Rational. software

IBM

Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z

- IBM System z, IBM WebSphere, and IBM Rational tools covered
- IBM z/OS and IBM System z application building
- Scenarios and examples



Reginaldo Barosa
Surya Kumar Bose
Scott Davis
Jeremy Hamilton
Joachim Kaltenbach
David Lawrence
Veronique Quiblier
Zhenhua (Eric) Shen
Marc Terwagne
Kenichi Yoshimura

Redbooks

ibm.com/redbooks

Archived

Archived



International Technical Support Organization

Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z

February 2009

Archived

SG24-7482-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Archived

First Edition (February 2009)

This edition applies to Rational Developer for System z Version 7.1 through Version 7.4 and WebSphere Developer for System z Version 7.0.

© Copyright International Business Machines Corporation 2009. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices ix
Trademarks x

Preface xi
The team that wrote this book xii
Become a published author xiv
Comments welcome. xiv

Part 1. IBM Rational Developer for System z 1

Chapter 1. Using IBM Rational Developer for System z Version 7.1 to work with COBOL programs 3
1.1 Objectives 4
1.2 Working with Developer for System z and COBOL. 5
1.3 Scenario 1: Local code and no CICS or DB2 7
1.4 Scenario 2: Local code using CICS and DB2 10
1.5 Scenario 3: Code on z/OS and no CICS or DB2. 15
1.5.1 Scenario 3.1: Remote syntax check using the z/OS compiler 15
1.5.2 Scenario 3.2: Local syntax check using the local compiler. 17
1.5.3 Scenario 3.3: Working offline - move to local and use the local compiler. 18
1.6 Scenario 4: Code on z/OS that uses CICS but not DB2 23
1.6.1 Scenario 4.1:- Remote syntax check using the z/OS compiler with a remote COBOL/CICS program 23
1.6.2 Scenario 4.2: Local syntax check with a remote COBOL/CICS program 27
1.6.3 Scenario 4.3: Local syntax check using local compiler, remote COBOL/CICS program, and copybook with EXEC CICS 29
1.7 Conclusion 31

Chapter 2. Scenario for problem determination tools with Rational Developer for System z 33
2.1 Introduction 34
2.2 IBM Application Performance Analyzer for z/OS 34
2.2.1 Highlights 35
2.2.2 Support for IBM subsystems. 36
2.3 IBM Debug Tool Utilities and Advanced Functions for z/OS. 37
2.3.1 Highlights 37
2.4 IBM Fault Analyzer for z/OS 38
2.4.1 Highlights 38
2.4.2 Support for IBM subsystems. 39
2.4.3 Analysis options 40
2.5 IBM File Manager for z/OS 40
2.5.1 Highlights 40
2.5.2 File Manager components 41
2.5.3 SOA support 42
2.6 Optim MOVE. 43
2.6.1 Test data management. 43
2.6.2 Processing flow. 44
2.7 Workload Simulator. 46
2.8 Rational Developer for System z. 47

© Copyright IBM Corp. 2009. All rights reserved.

2.8.1 Support for the IBM problem determination tools 47
2.9 Scenario for PD Tools with RD/z 48

Chapter 3. The Rational Enterprise Software Configuration Management (ESCM) implementation 69
3.1 Enterprise repository 71
3.1.1 Build management 71
3.1.2 Organization and identification of assets 72
3.1.3 Using ClearCase meta data in z/OS builds 73
3.2 Managing your z/OS assets with ClearCase 80
3.2.1 Remote Build 80
3.2.2 ClearCase TSO Client 87
3.2.3 Programmatic access to the ClearCase repository from z/OS 91
3.2.4 Using ClearQuest to manage the application life cycle 93
3.3 Incorporating Build Forge with Remote Build 96
3.3.1 Build Forge 97
3.4 Enterprise Application Lifecycle Management with Rational Developer for System z . 99
3.4.1 Relational Developer for System z: a very brief overview 100
3.4.2 Working with ClearCase managed artifacts 100
3.4.3 Connecting to your repository 102
3.4.4 Working with controlled elements 103
3.4.5 Build, migrations, and promotions 104
3.5 Using WebSphere Studio Asset Analyzer for impact analysis 107
3.5.1 What WebSphere Studio Asset Analyzer (WSAA) is 107
3.5.2 Identify application dependency 108
3.5.3 Impact analysis 114

Part 2. IBM WebSphere Developer for System z 117

Chapter 4. Traditional and modern application development tools 119
4.1 TSO/ISPF 120
4.2 The dawn of programming: a look back 120
4.3 Emulating a mainframe on PCs 121
4.4 WebSphere Developer for System z 121
4.5 Improving developer productivity with WebSphere Developer for System z 122
4.5.1 During the coding phase (code assist) 122
4.5.2 During syntax checking (Compilation) 122
4.5.3 JCL generation 122
4.5.4 z/OS interactive remote debug 123
4.5.5 Get z/OS listings with JES monitor 123
4.6 “End to end” debug 123
4.7 How easy is it to replace TSO/ISPF with WebSphere Developer for System z 123
4.8 WebSphere Developer for System z as the standard interface for the mainframe . . . 124

Chapter 5. Enterprise Service Tools 125
5.1 Web Services terminology 126
5.2 Enterprise Service Tool perspective 126
5.3 The Enterprise Service Tools projects 129
5.3.1 The Enterprise Service Tool single-service project 130
5.3.2 Service flow project 132
5.3.3 Database Application Generator project 132

Chapter 6. Web service enablement for an existing CICS application 133
6.1 How to transform your existing CICS application in a Web service 134

6.1.1 Generation of the Web service enablement components . . . 135
6.1.2 Configuring CICS Web Services on z/OS . . . 145
6.1.3 Deploy the generated assets to CICS Transaction Server for z/OS . . . 153
6.2 When to use interpretive versus programmatic generated adapters . . . 165
6.2.1 Compiled XML conversion . . . 165
6.2.2 Interpretive XML conversion . . . 166
6.3 Batch processor . . . 167
6.3.1 Specify the options for the batch processor . . . 168

Chapter 7. Web service enablement for an existing IMS application . . . 179
7.1 What is the IMS Soap Gateway . . . 180
7.2 How to transform your IMS Application in a Web service . . . 181
7.2.1 Generation of the Web enablement components . . . 182
7.2.2 Deploy the Web Service converter source file . . . 193
7.2.3 Deploy the WSDL file and the correlator file to IMS SOAP Gateway . . . 194
7.2.4 Test your IMS Web Service using WebSphere Developer for System z . . . 200

Chapter 8. Service Flow tools . . . 203
8.1 Service flow: What, why, and when . . . 204
8.2 Create the Service Flow project . . . 205
8.3 Record the terminal interactions . . . 210
8.4 Runtime code generation and deployment . . . 219
8.5 Test the deployed service . . . 227

Chapter 9. Menu Manager . . . 231
9.1 Menu Manager Preferences window . . . 232
9.2 Creation of custom menu items . . . 234

Chapter 10. Core System z remote development . . . 243
10.1 MVS files properties . . . 245
10.2 ISPF editing versus LPEX editing . . . 251
10.2.1 ISPF profile within LPEX Editor . . . 252
10.2.2 Editing using LPEX . . . 254
10.2.3 Advanced functions . . . 270
10.3 Developing and debugging a batch program . . . 276
10.4 Developing and debugging a CICS application . . . 295
10.5 Developing and debugging an IMS application . . . 318
10.6 DB2 stored procedures builder for PL/I and COBOL . . . 338
10.7 Fault Analyzer plug-in for WD/z . . . 362
10.8 File Manager plug-in for WD/z . . . 380
10.8.1 IBM File Manager concepts . . . 381
10.8.2 New file mappings . . . 382
10.8.3 Template association . . . 384
10.9 Formatted editing . . . 384
10.9.1 Standard editing capabilities . . . 385
10.9.2 Find/Replace . . . 386

