CTG Tutorial
CICS Transaction Gateway

Die ursprüngliche Version dieses Tutorials entstand Nov. 2004 aus einer Diplomarbeit von Herrn Gerrit Schlüter. Erweitert mit der Primzahltestanwendung von Hanno Eichelberger und Rouven Walter, Mai 2012.

Übersicht

1. Aufgabenstellung
2. Konfiguration Details
3. Voraussetzungen
4. Vorgehensweise
5. Erstellung des CICS-Programms
- **5.1 Das Cobol Programm**
- **5.2 COMMAREA**
- **5.3 Übersetzen des Cobol Programms**
- **5.4 Installation des übersetzten Cobol Programms**

6. Test der Konfiguration
7. Detail Information

1. Aufgabenstellung

In der vorliegenden Übungsaufgabe werden wir eine transaktionale Anwendung entwickeln, die aus Präsentations-Logik und Business-Logik besteht. Über den Browser wird eine Zahl eingegeben, und die Anwendung ermittelt, ob es sich dabei um eine Primzahl handelt oder nicht.

Wir implementieren die Business-Logik als ein CICS-Cobol-Programm und die Präsentations-Logik als ein Java-Servlet, welches in einem WebSphere-Servlet Container läuft.

Von besonderem Interesse ist die Art der Verbindung zwischen Präsentations-Logik und Business-Logik. Wir benutzen hierfür den CICS-Intersystem-Communication-Mechanismus, welcher die ECI- oder EXCI-Schnittstelle anbietet. Um diese Verbindung nicht zu Fuß zu programmieren, verwenden wir hierfür ein vorhandenes Java-Paket, das CICS-Transaction-Gateway (CTG).

Das CICS-Transaction-Gateway ist ein sog. Connector, der die Java-Connection-Architectur (JCA) implementiert. Es existieren viele derartiger Connectoren, z. B. für die Verbindung einer Java-Präsentations-Logik mit einer Oracle-, IMS-, DB2-Stored-Procedure oder SAP-R/3-Business-Logik.

Unsere Test-Konfiguration besteht neben einem Browser auf Ihrem PC aus einem WebSphere-Web-Application-Server (WAS) und einem CICS-Transaktionsserver, wie in Abbildung 1 wiedergegeben.

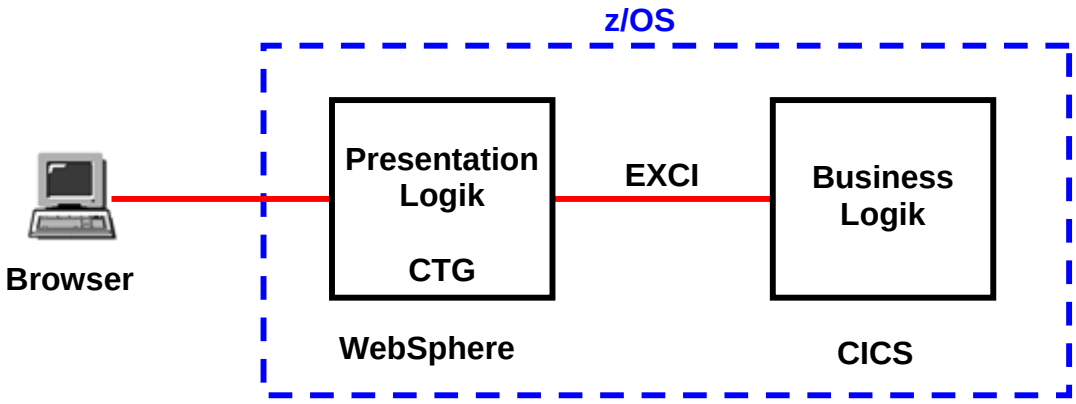

Abbildung 1: Allgemeiner Aufbau

Für eine derartige Konfiguration bestehen zwei Implementierungsalternativen. Eine integrierte Alternative ist, WebSphere und CICS in zwei getrennten Adressenräumen auf dem gleichen z/OS-System zu betreiben. In diesem Fall kann die Kommunikation zwischen beiden über eine vom z/OS-Kernel verwaltete Hauptspeicher-Queue unter Benutzung der EXCI-Schnittstelle erfolgen. Die andere „Distributed"-Alternative ist, WebSphere auf einem getrennten Linux- oder Windows-Server zu betreiben und über TCP/IP an z/OS und CICS anzuschließen. In diesem Fall benutzen wir die ECI-Schnittstelle.

Die vorliegende Übungsaufgabe benutzt die integrierte Alternative.

Selbst-Test

- **Ist das ECI-Protokoll eine CICS spezifische Implementierung des ECI-Protokolls?**
- **Würden wir auch ECI bzw. EXCI benutzen, wenn die Business-Logik genauso wie die Präsentations-Logik in Java implementiert wäre und auf einem Web-Application-Server laufen würde?**

Frage: ***Würden wir auch ECI bzw. EXCI benutzen, wenn die Business-Logik genauso wie die Präsentations-Logik in Java implementiert wäre und auf einem Web-Application-Server laufen würde?*** **Antwort:** ***Nein, wir würden RMI in diesem Fall verwenden. ECI und EXCI sind spezifisch die Schnittstellen, die für eine Kommunikation mit CICS verwendet werden.***

2. Konfiguration Details

Die in dieser Übungsaufgabe verwendete Anwendung ist recht anspruchslos: Über den Browser geben wir eine Ziffer ein, die Business-Logik ermittelt, ob es sich dabei um eine Primzahl handelt und die Präsentations-Logik teilt dem Browser das Ergebnis mit.

Das CICS-Cobol-Programm, welches die Business-Logik implementiert, erhält einen Input über die COMMAREA, ermittelt ein Ergebnis (ja/nein) und stellt es wieder in seine COMMAREA.

Die Präsentations-Logik besteht aus drei Java Server Pages (jsp), einem Servlet und einer Enterprise Java Bean (EJB). Die Java Server Page main.jsp nimmt vom Browser eine Eingabe (eine Ziffer in diesem Fall) entgegen. Eine zweite Java Server Page results.jsp liefert das Ergebnis an den Browser (Im Fehlerfall wird die Java Server Page error.jsp aufgerufen).

