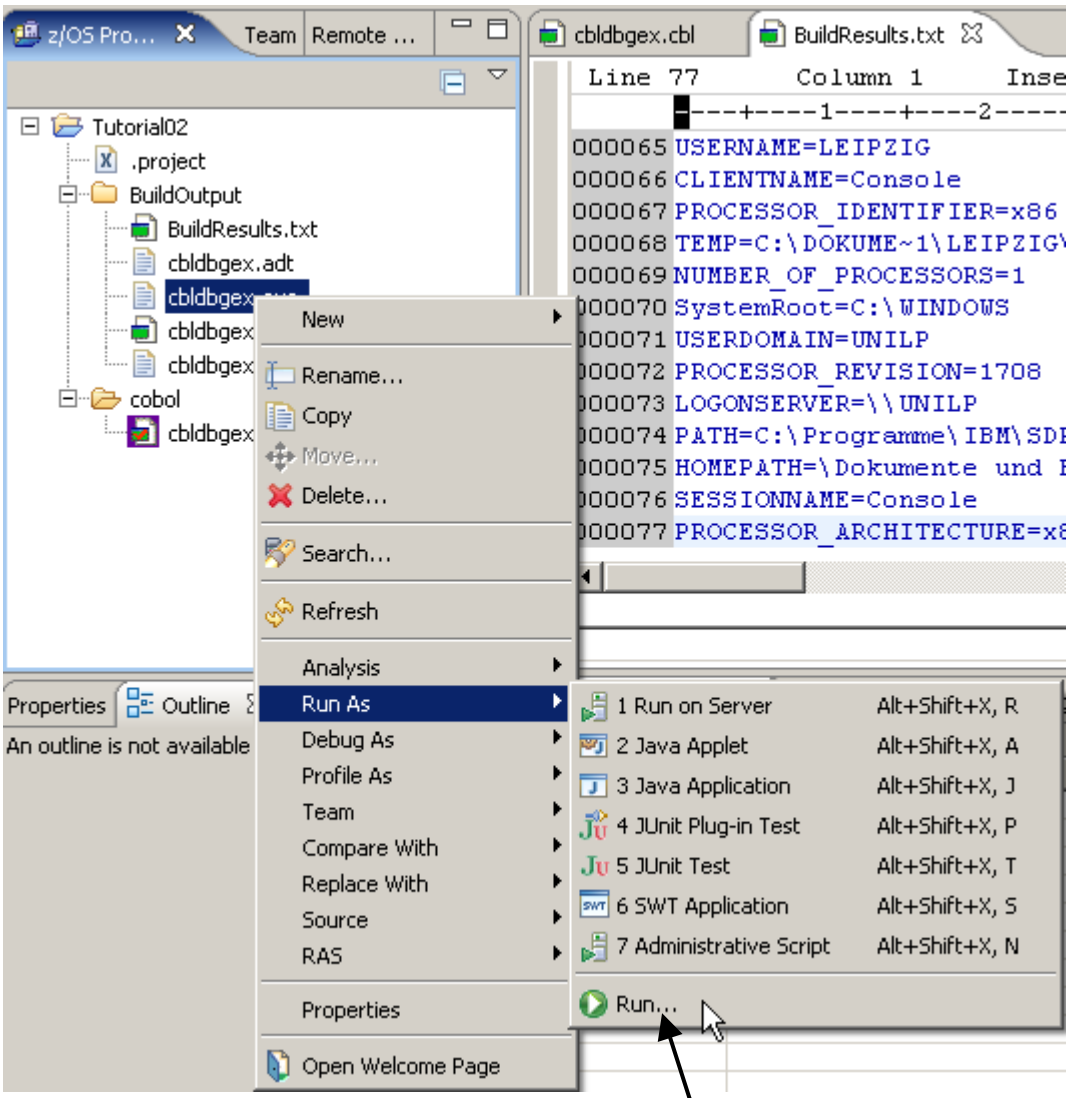


Abb. 4.3.1

1kr auf cbldbgex.exe. Selektiere Run As → Run… (<u>nicht</u> Run on Server)

1k

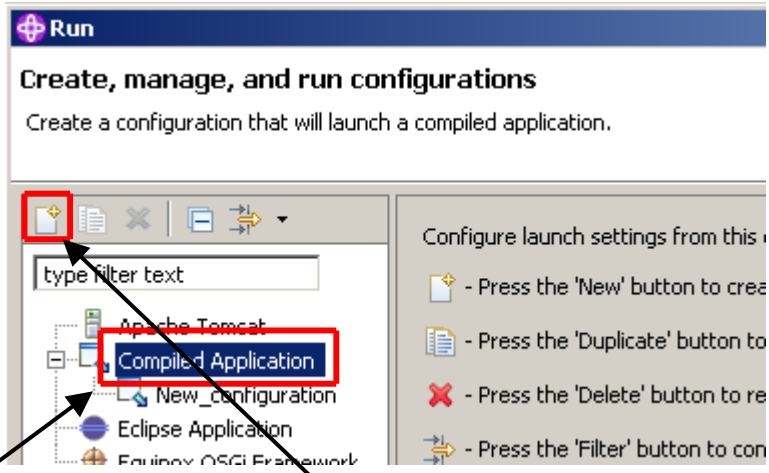


Abb. 4.3.2

Selectiere Compiled Application, 1k auf den New Button.

Als Folge erscheinen die Launch Configuration Tabs und Eingabefelder auf der rechten Seite der Dialog Box.

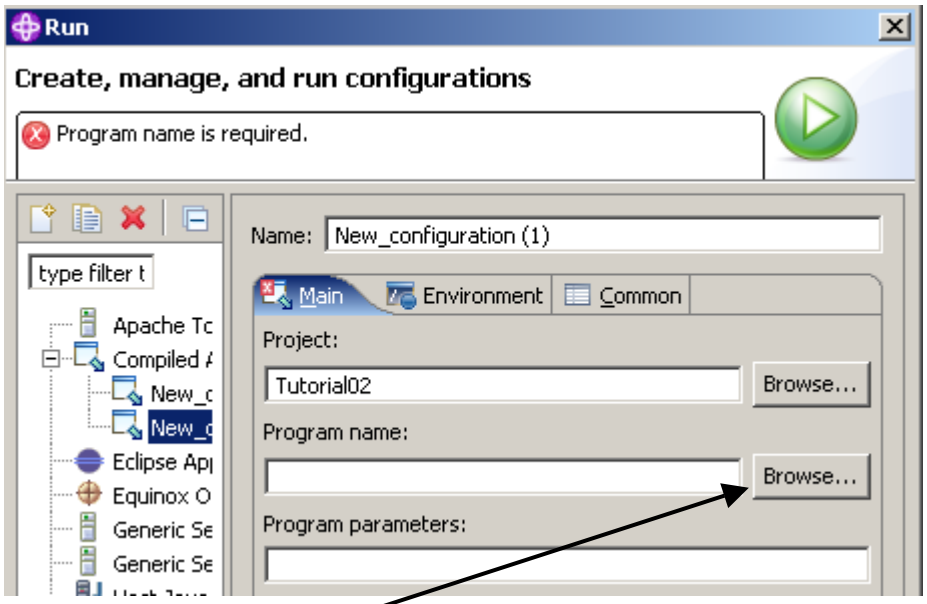


Abb. 4.3.3

Benutzen Sie den Browse button neben Program name, um cbldbgex.exe im BuildOutput Verzeichnis zu finden, und …

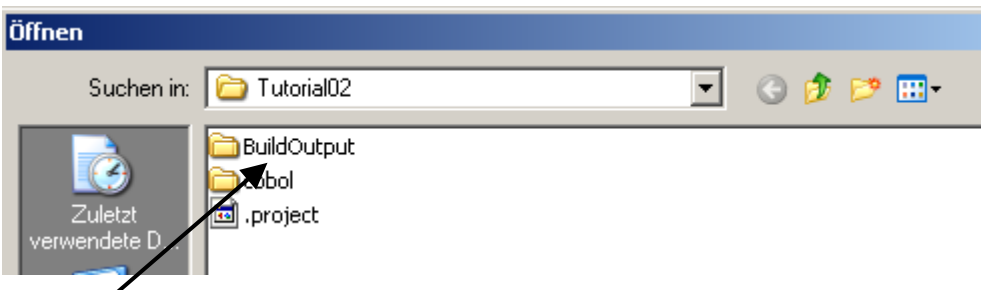


Abb. 4.3.4

1k auf BuildOutput folder um ihn zu öffnen, 2k .