Chapter 11. Software Configuration Management support in WebSphere Developer for System z V7 . . . 389
11.1 Overview of WD/z SCM features . . . 390
11.2 Overview of CARMA . . . 391
11.2.1 CARMA basics . . . 392
11.2.2 Extending CARMA . . . 394
11.3 Overview of SCLM Developer Toolkit V2 . . . 402

11.3.1 IDE based Java/J2EE application development 403
11.3.2 Using SCLM view 407

Chapter 12. Sample implementation of a CARMA RAM 411
12.1 Motivation 412
12.2 Modeling considerations 412
12.2.1 SCM data model 412
12.2.2 Underlying SCLM project hierarchy 415
12.2.3 Functional SCM model 416
12.3 Implementing RAM 417
12.3.1 Configuring CARMA RAM functionality 418
12.3.2 Implementing CARMA RAM functions 419
12.4 Sample CARMA session 434

Chapter 13. IBM System z server components 463
13.1 Packaging differences of WD/z V6 464
13.2 Remote System Explorer (RSE) 464
13.3 Job Monitor 464
13.4 TSO Command Server 467
13.5 CARMA and SCLM Developer Toolkit 468
13.6 Initial view 470
13.6.1 Workspace 470
13.6.2 Version difference coexistence 470
13.6.3 Perspectives 470
13.7 Connecting to a System z 474
13.7.1 Differences of Version 6 474
13.7.2 Defining a system connection 474
13.8 Connecting to the defined system 480
13.8.1 Mapping 482
13.9 Alternative views 484
13.10 Developing single resources or projects 487

Chapter 14. Connecting to CICS 489
14.1 Introduction 490
14.1.1 Connecting to existing COBOL/CICS programs 490
14.1.2 Resource adapters 491
14.1.3 Importers 492
14.1.4 J2C wizard 492
14.1.5 Example application 493
14.2 Preparing the workstation for this example 494
14.2.1 Verify that DB2 has the required sample table 494
14.2.2 Install WebSphere Developer for System z Version 7 495
14.2.3 Install CICS Transaction Gateway (CTG) 496
14.2.4 Install TXSeries Version 6.1 and create a region 497
14.2.5 Configure TXSeries to support CTG 497
14.2.6 Configure CTG to communicate with TXSeries 497
14.2.7 Testing if CTG is communicating with TXSeries 499
14.2.8 Load the sample COBOL into WD/z, compile/link, and create the executables 500
14.2.9 Install the generated programs and map into TXSeries 503
14.2.10 Test the provided Client/Server COBOL programs 505
14.3 Build a J2C application to invoke the COBOL/CICS Server 506
14.3.1 Create a Java bean that invokes the COBOL/CICS Server 506
14.3.2 Test the CICS ECI connector using the JSP test client 514
14.4 Use WD/z to generate a Web service implementation from the sample application and

test it using the Web Services Explorer 516
14.4.1 Create a Web service that invokes the J2C Java Bean 516
14.4.2 Test the Web service 518

Chapter 15. Using IBM WebSphere Developer for System z with local COBOL, TXSeries, and VSAM (SFS) 523
15.1 Preparation 524
15.1.1 Generating an SFS file 524
15.1.2 Modifying COBOL/CICS programs and BMS maps 531
15.2 Debugging COBOL/CICS with VSAM/SFS 538

Chapter 16. z/OS Database Application Generator 563
16.1 z/OS Database Application Generator 565
16.2 Using the Database Application Generator 566
16.3 Working with the UML model 566
16.4 Working with the existing z/OS DB2 table 566
16.5 Scenario 1: Working with the existing z/OS DB2 table 567
16.5.1 Creating a DB2 table to be used by the Database Application Generator 567
16.5.2 Using the Database Application Generator wizard to create a CRUD and list applications 572
16.5.3 Understanding the generated assets 584
16.5.4 Working with the generated JCL 589
16.5.5 Deploying the generated assets to z/OS 591
16.5.6 Configuring CICS Web Services on z/OS 594
16.5.7 Installing the z/OS CICS services 597
16.5.8 Testing the Web Services 601
16.6 Creating a Web service client using Java Server Faces 607
16.6.1 Creating a Web project with a simple JSP page 608
16.6.2 Add the Faces JSP controls that will invoke the Web service 609
16.6.3 Testing the JSP Web page 611
16.7 Scenario 2: Working with an Unified Modeling Language model 613
16.7.1 Exporting the UML model using Rational Software Architect 613
16.7.2 Creating a Simple Project and importing the UML model 615
16.7.3 Using the Database Application Generator wizard from UML 615
16.7.4 Creating the DB2 table in the z/OS system 622
16.7.5 Deploying the code generated 623
16.7.6 Testing the Web Services 624

Part 3. Appendixes 625

Appendix A. Source script for the build engine 627

Appendix B. Additional material 645
Locating the Web material 645
Using the Web material 645
System requirements for downloading the Web material 646
How to use the Web material 646

Related publications 647
IBM Redbooks 647
Other publications 647
Online resources 647
How to get Redbooks 648
Help from IBM 648

Index 649

Archived

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2009. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AS/400®
Build Forge®
CICS®
ClearCase®
ClearQuest®
DB2 Universal Database™
DB2®
developerWorks®
eServer™
IBM®
Language Environment®
MQSeries®
OMEGAMON®
Optim™
OS/390®
Parallel Sysplex®
pSeries®
RACF®
Rational®
Redbooks®
Redbooks (logo) ®
System i®
System z®
Tivoli®
VisualAge®
VTAM®
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

EJB, J2EE, J2SE, Java, JDBC, JRE, JSP, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Expression, Microsoft, Visual Studio, Win32, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Pentium 4, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Archived

Preface

IBM® Rational® Developer for System z® speeds the development of traditional mainframe, Web, and composite applications and accelerates the development of your Web applications, traditional COBOL and PL/I applications, Web services, and XML-based interfaces.

IBM WebSphere® Developer for System z was renamed to IBM Rational Developer for System z.

Rational Developer for System z helps improve the productivity of developers creating and maintaining mainframe applications and Web services before, during, and after the transition to a SOA-based environment, as well as facilitating shorter development cycles that help reduce costs by extracting value from existing applications through reuse and transformation, making them accessible as Web services.

Rational Developer for System z helps to simplify the overall service-flow development process with enterprise service tools and enables developers to rapidly create well-built composite processing that integrates transactional and WebSphere–based environments.

This book has two parts. Part 1 covers Rational Developer for System z and Part 2 covers WebSphere Developer for System z.