Die erste Java Server Page ruft über ein Form-Tag ein Servlet PrimeCTGServlet auf, welches die Präsentations-Logik enthält. Dieses benutzt eine Enterprise Java Bean CTGtesterCCIBean für die Kommunikation mit der CICS-COMMAREA mittels der EXCI-Schnittstelle. Wir verwenden EXCI an Stelle von ECI, weil WebSphere und CICS in unserem Fall unter dem gleichen z/OS-Kernel laufen. CTGtesterCCIBean enthält den CICS-Transaction-Gateway (CTG)-Code für die Kommunikation mit der CICS-COMMAREA.

Diese Zusammenhänge sind in Abb. 2 dargestellt.

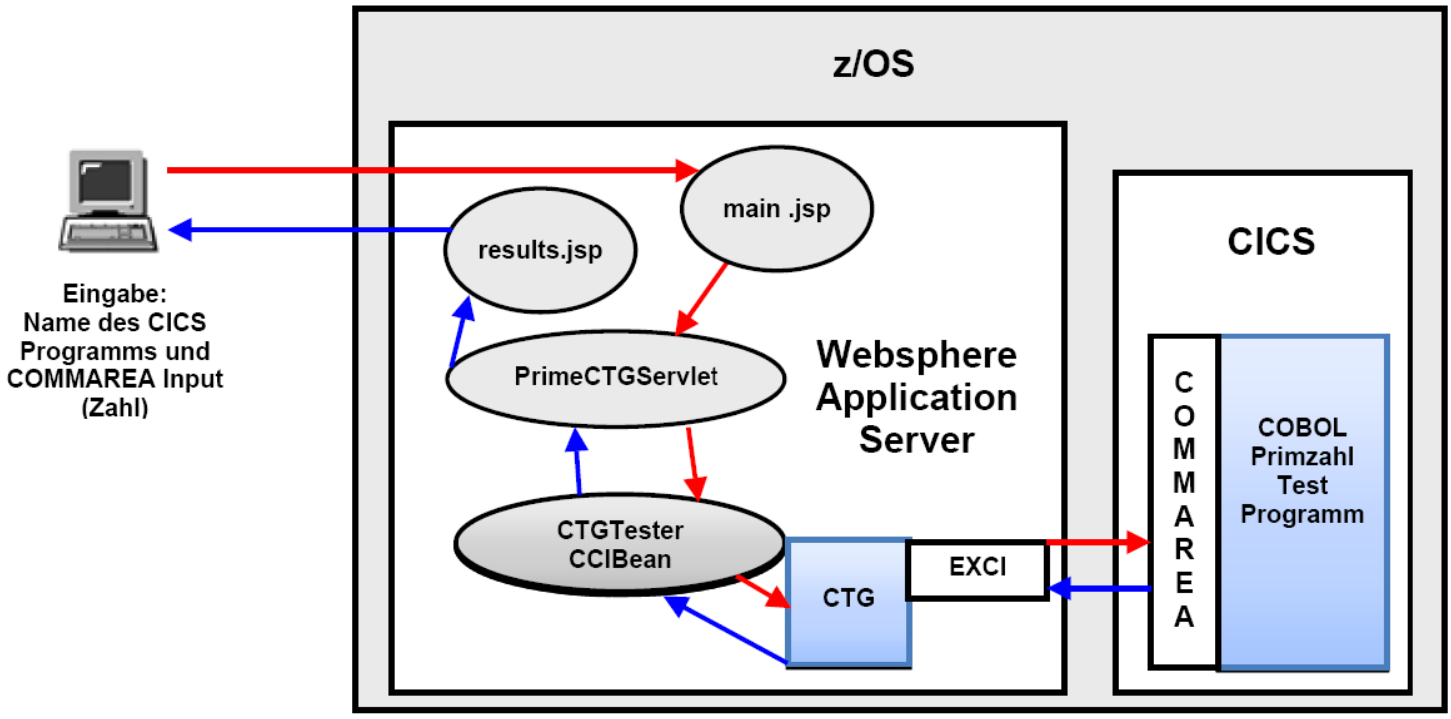

Abbildung 2: Überblick Primzahltester mit WAS, CTG und CICS

Selbst-Test

- **Läuft das CTG innerhalb des WAS-EJB-Containers?**

3. Voraussetzungen

Dieses Tutorial ist für den Tübinger Mainframe "Hobbit" (134.2.205.54) vorgesehen. An vielen Stellen taucht im Tutorial Text die Nummer "218" auf. Bitte ersetzen Sie diese durch Ihre eigene Praktikums ID. Angenommen Ihre ID ist prak238, dann ersetzen Sie zum Beispiel "PRAK218" durch "PRAK238" oder "CTG218" durch "CTG238".

4. Vorgehensweise

Zuerst erstellen und installieren wir ein CICS Cobol-Programm namens "CTG218", welches über die CICS-COMMAREA (Communications Area) eine Zahl übertragen bekommt und diese auf die Primzahleigenschaft testet. Das Ergebnis der Berechnung schreibt das Programm zurück in die COMMAREA.

Die Funktionsfähigkeit des erstellten und installierten CICS-Programms wird danach mit einem Browser und einem speziellen Servlet getestet. Der Anwender kann in einer HTML-Oberfläche eine Zahl eingeben, welche über das Servlet, eine Enterprise Bean und die COMMAREA zum CICS-Programm übertragen wird. Das Ergebnis des Primzahltests wird über denselben Weg zurückübertragen und im Browser angezeigt (siehe Abbildung 2).

CICS und WebSphere sind auf Hobbit bereits vorinstalliert. Zur Erleichterung haben wir das Servlet PrimeCTGServlet, die EJB CTGtesterCCIBean sowie die drei .jsp Pages main.jsp, results.jsp und error.jsp unter WebSphere bereits vorinstalliert; Sie brauchen also nur noch aufgerufen zu werden.

5. Erstellung des CICS-Programms

Zuerst soll das CICS Cobol Programm erstellt werden, auf welches das CICS Transaction Gateway später zugreifen soll. Der Unterschied zu den bisherigen CICS-Programmen aus der Aufgabe „Cobol Hello World unter CICS“ und „Datenbankzugriff mit CICS“ besteht darin, dass die Ausgabe nicht in Form von BMS-Maps auf dem Bildschirm erscheint. Stattdessen wird die Ausgabe in die CICS COMMAREA (Communications Area) geschrieben, die zum Datenaustausch zwischen den CICS-Programm und dem CICS Transaction Gateway verwendet wird (Abbildung 2).

Um das CICS-Programm zu erstellen, arbeiten wir sowohl mit dem TSO- als auch mit dem CICS-Subsystem. Wir verwenden am Anfang TSO, so dass wir mit dem Befehl "L TSO" eine TSO-Sitzung starten (Abbildung 2) und uns mit unseren Benutzerdaten anmelden.