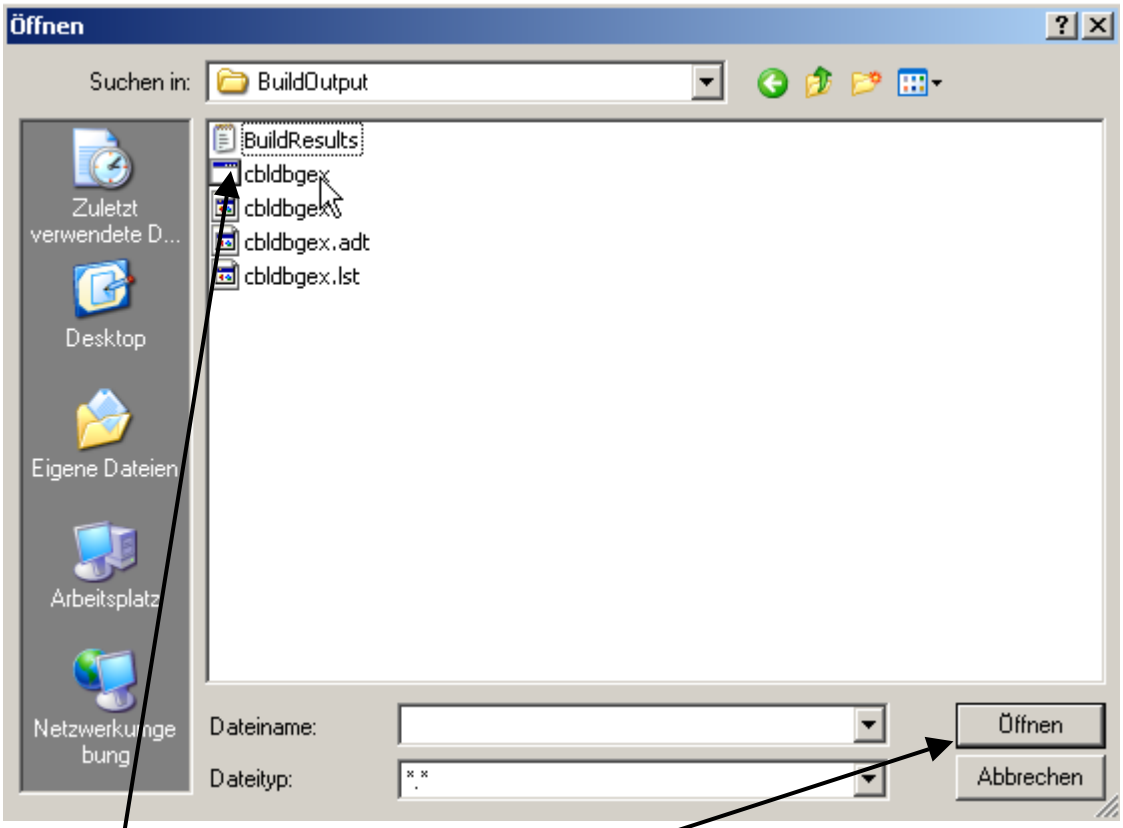


Abb. 4.3.5

Selektiere cbldbgex.exe. 1k auf open.

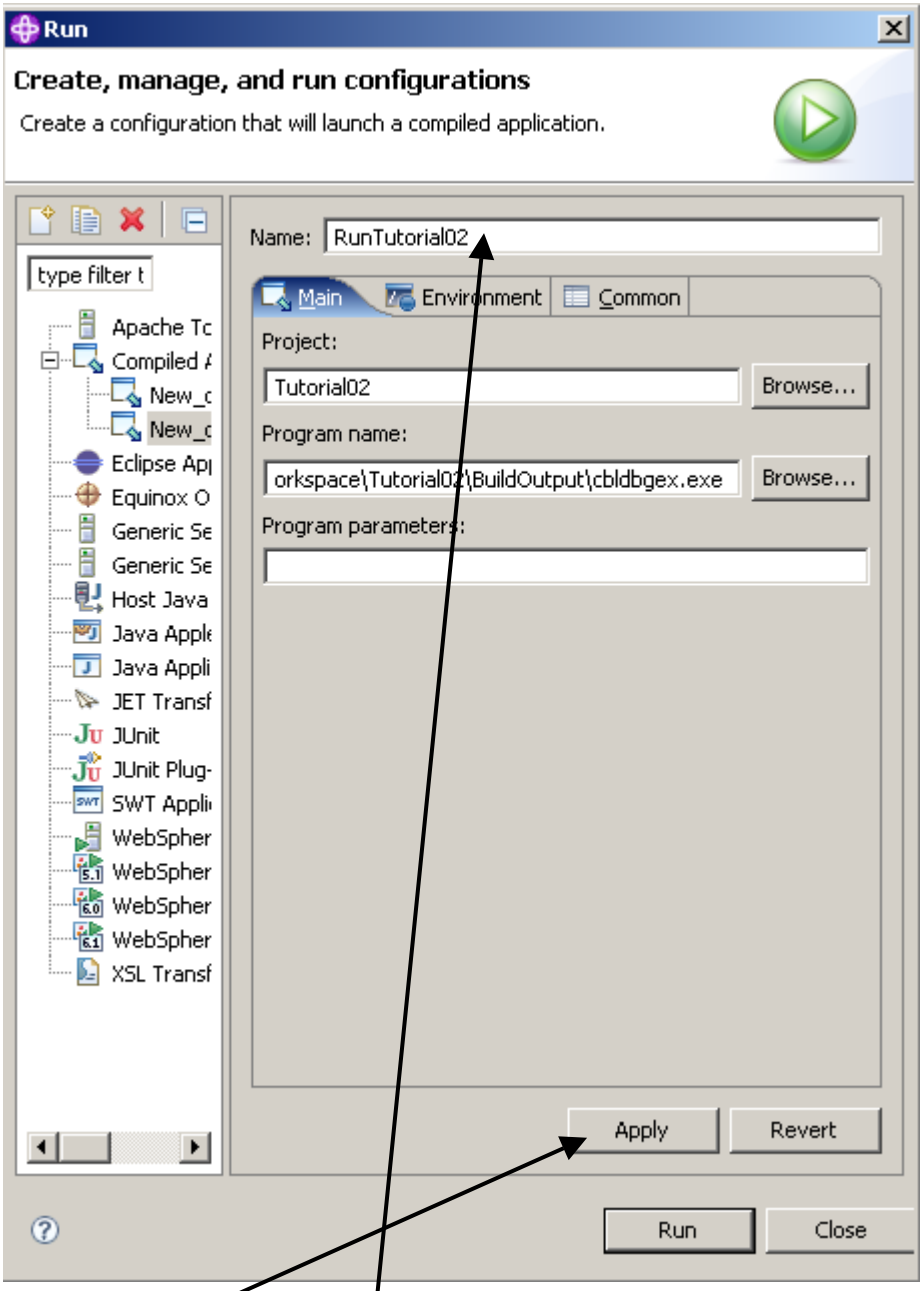


Abb. 4.3.6

In den Name Feld, einen Namen wie z.B. „RunTutorial02“ eingeben.

1k auf „Apply“ rettet (saves) die Änderungen. Jetzt…

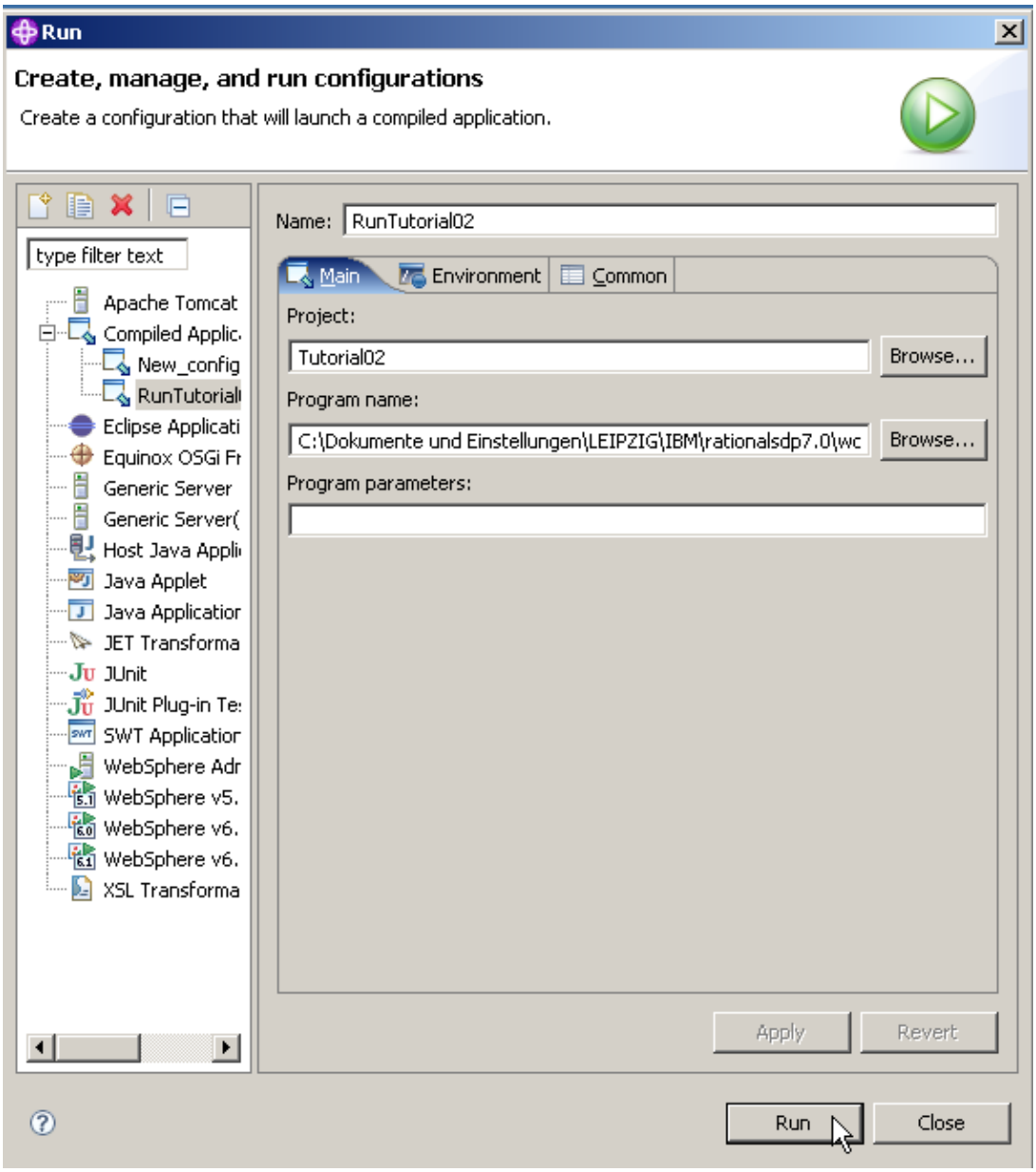


Abb. 4.3.7

Einen Namen wie RunTutorial02 eingeben. 1k auf Run.