Part 1 includes:

- Topics that cover how traditional application developers can use IBM Rational Developer for System z to significantly increase their productivity and efficiency when creating and maintaining traditional mainframe applications. This scenario shows you how. You download and use example code in a variety of typical development scenarios. The examples include programs that use CICS®, VSAM, and DB2®, including syntax checking and compiling. You also see the configuration steps to perform either local or remote syntax checking of COBOL programs. The program and associated assets (such as copybooks) could be stored on either z/OS® or the local workstation. Even though the configurations shown are for COBOL applications, PL/I programmers could follow this scenario and then apply similar steps to their PL/I programs. Scenarios show COBOL developers how to improve productivity using IBM Rational Developer for System z (hereafter called Developer for System z). You see how to check the syntax of COBOL assets located on the local workstation or by remotely accessing them on z/OS. You also see how to compile COBOL code locally on the local workstation or remotely accessing the compiler on z/OS. You go through several typical scenarios, you explore both local and remote syntax checking capabilities, and you see how to configure the workstation for the various scenarios.
- Additional scenarios use a subset of the Problem Determination Tools to demonstrate how the tools work together to quickly and efficiently detect an error, correct it, and test it. We do that with Fault Analyzer, Debug Tool, and File Manager, all under the umbrella of Rational Developer for System z. The same scenario can be accomplished in batch (green screen) without the use of Rational Developer for System z.
- Other scenarios present a detailed implementation with several alternative strategies that we used in the project for this book. We did this in the context of an overall ESCM life cycle, where Rational Developer for System z was positioned as the primary developer interface. We show how we implemented IBM Rational ClearCase® with the z/OS extensions to run native z/OS builds. We also describe how we used the integrations between Rational Developer for System z, IBM Rational ClearQuest®, and IBM Rational

© Copyright IBM Corp. 2009. All rights reserved.

Build Forge® and their associated integrations to implement an end-to-end ESCM life cycle.

Part 2 includes:

- Scenarios and examples of the Enterprise Service Tools, Web Service enablement for an existing CICS application, Web Service enablement for an existing IMS application, Service Flow tooling, Menu Manager, Core System z remote development, including debugging via Debug Tool, File Manager Interface, and Fault Analyzer Interface, Software Configuration Management support in WebSphere Developer for System z V7, a sample implementation of CARMA RAM, System z server components, using WebSphere Developer for System z to connect to CICS, an example of using J2EE™ connectors, CICS Transaction Gateway, and TXSeries, and using WebSphere Developer for System z to work with Local COBOL, TXSeries, and VSAM (SFS).

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center. and various international locations.

Reginaldo Barosa is an IBM Senior Certified Application Development Specialist. He provides sales support, helping customers with Enterprise Modernization Solutions and development tools, such as Rational Developer for System z. Before joining IBM US 8 years ago, Reginaldo worked for 27 years at IBM Brazil. He has co-authored IBM Redbooks® publications and has written two books, as well as other articles and tutorials for IBM developerWorks®. He holds a degree in Electrical Engineering from Instituto Mauá de Tecnologia, São Paulo, Brazil.

Surya Kumar Bose is an IT consultant from Bose Information Technology, an IBM Business Partner based in Hamburg. He has 33 years of experience in the software industry, covering a wide spectrum in the field of application development, mainly in the area of IBM mainframes. He has specialized in modernization and migration of older systems using WebSphere Developer for System z and its predecessors, WSED, and the VisualAge® products. He holds a degree in Physics from the University of Calcutta, India. He has written extensively in various computer journals and is an active participant at IT conferences.

Scott Davis is a certified IT specialist at IBM Germany. After college, Scott joined the IBM Software Development Lab in Boeblingen, Germany. After 8 years of programming in COBOL, PL\I for DB2, CICS, and IMS, he moved to the Software Group in IBM Germany.

Jeremy Hamilton is a System z Technical Sales Specialist from Atlanta, GA. He has five years of experience with IBM, four at the IBM Almaden Research Center in California working on Molecular Electronics and Nano Fabrication, and one as a vitality hire for the System z PD tools technical sales team. He is member of the Cherokee nation and holds a Bachelors of Science in Engineering/Physics from Fort Lewis College in Durango, CO, and a Masters of Science in Information Systems from Santa Clara University in Santa Clara, CA.

Joachim Kaltenbach is a Consulting IT Specialist working as an employee at the IBM Business Partner company SoforTe GmbH in Germany. Joachim has 15 years of experience in planning, implementing, and optimizing software development environments. His special field of knowledge is Software Configuration Management (SCM). Joachim is experienced in managing various SCM tools both in the IBM host and the distributed world.

David Lawrence is an enablement engineer for the Enterprise Tools and Compilers Ecosystem team. He initially joined Rational Software in 1996, after several years as a customer of Rational ClearCase. He has delivered consulting and training services for Rational SCM solutions to many customers in many industries (banks, telecoms, retail, and so on). He led the first installation and configuration of the ClearCase z/OS extensions at a large European bank in 2001. He is now responsible for enabling IBM Rational Sales teams and strategic customers in implementations of Rational solutions for System z.

Veronique Quiblier is an IBM Certified I/T Specialist with IBM Technical Sales in Europe. She has 20 years of experience in the field of Application Development. She provides sales support, helping customers with application transformation solutions and development tools.

Zhenhua (Eric) Shen is an enablement engineer in Rational IBM, focusing on SCM solutions for System z and System i®. Eric has 10 years of software development and enablement experiences. Prior to joining Rational in 2002, he was the software developer lead in a major industry software automation company. Eric holds a Master's of Science degree in computer engineering from Northeastern University and a MBA degree in general management from Babson College.

Marc Terwagne is a Software Technical Sales specialist for the Application Development Tools for System z team in Belgium. He has 22 years of experience in the IT field. Marc holds a degree in Computer Science from the School of Computer Science of Jemelle in Belgium. His areas of expertise include IT services for MVS and OS400 for insurance companies and industries as a programmer, programmer analyst, and project leader. Since 1999, he has worked in Technical Sales for the VisualAge and WebSphere Tools and now for AD Tools for System z.

Kenichi Yoshimura joined IBM in 1977, having previously worked in the industry in various customer sites. His current role is as a Senior IT Specialist in a technical sales role in the WebSphere Pillar of the A/NZ (Australia/New Zealand) zSeries® Software Group. This involves working with customers and IBM colleagues in the marketing and deployment of transaction and messaging systems (CICS, WMQ, and WebSphere Application Server), and the tools and languages that support the development and operation of applications in this space.

Thanks to the following IBM colleagues for their contributions to this book:

Beng Chiu
Omkar Nimbalkar
Jason C. Garcowski
Bob Kennedy
Ray Green
Joel Duquene
Charles Chu
Reginaldo Barosa
Roger Hewton
Dale Newby
Sanjay Chandry
Peeter Costigan
Dudley Thomas
Scott Laningham
Jim Hildner
Duncan Upton
Stephen Hunt
Alan Schwartz
Emeka Igbokwe

Archive

Steve Borriello
John Leake
Jack Verstappen
Steve Rubin
Francisco Anaya
Jim Hildner
Steve Seifert
Marty Shelton
John Casey
Thomas Pietrcollo
Mark Duckworth
Larry Cox
Tony Lee
Graham Hannington
Louis Gentry
Tsuneo (Sam) Horiguchi
Peter Henningsen
Manual Moreno Villares
Bill Halterman
Paolo Cravino
Brandt Onorato
Antonio Alonzo Lopez
Scott Longwell

and to Joe DeCarlo, Manager for Special Projects at the ITSO Center, San Jose, CA.