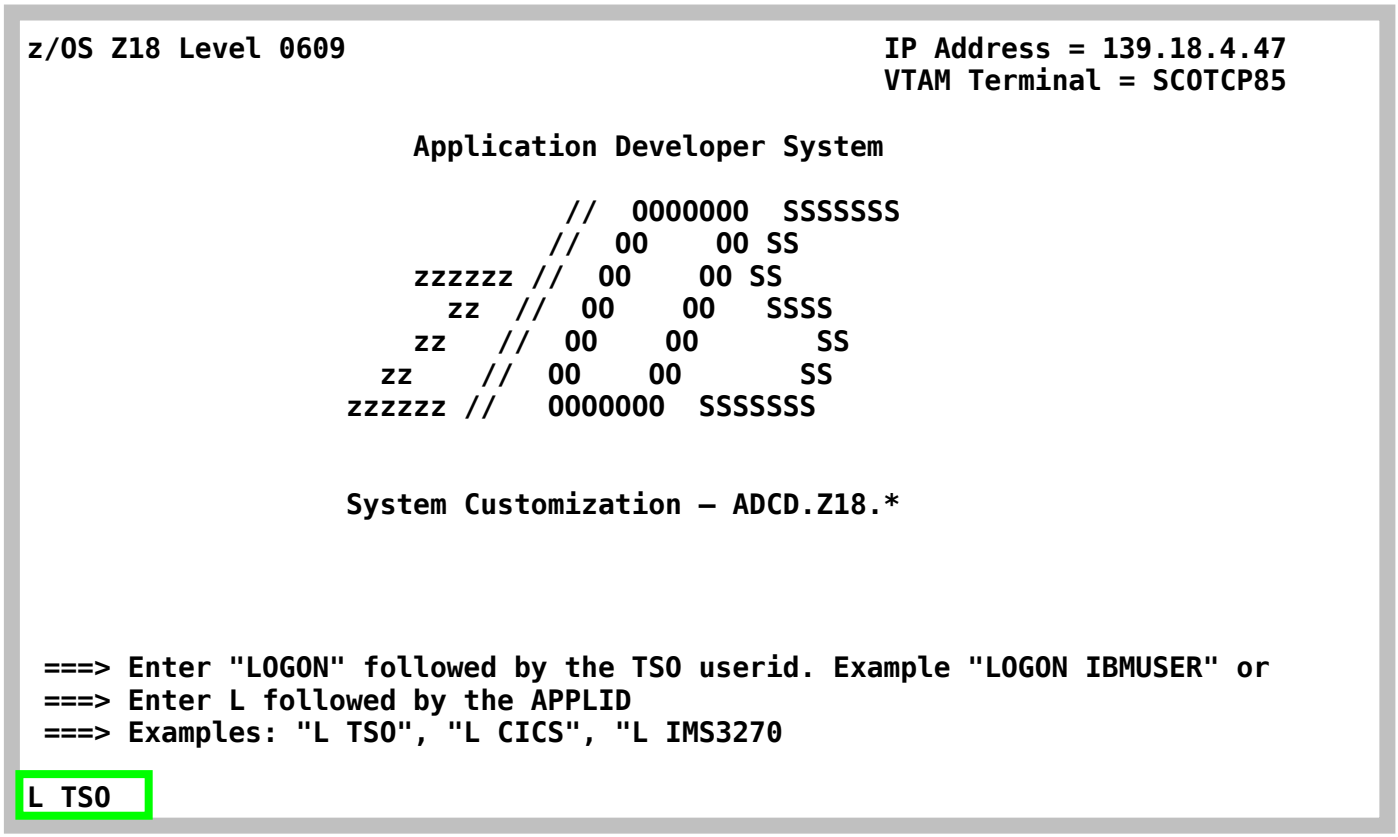

Abbildung 3: Start der TSO-Sitzung

Zuerst muss noch ein Dataset erstellt werden. Wir öffnen den "Data Set Utility" Screen und erstellen (Allocate) einen neuen Partitioned Dataset namens PRAK218.CICS.CTG. Dabei verwenden wir die in der nachstehenden Aufgabe angegebenen Parameter.

Aufgabe: ***Legen Sie den Dataset "PRAK218.CICS.CTG" an. Verwenden Sie dazu folgende Parameter:***

Space units ..... KB	***Record format .... FB***
Primary quantity .. 16	***Record length 80***
Secondary quantity 1	***Block size 320***
Directory blocks .. 2	***Data set name type: PDS***

Unsere Anwendung besteht aus einem JCL-Script, welches ein Cobol-Programm enthält. Das JCL-Script dient zur Übersetzung des Cobol-Programms. Wir erstellen dieses im neuen Partitioned Dataset "PRAK218.CICS.CTG".

In diesem Fall enthält das CICS-Programm lediglich die Businesslogik. Die Präsentationslogik wird später vom WebSphere-Application-Server in Form einer Webanwendung bereitgestellt.

5.1 Das Cobol Programm

Wir beginnen mit dem Quellcode des CICS-Cobol-Programms und editieren ein Member "CTG218" für den neu angelegten Dataset "PRAK218.CICS.CTG" (siehe Abbildung 4 und 5).

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT       PRAK218.CICS.CTG(CTG218) - 01.18               Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100 //PRAK218C JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
000200 //CALLPROC EXEC PROC=DFHYITVL,
000300 // PROGLIB=DFH310.CICS.PRAKLOAD
000400 //*
000500 //TRN.SYSIN DD *
000600        IDENTIFICATION DIVISION.
000700        PROGRAM-ID. CTG218.
000800        ENVIRONMENT DIVISION.
000900        DATA DIVISION.
000901
000910       *DEKLARATION VON VERWENDETEN VARIABLEN
001000        WORKING-STORAGE SECTION.
001100        01 NAME-TAB.
001200           02 PRIME       PICTURE X(20) VALUE "PRIME".
001210           02 NPRIME      PICTURE X(20) VALUE "NPRIME".
001301           02 B           PICTURE 9(7).
001302           02 C           PICTURE 9(7).
001303           02 I           PICTURE 9(7).
001304           02 D           PICTURE 9(7).
001305
001306       *VERKNUEPFUNG ZUR COMMAREA UEBER VARIABLE NR
001310        01 COMMAREA-IN.
001320           03 NR          PICTURE X(7).
001400        LINKAGE SECTION.
001500        01 DFHCOMMAREA.
001600           03 RES         PICTURE X(20).
001700
001710       *DAS PROGRAMM
001800        PROCEDURE DIVISION.
001801
001802       *    KOPIERE INHALT DER COMMAREA IN VARIABLE NR
001810            MOVE DFHCOMMAREA TO COMMAREA-IN.
001900            MOVE SPACES TO DFHCOMMAREA.
001901
Command ===>                                                 Scroll ===> PAGE
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Abbildung 4: Quelltext des CICS-Programms, Panel 1/2