Die Ausführung beginnt und normalerweise würden Sie diesen Dialog in einem neu geöffneten Konsole Fenster sehen:

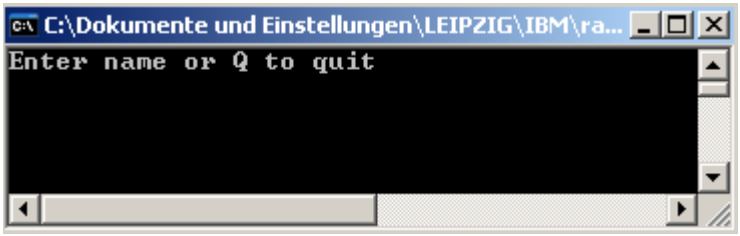


Abb. 4.3.8

Einen Namen eingeben, und …..

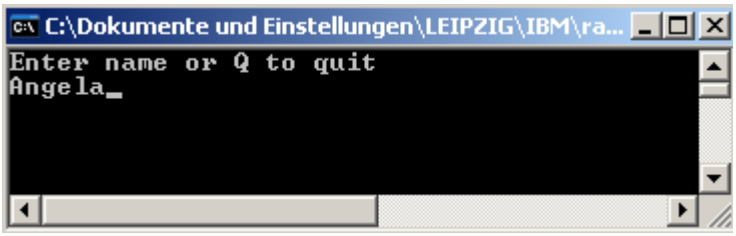


Abb. 4.3.9

Enter eingeben.

Eine Antwort wird zwar korrekt wiedergegeben, aber das Konsole Fenster wird wieder geschlossen, ehe Sie Zeit haben, etwas zu sehen.

Aufgabe: ***Erweitern Sie das Cobol Beispiel Programm, so dass das in Abb. 4.3.9 dargestellte Fenster permanent sichtbar bleibt, und z.B. durch Enter geschlossen wird.***

Aufgabe: ***Senden Sie einen entsprechenden Screen shot an Ihren Betreuer.***

Aufgabe: ***Als Option machen Sie sich mit den in Abschnitt 5 beschriebenen RDz Test/Debug Funktionen vertraut.***

Selbst-Test

- Warum ist es sinnvoll, eine Debug-Launch-Konfiguration zu erstellen ?

5. Testing/Debugging des COBOL Programms

5.1 Debug Perspektive

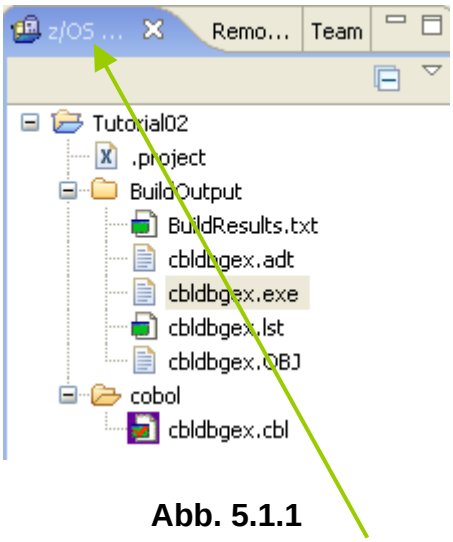


Abb. 5.1.1

Zur Erinnerung: Eclipse und RDz benutzen das Konzept einer Perspective. Derzeitig benutzen wir die z/OS Projects Perspective. Dies wird in dem lokalen Fenster angezeigt.

Eine Perspective definniert die Benutzerschnittstelle (GUI), mit der Sie arbeiten.

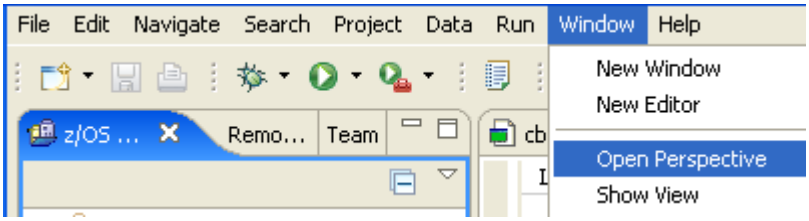


Abb. 5.1.2

Sie können jederzeit die gerade benutzte Perspektive wechseln: 1k auf Window → open Perspective. Am wichtigsten ist: Wenn Sie den Überblick verloren haben, können Sie immer zu der jetzt vertrauten z / OS Projects Perspektive zurückzukehren, einfach durch 1k Window → Open Perspective.

OK, fangen wir mit dem Testen und Debuggen unseres Cobol-Programms an. Hierfür werden wir schon bald in die Debug-Perspektive wechseln, und später, wenn wir fertig sind, zurück zu der z/OS Projects Perspektive.

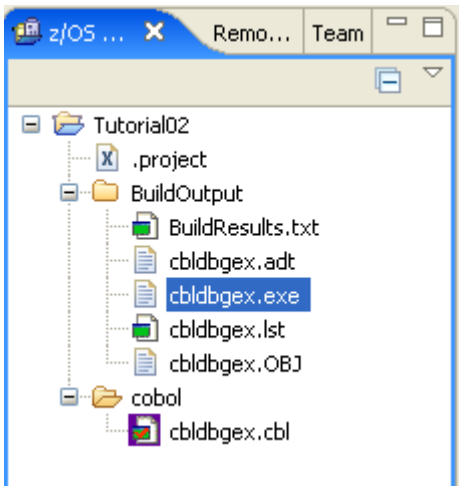


Abb. 5.1.3

Da wir die Launch Configuration bereits erstellt haben (siehe Abschnitt 4.3), 1kr auf cbldbgex.exe, und....

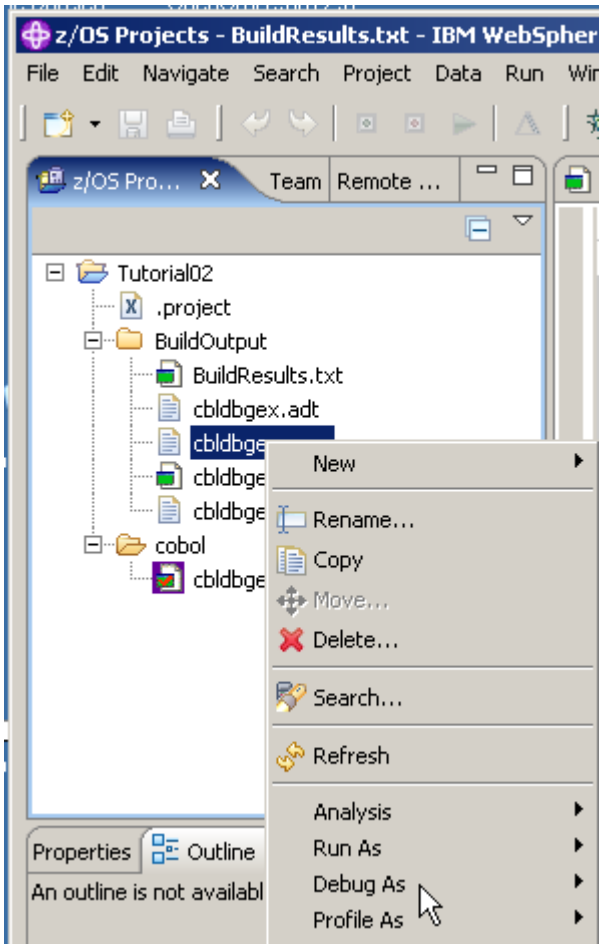


Abb. 5.1.4

selectiere “Debug As” und dann Debug.