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

Archived

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Archived

Archived

Part 1

IBM Rational Developer for System z

Part 1 includes topics developed with Rational Developer for System z V7.1 through V7.4.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

Archived

1

Using IBM Rational Developer for System z Version 7.1 to work with COBOL programs

Traditional application developers can use IBM Rational Developer for System z to significantly increase their productivity and efficiency when creating and maintaining traditional mainframe applications. This scenario shows you how. You download and use example code in a variety of typical development scenarios. The examples include programs that use CICS, VSAM, and DB2, including syntax checking and compiling. You also see the configuration steps to perform either local or remote syntax checking of COBOL programs. The program and associated assets (such as copybooks) could be stored on either z/OS or the local workstation. Even though the configurations shown are for COBOL applications, PL/I programmers could follow this scenario and then apply similar steps to their PL/I programs.

This scenario shows COBOL developers how to improve productivity using IBM Rational Developer for System z (hereafter called Developer for System z or RD/z). You see how to check the syntax of COBOL assets located on the local workstation or by remotely accessing them on z/OS.

You also see how to compile COBOL code locally on the local workstation or remotely accessing the compiler on z/OS. You go through several typical scenarios, explore both local and remote syntax checking capabilities, and see how to configure the workstation for the various scenarios.

This chapter covers several of many possible scenarios, including ones using COBOL with and without CICS, one with DB2 on the local workstation, and COBOL programs that access VSAM on z/OS.

Although this scenario demonstrates various tasks using COBOL assets, you could use similar steps to work with PL/I.

Note: For more information, download the SG247482-Chapter 1.zip file referenced in Appendix B, “Additional material” on page 645.

© Copyright IBM Corp. 2009. All rights reserved.

1.1 Objectives

When you complete this scenario, you should know how to use Developer for System z to:

- Work with COBOL batch code and with CICS statements that access either VSAM or DB2.
- Work with COBOL/CICS/DB2 code on the local workstation.
- Work with COBOL/CICS code on z/OS.
- Check the syntax of COBOL code on your local workstation.
- Compile COBOL code on your local workstation.
- Check the syntax of COBOL code stored on z/OS.
- Compile COBOL code stored on z/OS.

Prerequisites

To get the most out of this scenario, you should be comfortable with COBOL programming on z/OS and have some familiarity with Rational Developer for System z.

System requirements

On your workstation, you need Rational Developer for System z Version 7.1 and IBM TXSeries for Multiplatforms Version 6.1 installed to test the examples. TXSeries is included in Rational Developer for System z V7.1. You must also have the Developer for System z components installed and configured on the mainframe because you will need to connect and submit jobs for remote execution. You need to have a remote system already defined on your workstation and you should be connected to that system.

For a scenario that uses a local database, you will also need DB2 installed on your workstation and the database named SAMPLE created. This scenario was tested with DB2 Version 8.2, but you could use a later version, if desired.

You also will need to allocate z/OS data sets to hold the assets and a VSAM file. This scenario provides the sample programs, JCL, and properties that are used. See “About the download” on page 5 for information about the assets provided in the download. Table 1-1 shows the required data sets and their characteristics to implement the scenarios. The high level qualifier is XXX. You can use the ALLOC _PDS.jcl job (provided in the download) to help you allocate the data sets.

Table 1-1 z/OS data sets to allocate for scenarios

Data set name	Record format	Record length	Block size	Contents
XXX.POT.COBOL	FB	80	8000	COBOL programs
XXX.POT.COPYLIB	FB	80	8000	COBOL copybooks
XXX.POT.LISTING	FBA	133	1330	Compiler listing
XXX.POT.LOAD	U	0	6233	Load module
XXX.POT.OBJ	EB	80	3200	Object
XXX.GENERTED.JCL	FB	80	8000	JCL

About the download

The download includes code that you can use to implement the scenarios. Unzip the download file to a temporary drive, such as c:\temp. You load the code according to the instructions in the scenarios discussed later in this chapter.

The downloaded zip file includes these assets:

- SCENARIO#1_Project_interchange.zip: To create the project used in scenario #1.
- SCENARIO#1_PROPERTIES.xml: Properties used in scenario #1.
- SCENARIO#2_Project_interchange.zip: To create the project used in scenario #2.
- SCENARIO#2_PROPERTIES.xml: Properties used in scenario #2.
- SCENARIO#3_PROPERTIES.xml: Properties used in scenario #3.
- SCENARIO#4_PROPERTIES.xml: Properties used in scenario #4. ALLOC_PDS.jcl can be used to allocate the PDS required.
- DEFVSAM.jcl: Creates and loads the VSAM file used in the COBOL programs.
- P1VSAM.cbl: COBOL program used in the scenarios without CICS and DB2.
- VSAMCOPY.cpy: COBOL copybook used by P1VSAM and P4CICS programs.
- P4CICS.cbl: COBOL program used in the scenarios with CICS.
- COMAREA.cpy: COBOL copybook used by P4CICS program.
- P43CICS.cbl: COBOL program used in the CICS scenario with copybook with EXEC CICS.
- CUSTREC, CUSTIORQ and COPYCICS: COBOL copybooks used by P43CICS program.

1.2 Working with Developer for System z and COBOL

This section shows you some of the ways you can use Developer for System z to work with COBOL code that can reside either on your local workstation or on a remote System z machine running z/OS.

Understanding work style choices

Rational Developer provides two different ways to work with z/OS assets:

- You can use the Remote System Explorer perspective to perform z/OS-based tasks that are equivalent to those done in ISPF, such as:
  - View lists of data sets and members.
  - Allocate or delete data sets.
  - Create or delete members.
  - Edit text.
  - Submit JCL.
  - Get the output.

  You perform most of those tasks in the Remote Systems view. Before you can perform any of these tasks from the workbench, you need to define a system by specifying a set of properties. Refer to the IBM Rational Developer for System z Version 7.1.1 Information Center to learn the basics of working in the Remote System Explorer.

- You can also use the z/OS Projects perspective to complete specific development tasks. For example:
  - To develop code that resides and runs on z/OS, you define a z/OS project and one or more MVS subprojects. Projects are containers that hold one or more related subprojects. They enable you to build multiple load modules using a single build command. Subprojects contain the files and programs for a single load module. They are created under a particular remote system, and the properties of that system act as the defaults for subproject properties. You can modify or override the subproject properties.
  - To develop COBOL or PL/I code that resides and runs on the workstation, you define a local project. In this case, the project is not related to a system definition. This book will not cover this kind of project.

By default, the z/OS Projects perspective includes the Remote Systems view, where you can perform several of the mainframe tasks described above. This perspective also includes views that let you get the output of a specific job and then display it. To learn the basics of working in the z/OS Projects perspective, see the IBM Rational Developer for System z Version 7.1.1 Information Center.

System properties, z/OS subproject properties, and local project properties make the development environment more convenient. You may wish to share properties with team members to help start up new projects.

Checking syntax

Developer for System z provides two options for checking the syntax of your remote resources: *local syntax check* and *remote syntax check*.

When you develop COBOL (or PL/I code) in an MVS subproject, you can do a local or remote syntax check on a particular source file. A local syntax check uses the Windows® COBOL (or PL/I compiler) to check the syntax of resources. You can use a local syntax check on resources stored locally or on remote resources. Local syntax checking saves processing time and avoids the expense (if any) that is associated with added z/OS CPU cycles.

To perform a local syntax check, the source file you want to check must be in an MVS subproject. The subproject may be in either an “offline” or “online” state. If the subproject is in an offline state, then all the copybooks that the source uses should be part of the “offline” subproject for a proper syntax check.

A remote syntax check uses the JCL procedure support in Developer for System z to submit resources to a z/OS-based compiler for syntax checking. For remote syntax checking from an MVS subproject, the subproject must be in an online state. A remote syntax check lets you do a final syntax check on the target machine.