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       PRAK218.CICS.CTG(CTG218) - 01.18               Columns 00001 00072
 001902
 001903       *    KONVERTIEREN DES COMMAREATEXT IN NUMMERISCHE ZAHL
 001904            COMPUTE B = FUNCTION NUMVAL(NR).
 001905
 001906       *    TESTEN OB ZAHL = 1
 001907            IF B = 1 PERFORM STOP-NPRIM.
 001908
 001909       *    ZAHL DURCH ALLE KLEINEREN ZAHLEN TEILEN
 001910       *    UND CHECKEN OB ERGEBNIS GERADE
 001911            DIVIDE B BY 2 GIVING C ROUNDED.
 001912            PERFORM CHECK-PARA VARYING I FROM 2 BY 1 UNTIL I > C.
 001913
 001914            MOVE PRIME TO RES.
 001915            EXEC CICS RETURN
 001916                    END-EXEC.
 001917            EXIT.
 001918
 001919       *PROC WIRD AUFGERUFEN WENN ZAHL NICHT PRIMZAHL IST
 001930        STOP-NPRIM.
 001940            MOVE NPRIME TO RES.
 001950            EXEC CICS RETURN
 001960                    END-EXEC.
 001970            EXIT.
 001971
 001972       *PROC TEILT ZAHL DURCH TEILER UND UEBERPRUEFT REST
 001980        CHECK-PARA.
 001990            DIVIDE B BY I GIVING C REMAINDER D.
 002000            IF D = 0 PERFORM STOP-NPRIM.
 002500 /*
 002600 //LKED.SYSIN DD *
 002700   NAME CTG218(R)
 002800 /*
 002900 //
****** **************************** Bottom of Data ****************************

 Command ===>                                                Scroll ===> PAGE
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Abbildung 5: Quelltext des CICS-Programms, Panel 2/2

Das JCL-Script ruft die Prozedur "DFHYITVL" auf, die alle Übersetzungsschritte für das CICS-Programm enthält: CICS-Übersetzer, Cobol-Compiler und Linker.

5.2 COMMAREA

Das aufrufende Programm (Java-Presentation-Logik) übergibt seine Eingabe-Daten an die COMMAREA des aufgerufenen Cobol-Programms. Das Cobol-Programm gibt sein Ergebnis über die COMMAREA an das aufrufende Präsentationslogik-Programm zurück. Das CTG implementiert die Verbindung zwischen Präsentationslogik und Business-Logik.

Im Cobol-Quelltext wird die COMMAREA durch die "DFHCOMMAREA" Angabe in der "LINKAGE SECTION" in Zeile 001500 definiert. Sie können beliebige Namen in der Working-Storage-Section benutzen, aber der Name "DFHCOMMAREA" in der Linkage-Section ist Pflicht. DFHCOMMAREA dient zum Datenaustausch zwischen dem CICS-Programm und dem CICS-Transaction-Gateway. In unserem Fall enthält die COMMAREA nur ein Feld „RES“ für die zu testende Ziffer.

In der Procedure Division kopiert das Cobol Progamm den Inhalt der COMMAREA in die Variable COMMAREA-IN (Zeile 001810), berechnet seine Antwort, und kopiert das Ergebnis zurück in die COMMAREA in das Feld „RES“ (Zeile 001914 bzw. Zeile 001940). Zeile 001900 löscht vorher den Inhalt von DFHCOMMAREA.

Hierzu muss die COMMAREA-Struktur sowohl in der Working-Storage-Section, als auch in der Linkage-Section des Cobol-Programms deklariert werden. In der Linkage-Section muss die Definition als erste Deklaration erscheinen.

Bemerkung: Source-Code-Kommentare werden in z/OS COBOL mit einem „*“ oder „/“ in Spalte 7 definiert.

Die COMMAREA-Programmierung ist recht einfach zu erstellen. Ein Nachteil ist, dass die COMMAREA in der Größe auf 32 KByte begrenzt ist. Als Alternative kann eine als „Channels and Containers“ bezeichnete Lösung eingesetzt werden. Ein Tutorial hierüber ist zu finden unter http://docweb.cns.ufl.edu/news/n0510/Channels_Containers.pdf.

Selbst-Test

- **Warum wird die COMMAREA sowohl in der Working-Storage-Section als auch in der Linkage-Section definiert?**

Aufgabe: *Geben Sie das COBOL-Programm ein und übersetzen Sie es anschließend mit dem Befehl "SUB" in der Kommandozeile.*

5.3 Übersetzung des Cobol Programms

```
14.11.21 JOB07408 $HASP165 PRAK218C ENDED AT N1  MAXCC=0 CN(INTERNAL)
***
```

Abbildung 6: Bestätigung der Jobverarbeitung

"MAXCC=0" heißt, dass der Job erfolgreich ausgeführt wurde. Das CICS-Programm wurde nun übersetzt und liegt in ausführbarer Form vor.

Nun muss das übersetzte Programm in CICS eingebunden (definiert und installiert) werden. Dazu loggen wird uns im Anmeldebildschirm mittels "L CICS" ein {Abbildung 7).

5.4 Installation des übersetzten Cobol Programms in CICS

```
z/OS Z18 Level 0609                                IP Address = 139.18.4.47
                                                   VTAM Terminal = SCOTCP85

                     Application Developer System

                              //  OOOOOOO  SSSSSSS
                             //  OO    OO SS
                     zzzzzz //  OO    OO SS
                       zz  //  OO    OO   SSSS
                     zz   //  OO    OO       SS
                   zz    //  OO    OO       SS
                 zzzzzz //   OOOOOOO  SSSSSSS


                 System Customization - ADCD.Z18.*




 ===> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
 ===> Enter L followed by the APPLID
 ===> Examples: "L TSO", "L CICS", "L IMS3270

L CICS
```

Abbildung 7: Einloggen ins CICS

Nachdem wir unter unserem Mainframe-Login eingeloggt sind, löschen wir noch durch Betätigung der (eventuell emulierten) CLEAR-Taste das CICS-Fenster. Nun steht der Cursor in der linken oberen Ecke und wartet auf die Eingabe einer CICS-Transaktion.