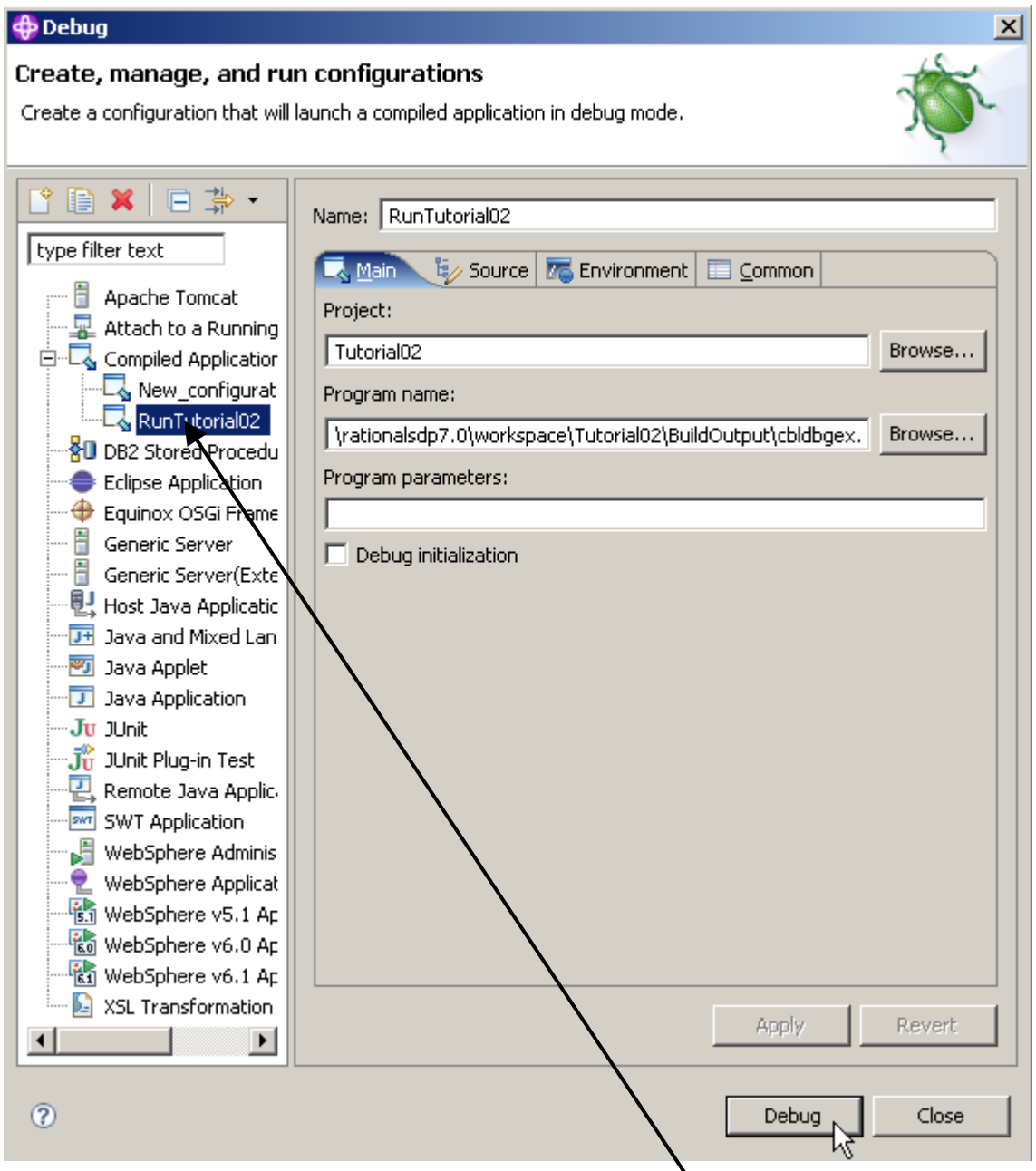


Abb. 5.1.5

Wenn das Debug Window geöffnet wird, selektieren Sie die in Abschnitt 4.3 erstellte Launch Configuration RunTutorial02 und click Debug.

In dem folgenden Dialog Fenster wird gefragt, ob Sie die Debug-Perspektive wechseln wollen. Derzeitig arbeiten sie mit der z/OS Projects Perspektive. Das Umschalten der Perspektiven ist ein normaler Vorgang, wenn Sie RDz benutzen. Sie können immer wieder zur z/OS Projects Perspektive zurückkehren,, indem Sie auf Window → open perspective → z/OS Projects Perspective klicken.

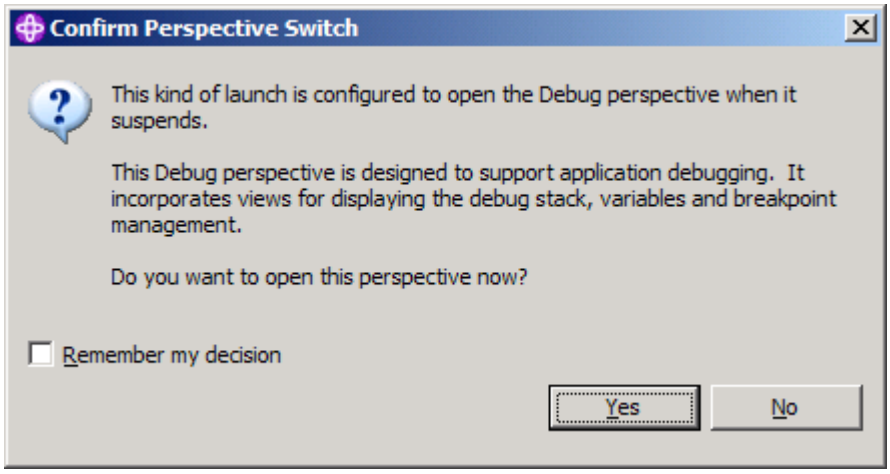


Abb. 5.1.6

Click Yes. Dies schaltet von der z/OS Projects Perspective auf die Debug Pperspective um.

Beachten Sie, das obige Dialog Fenster ist möglicherweise hinter dem Konsole/DOS Fenster versteckt. Möglicherweise müssen Sie das Konsole-Fenster minimieren, um den Dialog zu sehen.

Die Debugger *P*erspective wird geöffnet und das Debug beginnt.

5.2 Debug Steps

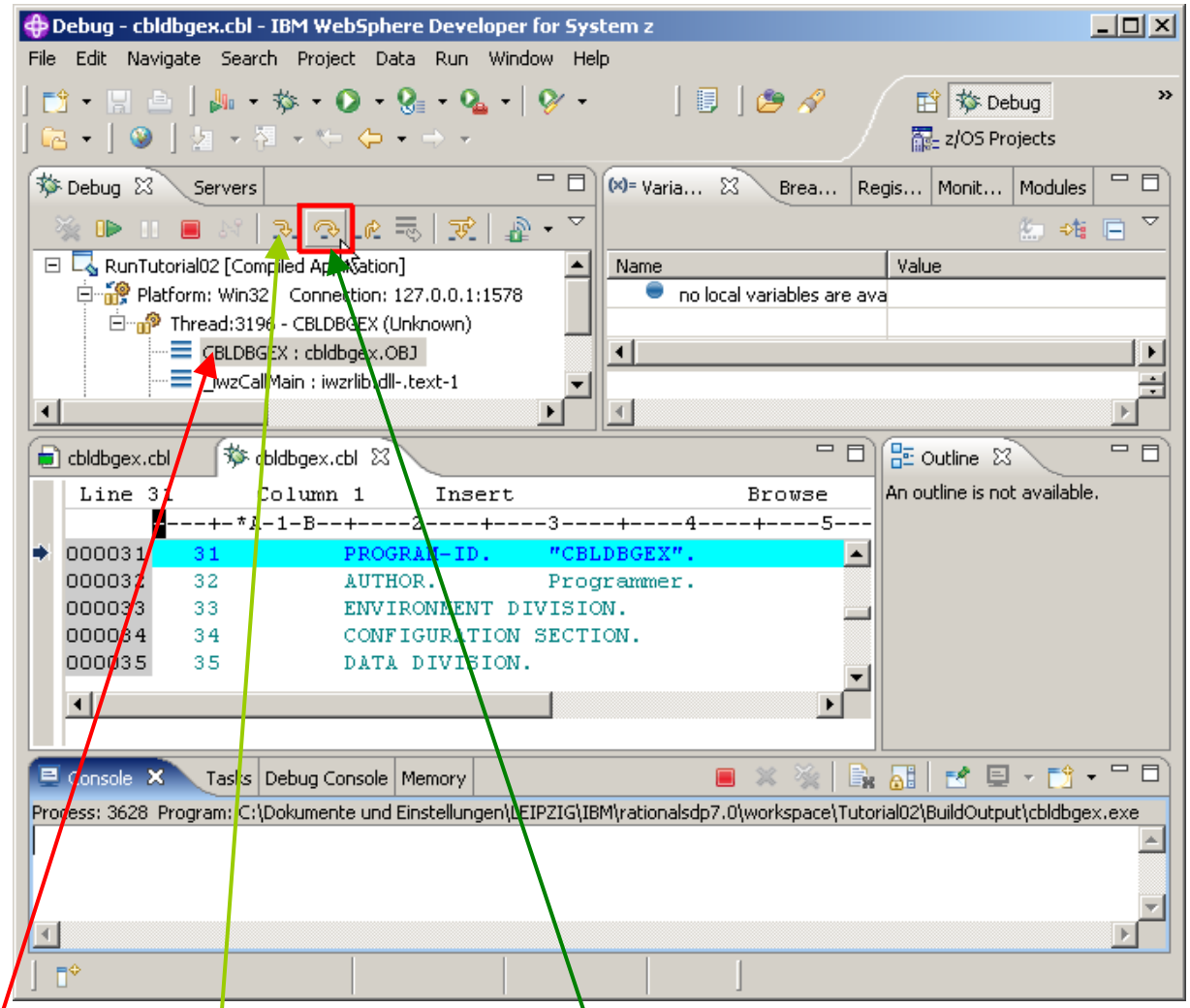


Abb. 5.2.1

Dieser Eintrag sollte hervorgehoben sein.

In der Debug-Ansicht, click auf das Step Over Icon (oderr alternativ hit F6 zwei mal). (Verwenden Sie kein Step Into)., Wenn Sie während dieser Schritte etwas falsch gemacht haben, schließen Sie die Debug-Perspektive und die gerade bearbeitete Datei wieder, und fangen Sie mit Abschnitt „5. Testing/Debugging des COBOL Programms“ erneut von vorne an.

Denken Sie daran, Sie sind jetzt in der Debug-Perspektive. Wenn Sie zu Abschnitt „5. Testing/Debugging des COBOL Programms“ zurückkehren wollen, click auf Window – open Perspective - z/OS project.