For both local or remote syntax checks, any copybooks, include files, or macros are checked to determine if they are referenced in the source and if they are accessible. Errors are displayed in the Remote Error List view, and you can double-click an error entry to open the file (potentially an included file) where the error was found.

In the absence of language elements that require special processing, first check your code locally. After you have removed most of the errors, you can check your code against a z/OS-based compiler.

Archived

Introducing the scenarios

Let us look at the scenarios in this chapter:

- Scenario 1: The COBOL program and the copybooks are stored on the local workstation, and the user has no connection to z/OS. The program does not use CICS or DB2.
- Scenario 2: The COBOL program and the copybooks are stored in the local workstation and the user has no connection to z/OS. The program uses CICS and DB2.
- Scenario 3: The COBOL program and copybooks are stored on z/OS. The program does not use CICS or DB2.
  - Scenario #3.1: Check syntax using the compiler z/OS.
  - Scenario #3.2: Check syntax using the compiler on the local workstation.
  - Scenario #3.3: Working offline (Move to local and use the local compiler).
- Scenario 4: COBOL program and copybooks are stored on z/OS. The program uses CICS but not DB2.
  - Scenario #4.1: Remote syntax check (using z/OS Compiler) for remote COBOL/CICS program.
  - Scenario #4.2: Perform a local syntax check (using the local compiler) for a remote COBOL/C ICS program.
  - Scenario #4.3: Perform a local syntax check (using the local compiler) for a remote COBOL/C ICS program, and the copybook has EXEC CICS statements.

1.3 Scenario 1: Local code and no CICS or DB2

In this scenario, you work with a COBOL program that does not use CICS statements and does not access DB2 tables. The COBOL program has COBOL copybooks with data definitions. The COBOL source and the COBOL copybook are stored in the local workspace. In this case, you can do some work offline, without any z/OS connection. Also, depending on the COBOL program, you could even test and debug the program, even if the program uses DB2 tables, because you can have local databases defined.The sample program used in this scenario (P1VSAM) accesses VSAM data sets. It would not normally be a good candidate for local tests because you cannot emulate VSAM in the local workstation. However, you can clear the syntax errors using the local COBOL compiler provided with Developer for System z if all necessary COBOL copybooks are also available in the local workstation.

Example 1-1 shows part of this local program used in this test case.

Important: You need the VSAMCOPY copybook to avoid compiler errors.

Example 1-1 P1VSAM, a sample program without CICS and DB2

```
IDENTIFICATION DIVISION.
PROGRAM-ID. P1VSAM.
AUTHOR. R. Barosa.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT POTVSAM-FILE
ASSIGN to POTVSAM
ORGANIZATION is INDEXED ACCESS MODE is RANDOM RECORD KEY is CUST-NO FILE STATUS is
POTVSAM-STATUS.
```

```
DATA DIVISION.
FILE SECTION.
FD POTVSAM-FILE.
COPY VSAMCOPY.
WORKING-STORAGE SECTION.
*
01 WS-PROGRAMPIC X(08) VALUE 'P1VSAM'.
01 WS-LITERAL-WS PIC X(48) VALUE
....
....
PROCEDURE DIVISION.
DISPLAY "Program P1VSAM starting.... " OPEN INPUT POTVSAM-FILE
IF POTVSAM-STATUS = '00'
CONTINUE
ELSE
DISPLAY 'POTVSAM: FAILURE-OPEN-!!!' GO BACK.
MOVE 1 to W-CUST-NO.
PERFORM 0100-POTVSAM-READ THRU
0100-EXIT UNTIL W-CUST-NO = 010. CLOSE POTVSAM-FILE.
....
....
```

In this example, the program is stored in a local project under the folder named program and the COBOL copybook is stored under folder named copybook, as shown in Figure 1-1. An exported project interchange file is provided if you are interested in following this example.

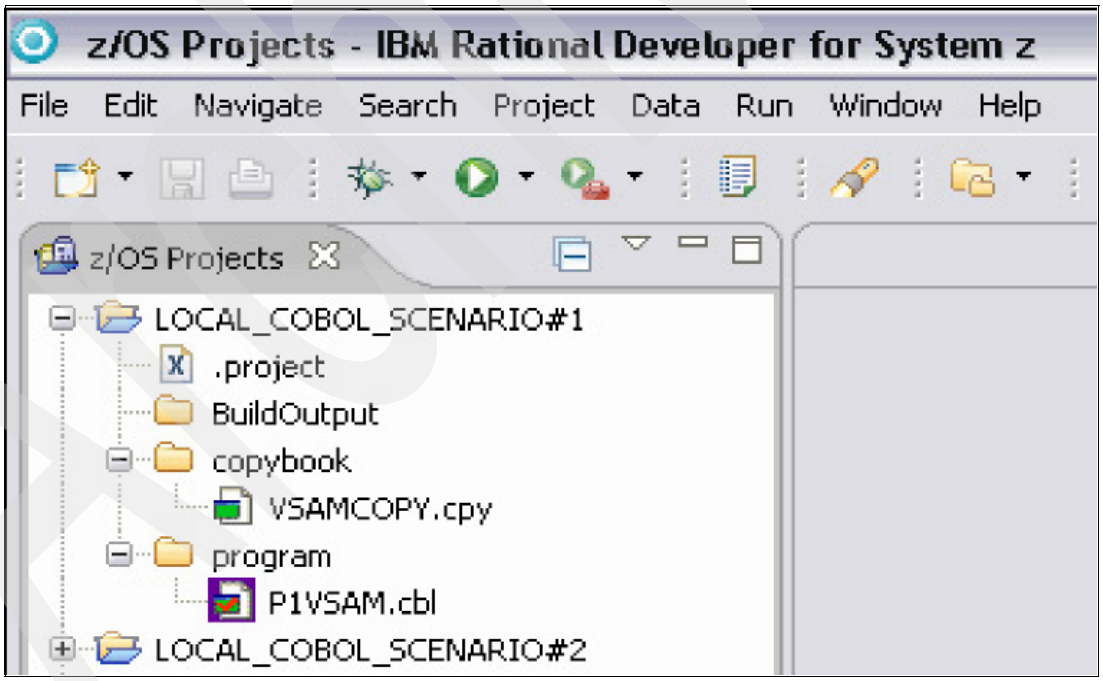


Figure 1-1 Scenario 1: Local COBOL and copybook without CICS and DB2

If you perform a local syntax check against this program without configuring the correct project properties, you will have syntax errors indicating undefined data. In the project property, you need to specify the location of the copybook, because the data names are defined there.

In our example, the copybooks are stored in C:\Workspaces\ scenario _COBOL\ LOCAL _COBOL _SCENARIO#1\ copybook, so make sure the SYSLIB settings point to this location.

To create the project and perform a COBOL syntax check using the provided assets, do these steps:

1. In Developer for System z, select **File** → **Import** → **Other** → **Project interchange**, and point to SCENARIO#1 _Project_interchange. zip. When the project is created, a syntax check is performed and some errors occur with return code 12.
2. Right-click the project **LOCAL_COBOL_SCENARIO#1**, select **Import Properties,** and point to SCENARIO#1 _PROPERTIES.xml. This step is not required now, but it will be when you move projects to other workspaces. The exported properties will not reflect your workspaces; you will need to adjust them in the next steps.
3. Right-click the folder **copybook** and select **Properties**. In the Info page, press Ctrl+C to copy this location. Later, you will paste it into an entry field.
4. Right-click the **LOCAL_COBOL_SCENARIO#1** project and select **Properties**. In the Local Cobol Build Options property, paste this location to SYSLIB, as shown in Figure 1-2. (Again, you need to specify this copybook location to be able to perform local syntax checking.)