Eine evtl. bereits vorhandene CICS-Group „PRAK218“ muss mit Hilfe von „CEDA DELETE ALL GROUP(PRAK218)“ gelöscht werden.

Wir definieren das Programm mit Hilfe des Befehls "CEDA DEFINE PROGRAM(CTG218) GROUP(PRAK218)" (siehe Abbildung 8).

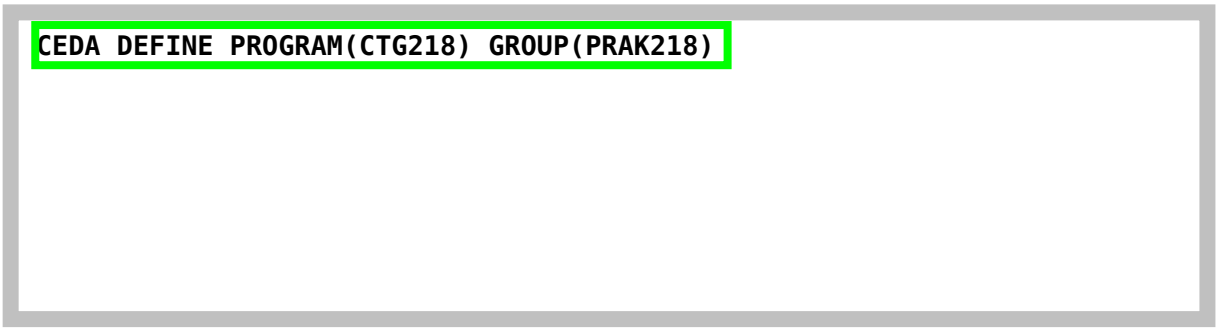

Abbildung 8: Definition des Programms

CICS möchte von uns nun einige Parameter für das Programm wissen. Wir ändern lediglich die Sprache in "Le370" und bestätigen mit der Eingabetaste (Abbildung 9)

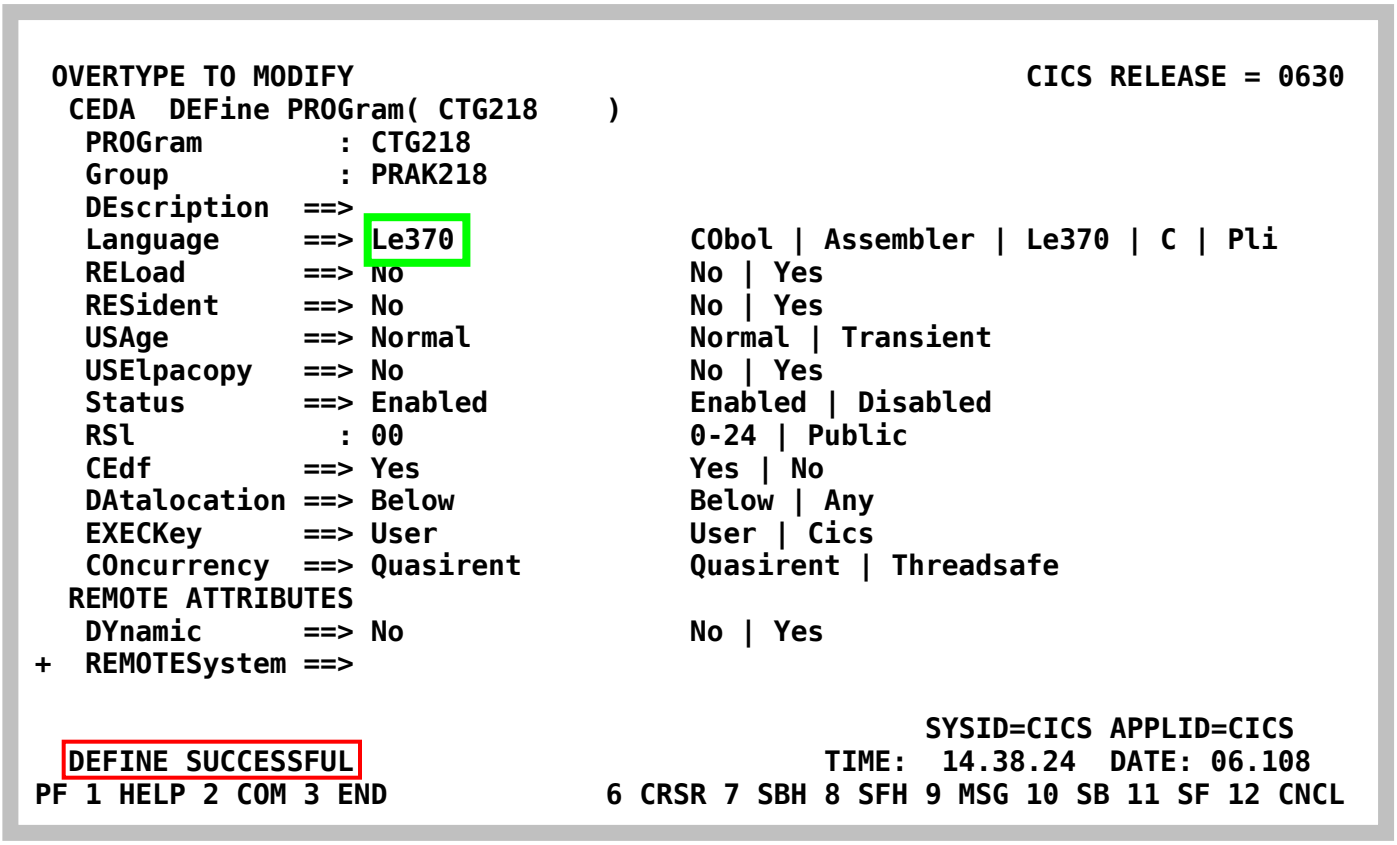

Abbildung 9: Parameter des CICS-Programms

Wir installieren daraufhin die Gruppe "PRAK218" neu, indem wir den CEDA-Befehl "CEDA INSTALL GROUP(PRAK218)" eingeben (Abbbildung 10).

.