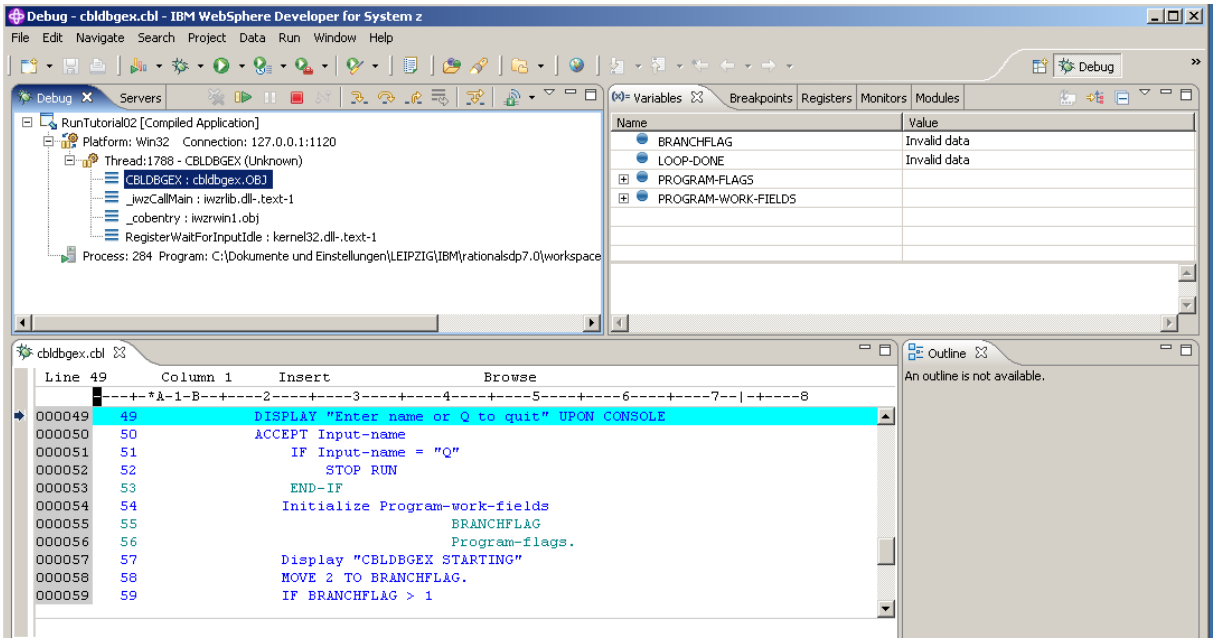


Abb. 5.2.2

Nach einem click auf das Step Over icon ( ) , startet die Ausführung des COBOL Programms. Sie sehen etwas ähnliches wie den hier gezeigten Screen.

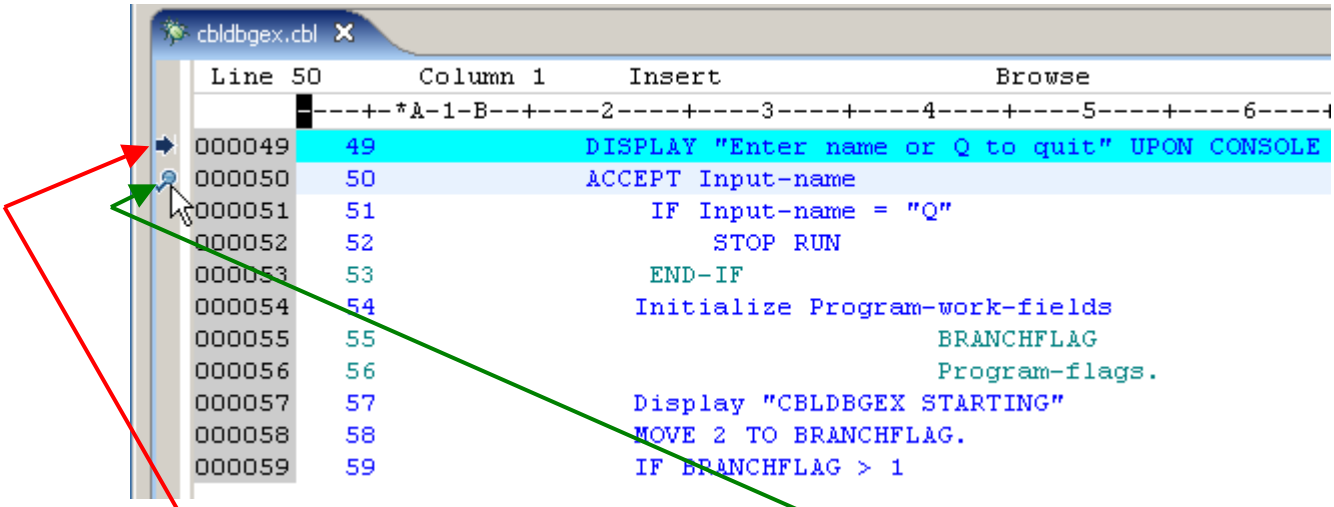


Abb. 5.2.3

Dieser kleine Pfeil zeigt an, dass die Programmausführung nach der Ausführung dieser Zeile gestoppt wurde.

Fügen Sie einen Breakpoint in das ACCEPT Statement ein. Hierzu bewegen Sie den Mauszeiger auf den grauen Bereich vor der Zeilennummer, und klicken Sie zwei mal. Ein kleiner Kreis zeigt einen Haltepunkt (breakpoint) in Zeile Nr. 000050 an.

Ein Breakpoint bewirkt, dass die Ausführung eines Programms an der Stelle unterbrochen wird, wo der Haltepunkt gesetzt wurde.

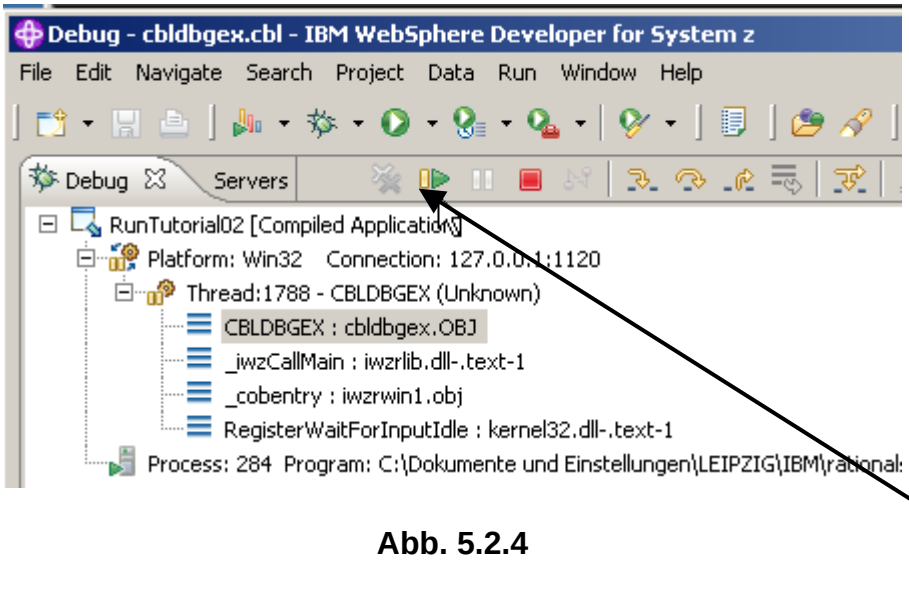


Abb. 5.2.4

Restarten Sie das Programm bis der Breakpoint gefunden wird. Hierzu klick auf das Resume icon ( ), oder drücke F8.

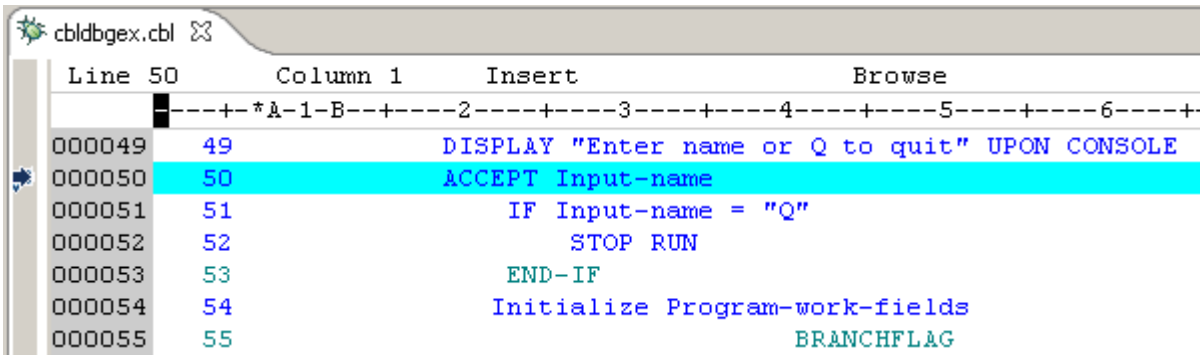


Abb. 5.2.5

Die Programm-Ausführung wird an der ACCEPT-Anweisung in Zeile 000050 stoppen.

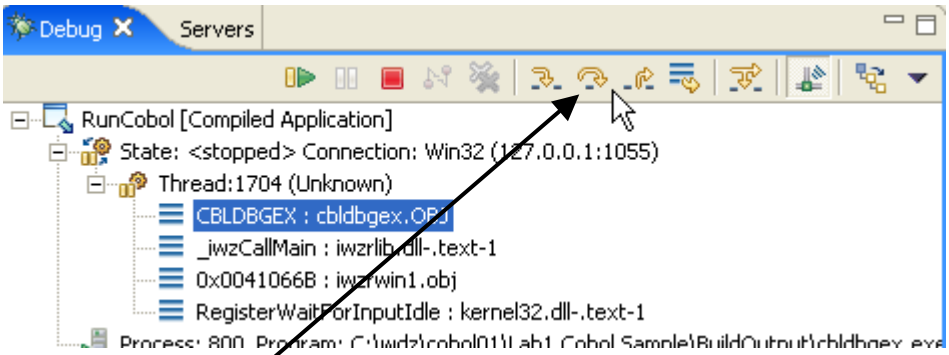


Abb. 5.2.6

Mit dem Step Over icon (oder F6) ein Statement ausführen..