Tip: If you need to refer to multiple directories for resolving copybooks, then separate the SYSLIB entries using a semi-colon. Later, you will see that for remote properties and the SYSLIB property, the libraries need to be separated by spaces.

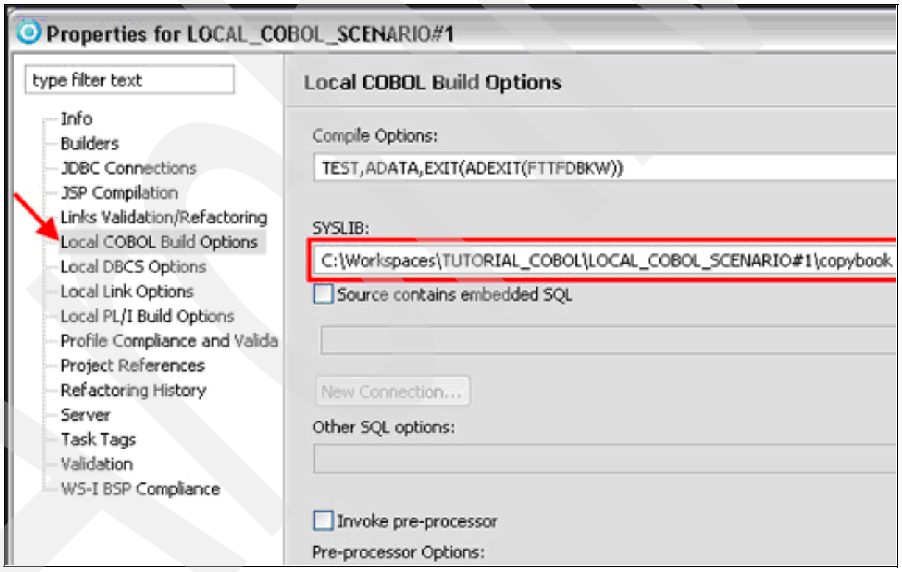


Figure 1-2 SYSLIB required for local COBOL syntax check without CICS or DB2

5. To run this program, right-click the **P1VSAM.cbl** program and select **Nominate as Entry Point**.
6. To perform local syntax checking, right-click the **P1VSAM.cbl** program and select **Local Syntax Check**. The possible errors or warnings will be shown in the Remote Error List view.
7. To create an executable, right-click the **LOCAL_COBOL_SCENARIO#1** project, and select **Rebuild project**. The executables are created in the BuildOutput folder.

This scenario showed an example COBOL program with copybooks that have no DB2 or CICS access. You could use the Interactive debug and do some simple tests with this code, bypassing the VSAM file I/Os.

In the next scenario, we add more complexity with a COBOL program that uses CICS and accesses a LOCAL DB2 table.

1.4 Scenario 2: Local code using CICS and DB2

In this scenario, you work with a COBOL program that uses CICS statements and accesses DB2 tables. The COBOL program has COBOL copybooks with data definitions, and the COBOL source and the COBOL copybook are stored in the local workspace. In this case, you can do some work offline, without any z/OS connection.

In this example, you can even test and debug the COBOL program without a z/OS connection because you have a local DB2 database and table. This program is part of a client/server example. You focus now on the syntax checking capabilities of Developer for System z; you do not test and debug this code. To perform the syntax check of this program, you need to have the DB2 SAMPLE database and the STAFF table installed on your workstation or in any server that has DB2 installed and to which you have access.

Example 1-2 shows part of the COBSERV local program used in this scenario. You need the STAFFDB and COMMAREA copybooks to avoid compiler errors.

Example 1-2 COBSERV sample program with CICS and DB2 access

```
PROGRAM-ID. COBSERV.
AUTHOR. Reginaldo Barosa.
INSTALLATION. IBM Boston.
DATE-WRITTEN. 23/03/07
***************************************************************
* DESCRIPTION : CICS SUBROUTINE TO RETREIVE CUSTOMER DATA
*
* Receive customer #, read DB2, move data to COMMAREA
*
* ENVIRONMENT : CICS, COBOL II, DB2
*
* CICS TRANS : NONE
*
* SUBROUTINE : NONE
*
* DB2 Database: SAMPLE
*
* DB2 Table : STAFF
*
***************************************************************
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-PROGRAM PIC X(08) VALUE 'COBSERV'.
01 WS-LITERAL-WS PIC X(48) VALUE
' WORKING STORAGE STARTS HERE'.
01 WORK-VARIABLES.
05 WS-RESP PIC S9(8) VALUE 0 COMP.
EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
01 I PIC S9.
COPY STAFFDB.
*
---------------------------------------------------------- *
LOCAL-STORAGE SECTION.
*
---------------------------------------------------------- *
01 WS-LITERAL PIC X(48) VALUE
' LINKAGE SECTION STARTS HERE'.
*
---------------------------------------------------------- *
LINKAGE SECTION.
*
---------------------------------------------------------- *
COPY COMMAREA.
*
---------------------------------------------------------- *
PROCEDURE DIVISION.
IF EIBCALEN = 0
EXEC CICS
RETURN
END-EXEC
END-IF.
IF EIBCALEN = LENGTH OF DFHCOMMAREA
PERFORM 0000-GET-CUSTOMER-DATA
EXEC CICS
RETURN
END-EXEC
ELSE
EXEC CICS
RETURN
END-EXEC
END-IF.
*
---------------------------------------------------------- *
0000-GET-CUSTOMER-DATA.
*
---------------------------------------------------------- *
MOVE WORK-ID TO db2-id.
EXEC SQL CONNECT TO SAMPLE END-EXEC.
EXEC SQL SELECT ID, NAME,DEPT,
JOB, SALARY, COMM
INTO :db2-id,
:db2-name, :db2-dept, :db2-job,
:db2-salary:db2-salary-null,
:db2-comm:db2-comm-null
FROM STAFF WHERE ID = :db2-id
END-EXEC.
IF SQLCODE = 0 THEN
MOVE 0 TO WORK-RETCODE
MOVE db2-name-data(1:db2-name-len)TO WORK-NAME
IF db2-salary-null IS NOT NEGATIVE
MOVE db2-salary TO WORK-SALARY
….
```

Archived

In this example, the program is stored in a local project under the program folder and the COBOL copybooks are stored under the copybook folder, as shown in Figure 1-3.