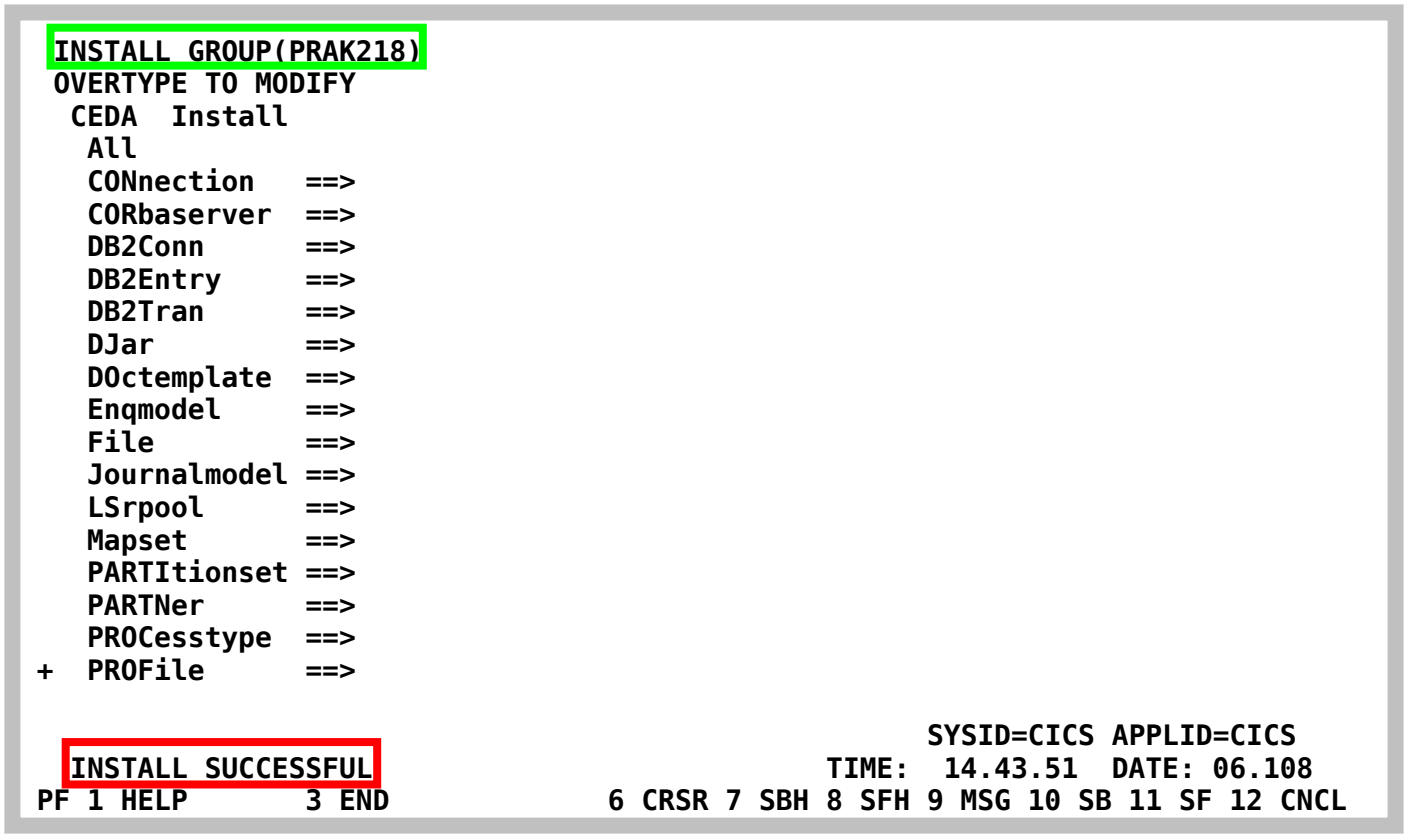

Abbildung 8: Installation der Gruppe

6. Test der Konfiguration

Nachdem nun das CICS-Primzahltestprogramm compiliert und in CICS definiert und installiert wurde, möchten wir den Aufruf von diesem über das CICS-Transaction-Gateway testen. Der Test erfolgt in einem normalen Webbrowser Ihrer Wahl.

Wir haben die beiden Java Server Pages main.jsp und results.jsp sowie das Servlet PrimeCTGServlet zu einem Web-Archiv (.war) zusammengefasst und in WebSphere im Verzeichnis PrimeCTGWeb untergebracht. Nachrichten werden über Port 9527 entgegengenommen. In den Webbrowser ist als Adresse

http://hobbit.cs.uni-tuebingen.de:9527/PrimeCTGWeb/main.jsp

einzugeben. So wird der CTG-Primzahltester "PrimeCTG" gestartet.

Nun müssen den Namen ihres CICS-Primzahltestprogramm und eine zu testende Zahl eingeben. Per Klick auf den Submit-Button wird der Test des Zugriffs auf das CICS-Programm über das CICS Transaction Gateway gestartet (Abbildung 11).