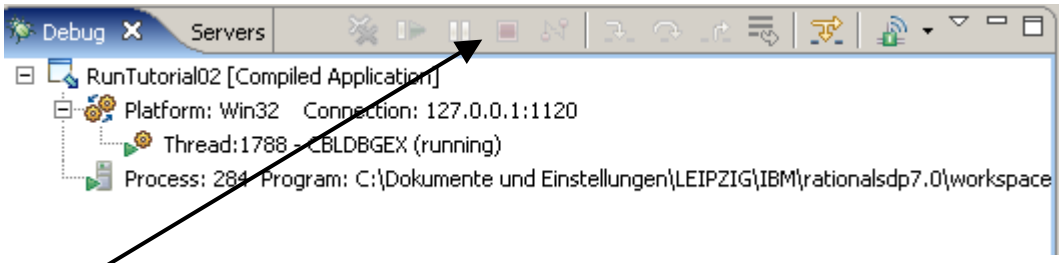


Abb. 5.2.7

Die Icons sind nun deaktiviert (die Symbole sind hellgrau), da Sie zu dem Konsole Fenster gehen müssen, um eine Antwort einzugeben. Wir warten auf die Eingabe in der Konsole.

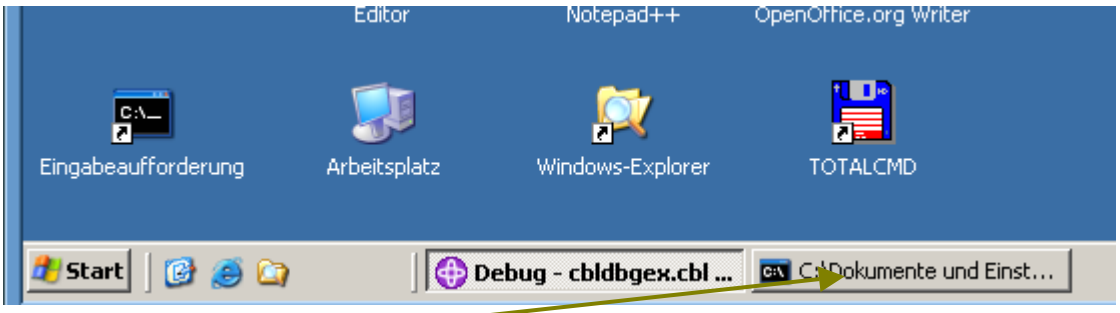


Abb. 5.2.8

A Konsole Window erscheint. Es kann minimiert oder versteckt unter dem RDz Window sein. Klicken Sie auf die Registrierkarte, um es zu öffnen.

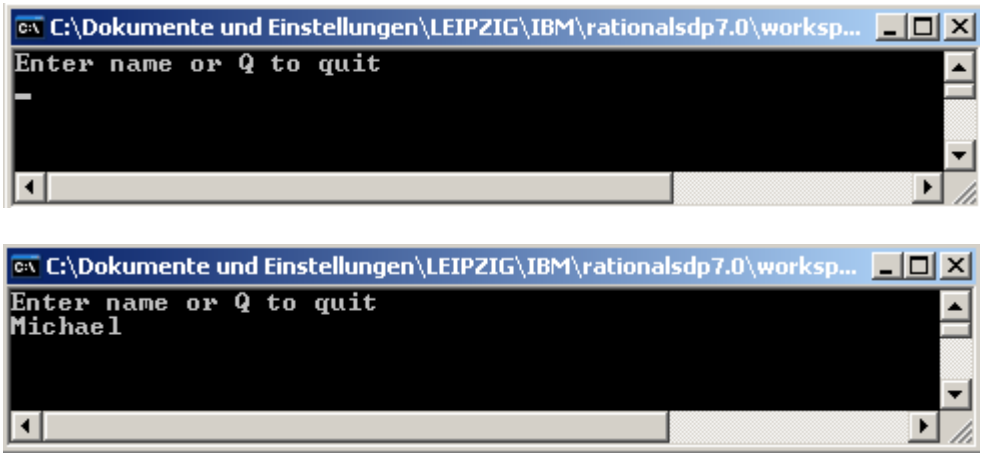


Abb. 5.2.9

Stellen Sie das Konsole-Fenster wieder her, geben Sie einen Namen ein und drücken Sie Enter, und minimieren Sie das Konsole-Fenster (nicht schließen), um mit debug fortzufahren.

5.3 Variablen Werte

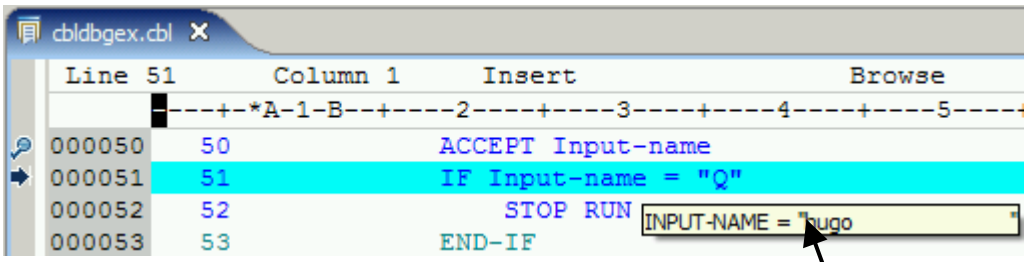


Abb. 5.3.1

Um den Wert einer Variablen zu überwachen (z.B. die Variable INPUT-NAME), bewegen Sie den Maus Zeiger auf den Data Namen, und warten Sie 5 – 10 Sekunden. Der Wert wird wie hier gezeigt wiedergegeben.

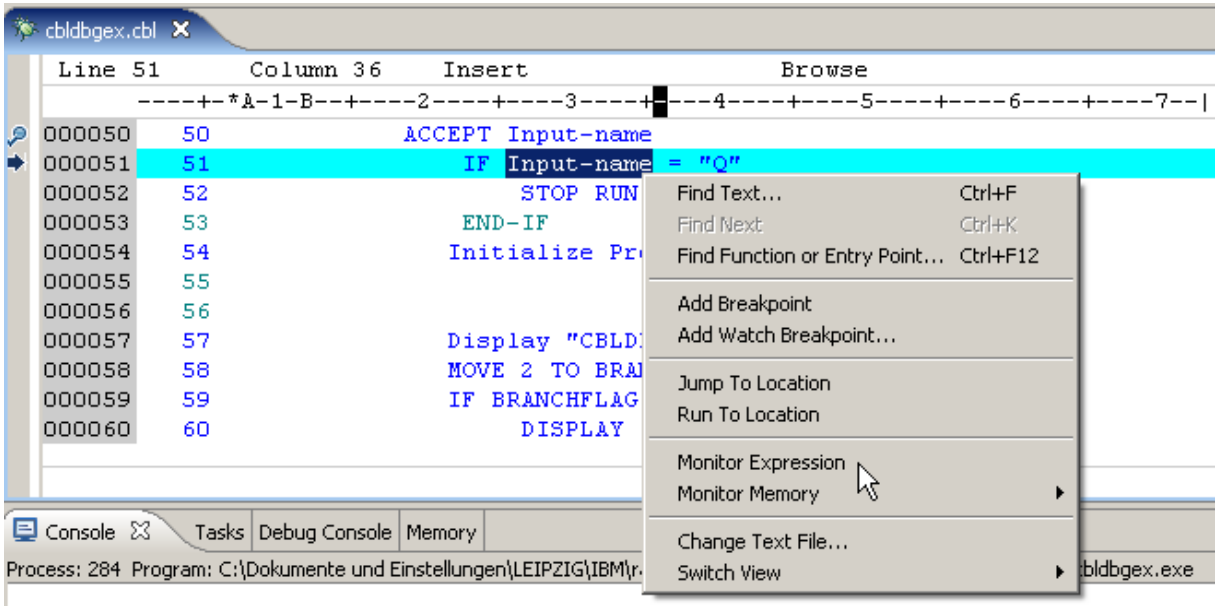


Abb. 5.3.2

Sie können das Feld “Input-name” selektieren, 1kr. Im Kontext menu click “Monitor Expression”.

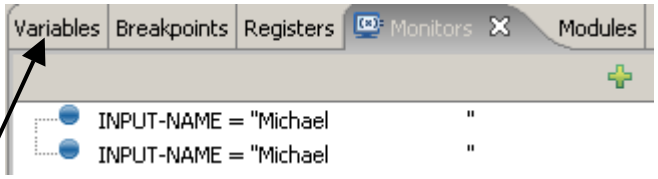


Abb. 5.3.3

In der rechts oberen Ecke wird der Wert der Variablen wiedergegeben. (Wenn Sie dies 2 mal machen, wird der Wert 2 mal wiedergegeben).

1k auf den “Variables” Tab um zum Fenster zurückzukehren.

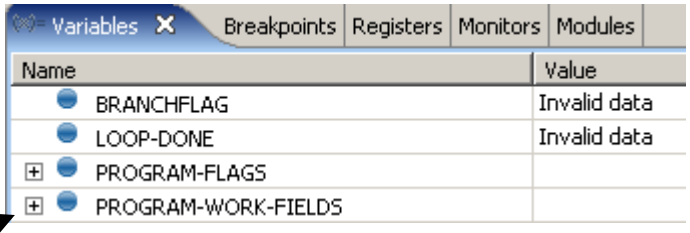


Abb. 5.3.4

Expandiere Program-Work-Fields

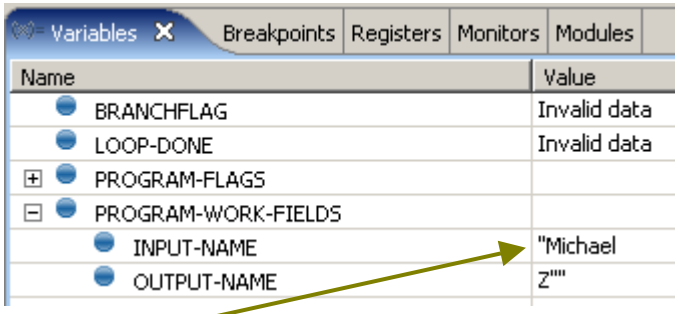


Abb. 5.3.5

Der Wert der Variablen wird dargestellt.