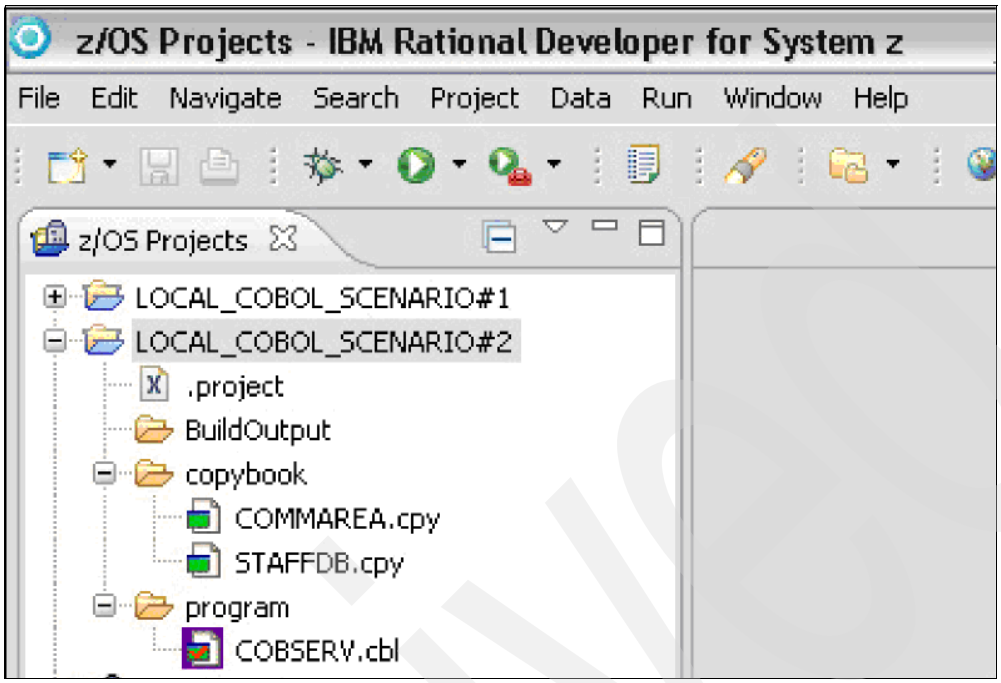


Figure 1-3 Scenario 2 - Local COBOL and copybook with CICS and DB2

If you perform a local syntax check against this program without configuring the correct project properties, you will have syntax errors indicating undefined data. In the project property, you need to specify the location of the copybook because the data names are defined there.

In this scenario, the copybooks are stored in C:\Workspaces\scenario_COBOL\LOCAL_COBOL_SCENARIO#2\copybook, so you need to specify this location in the SYSLIB settings. Also, DB2 and CICS use some definitions that are not found by default; you will need to specify them for SYSLIB as well. For example, if DB2 is installed at c:\ sqllib and TXSeries uses its default location, you need to add the path c:\opt\cics\include;c:\sqllib\include\cobol_a;.

To create the project and perform a COBOL syntax check using the assets provided in the download (see “About the download” on page 5 for more information), do these steps:

1. In Developer for System z, select **File** → **Import** → **Other** → **Project interchange** and browse to SCENARIO#2_Project_interchange.zip.When the project is created, a syntax check is performed and some errors occur with return code 12.
2. Right-click the **LOCAL_COBOL_SCENARIO#2** project, select **Import Properties**, and browse to SCENARIO#2_PROPERTIES.xml. This step is required because you need to have some DB2 and CICS special definitions. Also, the exported properties might not reflect your workspaces; you will need to adjust them in the next steps.
3. Right-click the copybook folder and select **Properties**. In the Info page, press Ctrl+C to copy this location so you can paste it to an entry field in the next step.
4. Right-click the project **LOCAL_COBOL_SCENARIO#2** and select **Properties**. Click **Local COBOL Build Options** (in the list on the left pane) and paste the location to SYSLIB, as shown in Figure 1-4 on page 13.

Important: You must specify the location to be able to perform local syntax checking.

5. In the SYSLIB field, type the path to the DB2 sqlca.cbl file, separating the parts with semicolons. The default path is C:\Program Files\IBM\SQLLIB\include\cobol_a. Because you are using DB2 and CICS, you also need to add the CICS required libraries. The path for DB2 and CICS is: c:\opt\cics\include;c:\sqllib\include\cobol_a;.
6. The COBOL program for this scenario accesses a local database named SAMPLE. To specify the database name to the compiler, specify it in the Compile Options field:

   SQL ('DATABASE SAMPLE').

Tip: Instead of adding the database name and passwords in the compile options, you could click the **New Connection** button to configure the DB2 database connection. In fact, this would be the preferable way, especially if the database is on the System z server. However, to simplify this scenario, you can just specify the database name in the compile options.

Make sure that **Source contains embedded SQL** is checked.

Because this program uses CICS, make sure that **Invoke the pre-processor** is checked, and in the Pre-processor Options field, add CICSNT -IIBMCOB –adesvp.

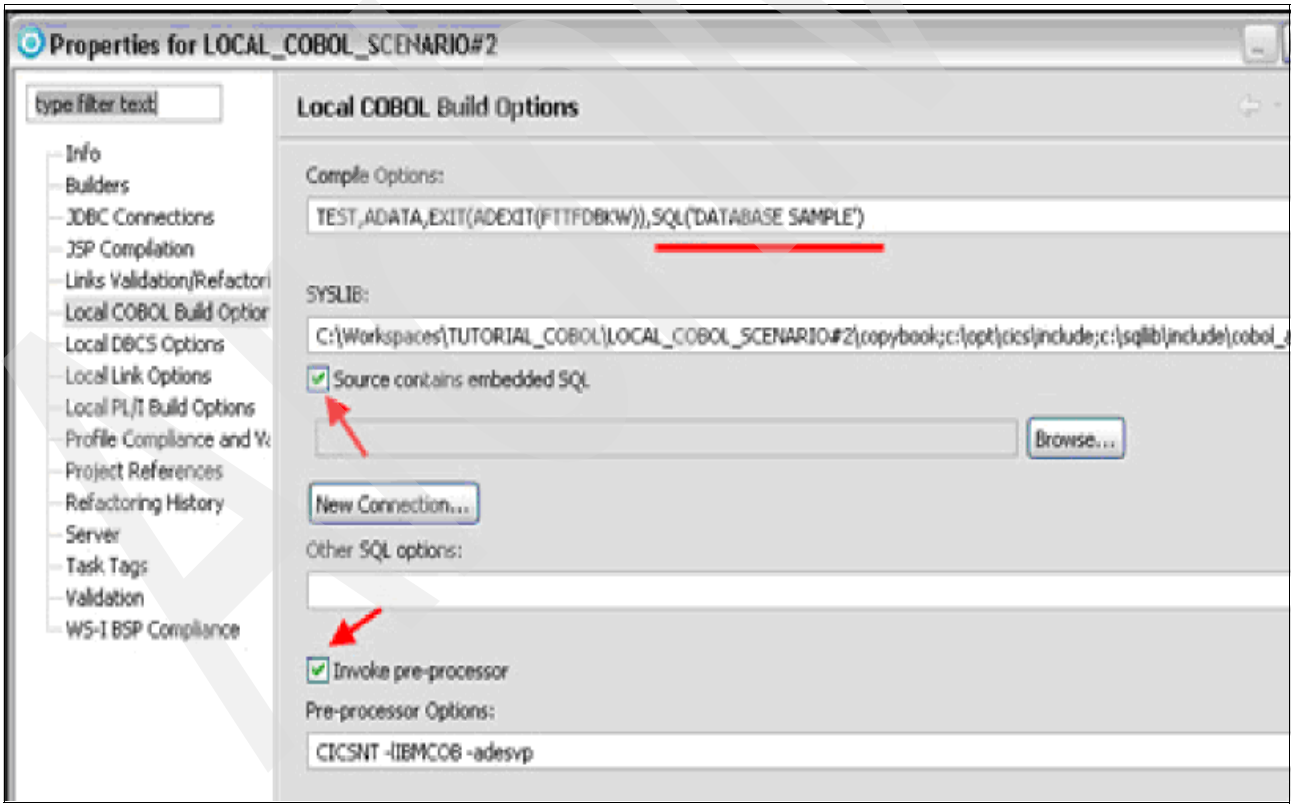


Figure 1-4 SYSLIB required for local COBOL syntax check with CICS and DB2

7. In the left pane, click **Local Link Options**. In the Link Options field, enter /de c:\opt\cics\lib\cicsprIBMCOB.lib.