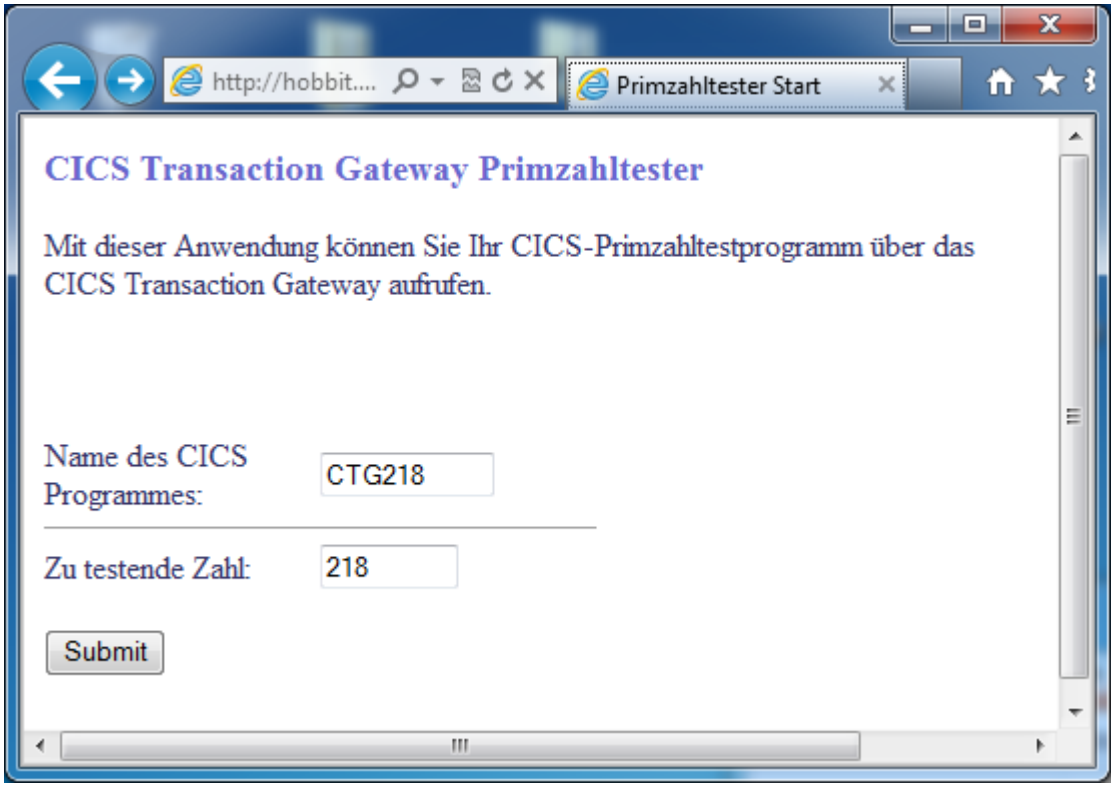

Abbildung 11: Aufruf des Primzahltest über Browser

Als Ergebnis erscheint die Information, ob die Zahl Primzahl ist (PRIME) oder nicht (NPRIME), siehe Abbildung 12.

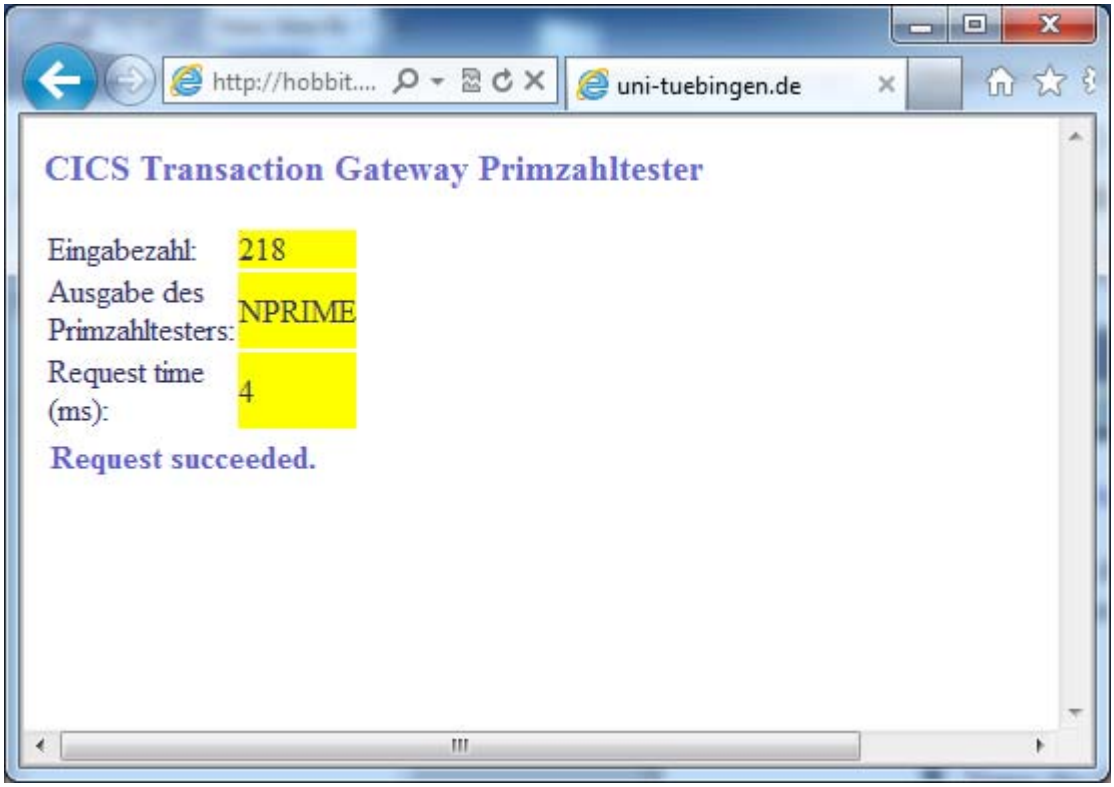

Abbildung 12: Ergebnis des Primzahltests

Aufgabe: *Starten Sie die CTG-Primzahltest-Oberfläche „PrimeCTG".*

Aufgabe: *Geben Sie den Namen Ihres CICS-Primzahltestprogramms und eine zu testende Zahl ein (möglichst den nummerischen Teil Ihrer PRAK-ID). Machen Sie einen Screenshot vom Testergebnis.*

Aufgabe: *Nachdem Ihnen ein erfolgreicher Test gelungen ist, weisen Sie dies bitte anhand eines Screenshots nach (JPEG-Format, max. Größe 99 KB). Dieser soll den gesamten Inhalt des Browser-Fensters zeigen.*

Aufgabe: *Schicken Sie diesen Screenshot im JPEG-Format an Ihren Betreuer.*

Aufgabe: *Loggen Sie sich sauber aus dem CICS aus. Dies geschieht mit dem Befehl "CESF LOGOFF". Details dazu sind in Aufgabe 2, C-Version, beschrieben. Das Schließen des 3270-Emulator-Fensters ohne ein solches sauberes Ausloggen aus CICS ist ebenfalls verboten, weil dies auch CICS zum Absturz bringen könnte, was ein Bearbeiten der CICS-Tutorien für Sie und andere bis auf weiteres unmöglich machen würde.*

7. Detail Information

Die für die Dateneingabe benutzte Java Server Page main.jsp ist in Abb. 11 dargestellt.

```
<FORM ACTION="PrimeCTGServlet" METHOD="POST">
Mit dieser Anwendung können Sie Ihr CICS-Primzahltestprogramm über das CICS
Transaction Gateway aufrufen.<P></P><TABLE border="0" cellpadding="0"
cellspacing="0">
        <TBODY>
                <TR>
                        <TD width="138">Name des CICS Programmes:</TD>
                        <TD width="138"><INPUT type="TEXT" name="funcName" size="10"
                                maxlength="6" value="ECIPROG"></TD>
                </TR>
                <TR>
                        <TD colspan="3">
                        <HR noshade size="1">
                        </TD>
                </TR>

                <TR>
                        <TD width="138">Zu testende Zahl:</TD>
                        <TD colspan="2"><INPUT type="TEXT" name="commareaInput"
                                size="7"
                            value=""></TD>
                </TR>



        </TBODY>
</TABLE><P>
<INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
```