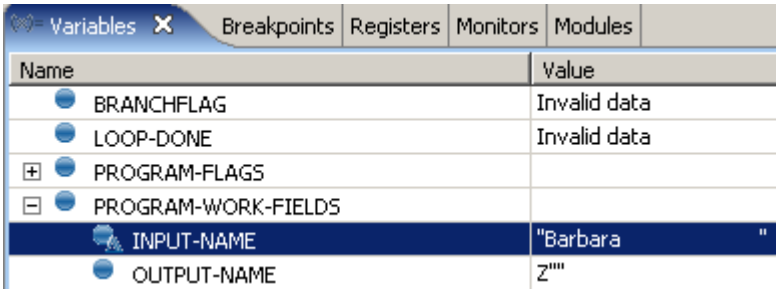


Abb. 5.3.6

Klick auf Input –Name und überschreibe den Wert mit “Barbara”. Wenn Sie jetzt den Mauszeiger auf das Input-Field in Zeile 000051 bewegen, wird der geänderte Wert angegeben.

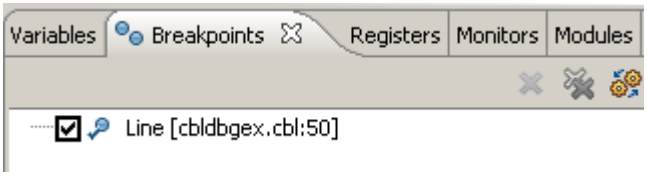


Abb. 5.3.7

In der rechts oberen Ecke, 1k auf den Breakpoints Tab um alle Breakpoints in dem Programm zu sehen.

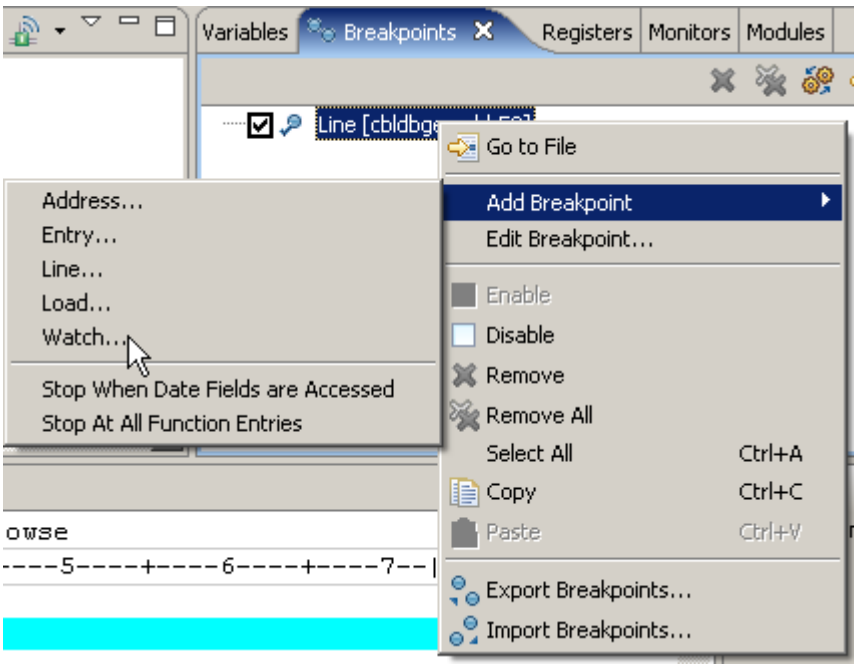


Abb. 5.3.8

Wir fügen nun einen Watchpoint hinzu.. 1kr auf den Breakpoint View. Selektiere Add Breakpoint → Watch…, 1k

5.4 Watchpoint

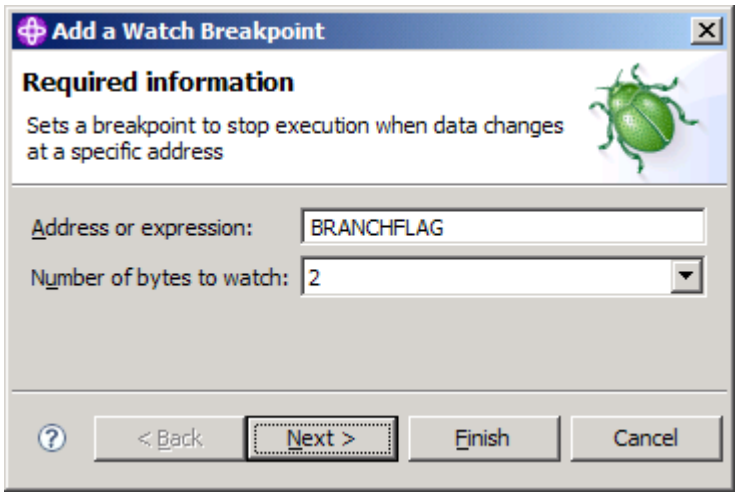


Abb. 5.4.1

BRANCHFLAG in das “Address or expression” Feld eingeben, sowie eine “2” in “Number of bytes to watch”. Wir wünschen einen Stop an diesem Feld, wenn es geändert wird.
1k on Next.

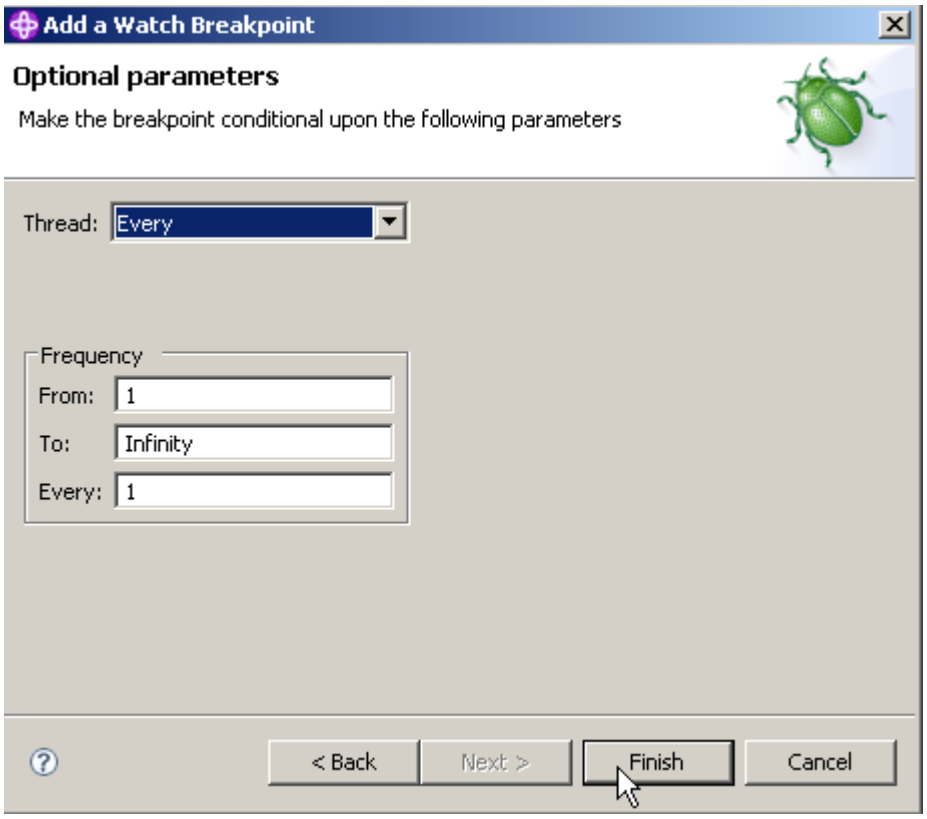


Abb. 5.4.2

Die Default Werte übernehmen, 1k auf Finish.

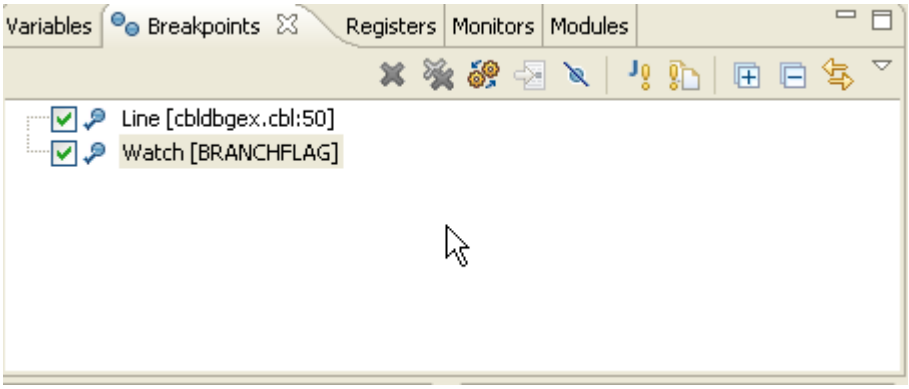


Abb. 5.4.3

Entfernen Sie den Breakpoint (an Stelle von entfernen könnten Sie ihn auch disablen). Öffnen Sie den Breakpoints View, 1kr innerhalb des Fensters, und …

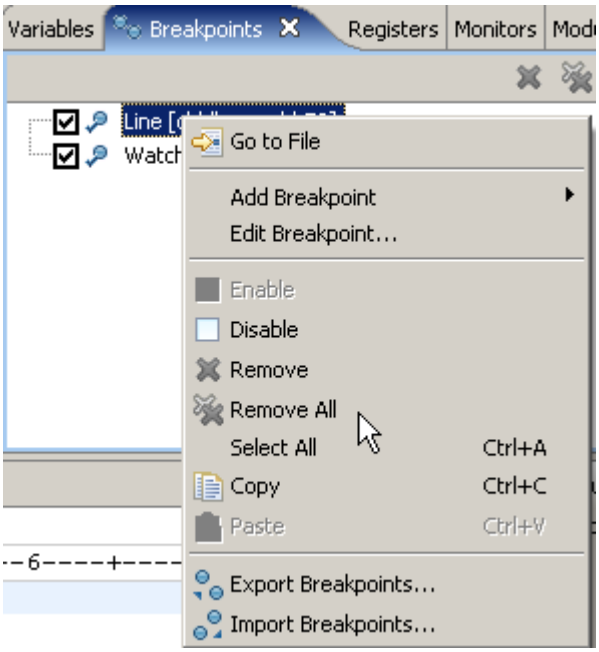


Abb. 5.4.4

selektiere “ Remove All” von dem Kontext Menu (1kr)

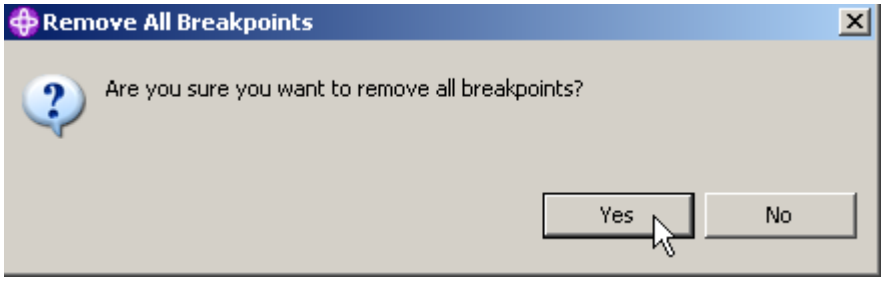


Abb. 5.4.5

Dies löscht alle Breakpoints.

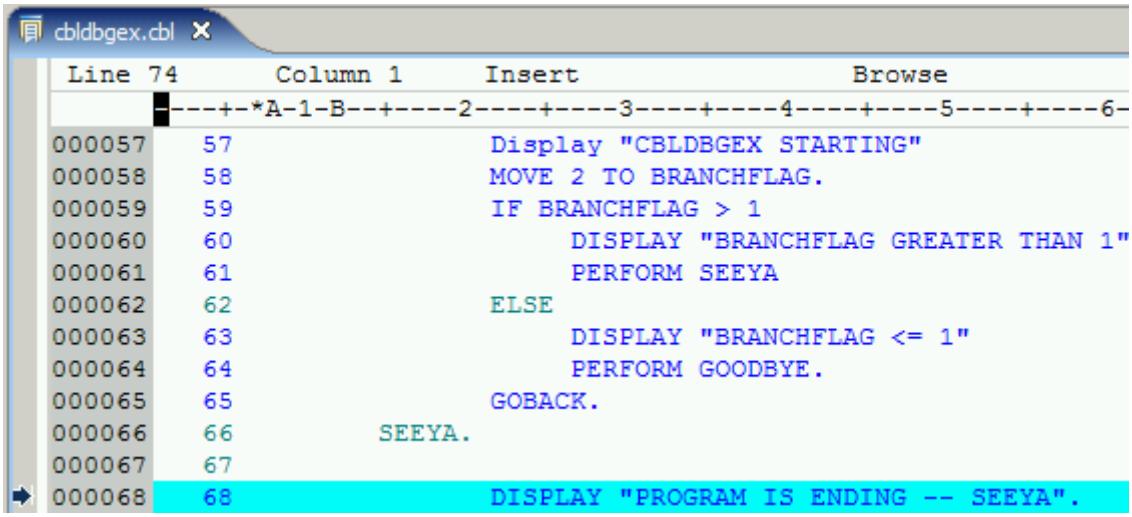


Abb. 5.4.6

In dem Debug Window, klicken Sie wiederholt Step Over (F6) bis Sie Zeile 68 erreichen:

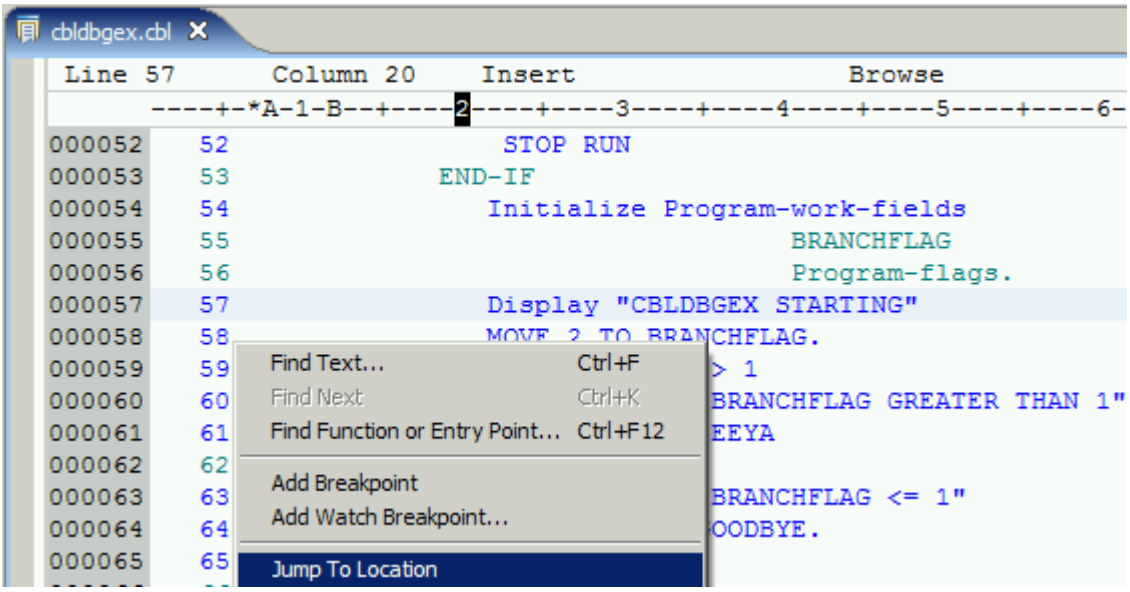


Abb. 5.4.7

Scroll zurück zu Zeile 58. Im Kontext Menu selektieren Sie “Jump To Location”. Dies bewirkt dass die Programm Ausführung zurück zu Zeile 58 bewegt wird.

Spielen Sie mit Debug, um vertrauter zu werden.

Wenn Sie das Programm weiter ausführen wollen, benutzen sie den Resume Icon ( ).

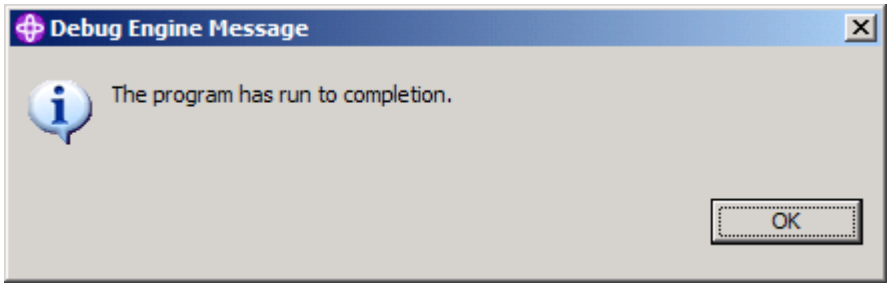


Abb. 5.4.8

Eine Nachricht zeigt das Programm ende an.

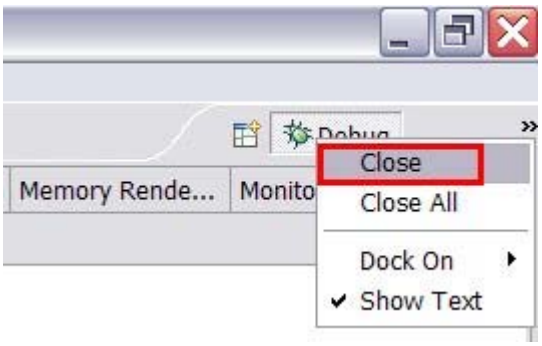


Abb. 5.4.9

Click OK um die Debug Perspective zu schließen (Upper Right).

Selbst-Test

- Wodurch unterscheiden sich die RDz Test/Debug Funktionen von denen, die unter TSO und ISPF verfügbar sind ? Es gibt zahlreiche Personen, die lieber mit TSO als mit RDz arbeiten.

Und das war es

Sie können die gleichen Debug Funktionen für CICS, IMS oder DB2 Programme benutzen, welche unter z/OS laufen. Hierzu ist es erforderlich, den Interactive Debugger auf diesen Platformen zu installieren.

Herzlichen Glückwunsch, Sie haben Tutorial 02 erfolgreich abgeschlossen. Bis hierhin mag das alles interessant sein, erläutert aber noch nicht das Schreiben von Cobol Programmen unter z/OS. Wir werden dies in dem nächsten Tutorial nachholen.

Ende