Abbildung 13: Form Tag Teil von main.jsp

Selbst-Test
Schauen sie sich den Code des Form Tags in Abb. 13 an:

- **Welche Variable wird mit dem Namen „funcName“ gekennzeichnet?**
- **Welche Variable wird mit dem Namen „commareaInput“ gekennzeichnet?**

Abbildung 14 zeigt einen Ausschnitt von results.jsp mit den Java Expressions.

```
<TR>
        <TD width="121">Eingabezahl:</TD>
        <TD BGCOLOR=yellow width="97">
                <%= commareaInput %></TD>
</TR>

<TR>
        <TD width="121">Ausgabe des Primzahltesters:</TD>
        <TD BGCOLOR=yellow width="97"><%= results %></TD>
</TR>

<TR>
        <TD width="121">Request time (ms):</TD>
        <TD BGCOLOR=yellow width="97"><%= requestTime %></TD>
</TR>
```

Abbildung 14: Aufruf der Java Expressions in results.jsp

Die Zeichenfolgen <%= und %> schließen eine Java Epression ein

Selbst-Test

- **Wo befindet sich der Code für die drei Java Aufrufe „commareaInput“, „results“ und „requestTime“?**

Abbildung 15 zeigt einen Ausschnitt von error.jsp.

```
<jsp:useBean class="java.lang.String" id="errorException" scope="request"/>

<BODY>
Es ist ein Fehler aufgetreten. Probieren Sie es noch einmal, oder kontaktieren Sie Ihre
Betreuer.

<P>
<TABLE BGCOLOR=red>
<TBODY>
        <TR><TH>Request failed:</TH></TR>
        <TR>
                <TD VALIGN=TOP>Exception occurred:</TD>
                <TD BGCOLOR=YELLOW><%= errorException %></TD>
        </TR>
</TBODY></TABLE></BODY>
```

Abbildung 15: Aufruf der Java Expressions in error.jsp

Abbildungen 16 und 17 zeigen einen Ausschnitt aus PrimeCTGServlet.java .

```
public void processRequest(HttpServletRequest request,HttpServletResponse response)
throws IOException, ServletException {

/**
* Konvertierung der Eingabezahl.
*/
  String number = request.getParameter("commareaInput")+"";
  int intNumber = 1;
  try{
    intNumber = Integer.parseInt(number.replaceAll("^0*",""));
  }catch(Exception e){}
  number = ""+intNumber;
  for (int i=0;i<7-(intNumber+"").length();i++)
    number = "0"+number;

/**
* Laden der Eingabezahl in ECIRequest Input.
*/
  // retrieve HTTP request parameters
  MapRequest2Details(request, eciReq1, "");
  eciReq1.commarea=number;

  ……

/**
* Aufruf der CTGTesterCCIBean mit ECIRequest Input als Parameter.
*/
  Context ic = new InitialContext();
  Object or = ic.lookup("java:comp/env/ejb/CTGTesterCCI");
  CTGTesterCCI tester = null;
  if (or != null) {
     CTGTesterCCIHome home = (CTGTesterCCIHome) PortableRemoteObject.narrow
                            (or,CTGTesterCCIHome.class);
     tester = home.create();
  }
  beforeReqTime = new Date();
  String results = tester.execute(eciReq1);
  …
```

Abbildung 16: Ausschnitt aus PrimeCTGServlet.java
Teil 1/2

```
/**
* Get Connection to CICS aus WAS Connection Factory
*/
public void ejbCreate() {
  javax.naming.Context ic=null;
  ic = new javax.naming.InitialContext();
  Connection eciConn = null;
  String cfRef = new String("java:comp/env/ECI");
  cf = (ConnectionFactory) ic.lookup(cfRef);
   ….
}


/**
* Aufruf von CICS Programm mittels CICS Connection aus Connection Factory
*/
private JavaStringRecord flowEciRequest(…){
  eciConn = getConnection(eciRD, cf);
  eciInt = getInteraction(eciConn);

  eciInt.execute(eSpec, jsr, jsr);

}
```

Abbildung 17: Ausschnitt aus PrimeCTGServlet.java
Teil 2/2

Selbst-Test
Schauen sie sich den Code des Form Tags in Abb. 13 an:

- **Welche Variable wird mit dem Namen „funcName“ gekennzeichnet?**
- **Welche Variable wird mit dem Namen „commareaInput“ gekennzeichnet?**

Der auf dem z/OS-WebSphere-Application-Server vorinstallierte Code besteht aus 7 Komponenten:

1. **main.jsp**
   ist die Java Servlet Page, die vom Browser aufgerufen wird.

2. **results.jsp**
   ist die Java Servlet Page, die an den Browser zurück gegeben wird

3. **error.jsp**
   ist die Java Servlet Page, die angezeigt wird, wenn ein Fehler aufgetreten ist

4. **PrimeCTGServlet.java**
   ist das Servlet, welches von main.jsp aufgerufen wird.

Diese 3 Komponenten wurden zu einem Web Archiv (.war) zusammengefasst, und im Verzeichnis PrimeCTGWeb installiert.

5. **CTGTesterCCIBean.java**
   Ist die EJB, welche das CICS Transaction Gateway enthält

6. **CTGTesterCCIHome.java**
   Ist die Home Interface von CTGTesterCCIBean.java

7. **CTGTesterCCI.java**
   Ist die Object Interface von CTGTesterCCIBean.java

Den vollständigen Code können Sie unter
http://www.cedix.de/Vorles/Band3/javazos/ctg/index.html

herunterladen.

Falls Sie dies interessiert, können Sie die Websphere Console unter "http://134.2.205.54:9525/ibm/console" finden, Als Benutzername können Sie einen beliebigen Werte eingeben, da das System nicht geschützt ist.

Das Tutorial ist eine Erweiterung eines Beispiels welches in dem folgenden Redbook enthalten ist:

CICS Transaction Gateway V5, The WebSphere Connector for CICS
August 2002, SG24-6133-01, Kapitel 10,
http://www.redbooks.ibm.com/redbooks/pdfs/sg246133.pdf
oder www.cedix.de/VorlesMirror/Band3/Java04.pdf

Und das war es, Sie haben erfolgreich das CTG Tutorial bearbeitet. Herzlichen Glückwunsch !