Important: If you did not install TXSeries in its default install location, change the c:\opt\cics part of the second link options value to the directory where you installed TXSeries, as shown in Figure 1-5 on page 14.

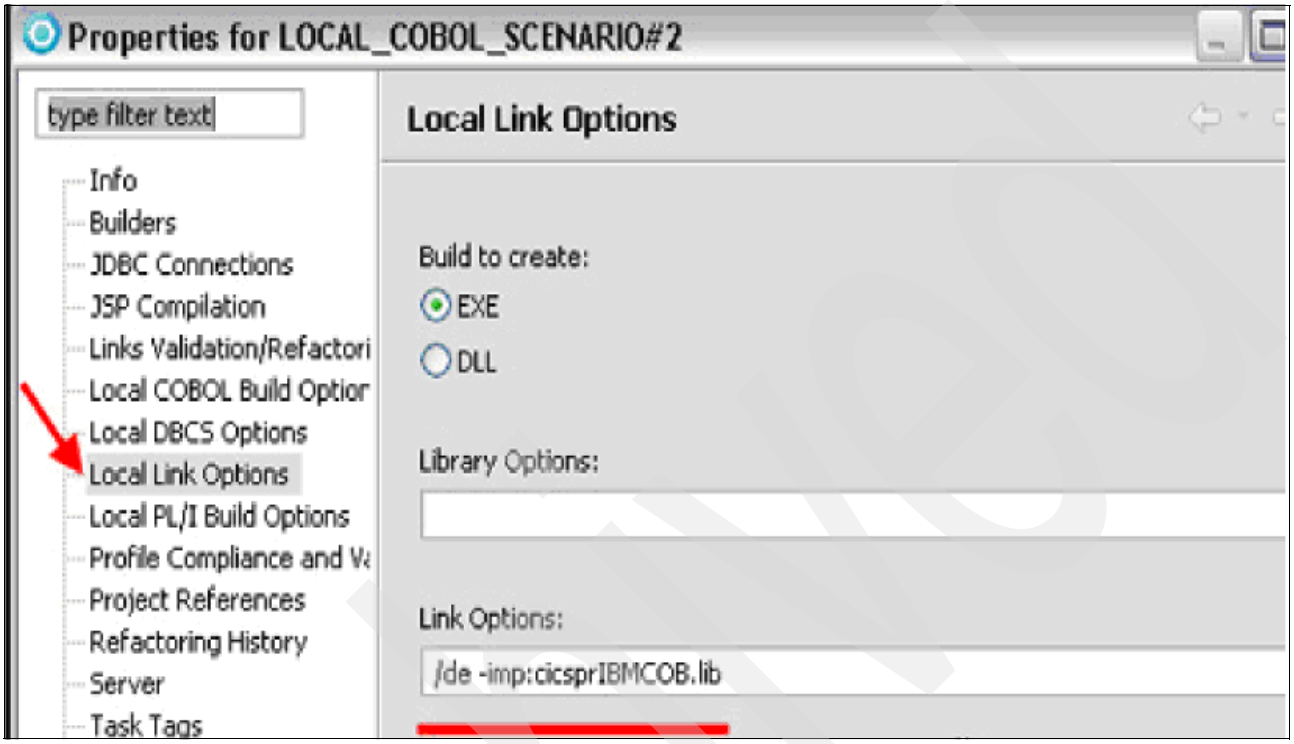


Figure 1-5 Local link options for COBOL with CICS

8. Close the properties.
9. To run the program after running the syntax check, right-click the **COBSERV.cbl** program and select **Nominate as Entry Point**.
10. To perform a local syntax check, right-click the **COBSERV.cbl** program and select **Local Syntax Check**. Any error messages and warnings display in the Remote Error Listview.
11. To create an executable, right-click the **LOCAL_COBOL_SCENARIO#2** project and select **Rebuild project**. The executables are created in the BuildOutput folder. Because this is a CICS executable, using the options that you specified, a file named COBSERV.ibmcob is created. This is the file that you export to CICS.

This scenario shows an example that uses a COBOL program with copybooks that use DB2 and CICS. You can use the Interactive debug to perform some simple tests with this code.

In the next two scenarios, you work with COBOL programs that are stored on the z/OS system.

1.5 Scenario 3: Code on z/OS and no CICS or DB2

In this scenario, you work with a COBOL program that does not use CICS statements and does not access DB2 tables; however, the COBOL program has COBOL copybooks that include data definitions.

1.5.1 Scenario 3.1: Remote syntax check using the z/OS compiler

To perform syntax checks, the compiler needs to have access to the COBOL copybooks; otherwise, the compiler will flag errors for missing data names that are defined.

For this scenario, you use the P1VSAM COBOL program, the same program used in Scenario 1; however, this time the program is located on z/OS, not on the workstation. P1VSAM does not have CICS or DB2 statements; instead, it accesses VSAM data sets on z/OS.

Before you start this scenario, you need to allocate some PDSs and move the COBOL program and the COPYBOOK to that data set:

1. Create a z/OS connection using the Remote Systems view, and connect to z/OS. This example assumes you have already created a connection called ZSERVEROS.
2. Use the z/OS Projects perspective to locate and edit the JCL (ALLOC_PDS.jcl, which was provided in the download) to allocate the PDS on z/OS. Use the Remote Systems view and Local/Local Files/Drivers folders. Change EMPOTXX to your z/OS user ID.
3. Submit the allocation data sets JCL to z/OS using “submit to ZSERVEROS” where ZSERVEROS is the z/OS connection you have created previously.
4. To copy the P1VSAM COBOL program to XXX.POT.COBOL and the VSAMCOPY copybook to XXX.POT.COPYBOOK, use the Remote Systems view and copy/paste from the assets used in Scenario #1.

To create the z/OS project and perform a COBOL syntax check, do these steps:

1. First, make sure you are connected to z/OS. Expand **MVS Files** to see the data sets that you allocated.
2. The download includes the SCENARIO#3_MVS_PROPERTIES.xml file, which defines the properties for the data sets you allocated; these properties are required for this scenario. Review this file and, using a text editor, adjust the properties for your environment, if necessary. When you have completed any adjustments, use the Remote Systems view, right-click **MVS Files**, select **Import Properties**, and point to your revised SCENARIO#3_MVS_PROPERTIES.xml. The generated JCL feature of Developer for System z uses those properties as a model.
3. Using the z/OS Projects perspective, create a z/OS project named REMOTE_COBOL_SCENARIOS and an MVS subproject named SCENARIO#3. Add the P1VSAM COBOL member in EMPOTXX.POT.COBOL to the SCENARIO#3 subproject. (Refer to the IBM Rational Developer for System z Version 7.1.1 Information Center to learn the basics of working with z/OS projects.)

Tip: The data set that contains the COBOL required copybook, VSAMCOPY, does not need to be part of the project.

4. Next, you need specify some properties and point to the data set where the COBOL copybooks (EMPOTXX.POT.COPYLIB) reside. During a syntax check, Developer for System z gathers the remote properties you set when you enable remote synchronization and generates a syntax check JCL, and submits the JCL to the remote system. Figure 1-6 shows the required COBOL settings to perform our scenario. If you do not specify the data set, the compiler will not find the copybook.

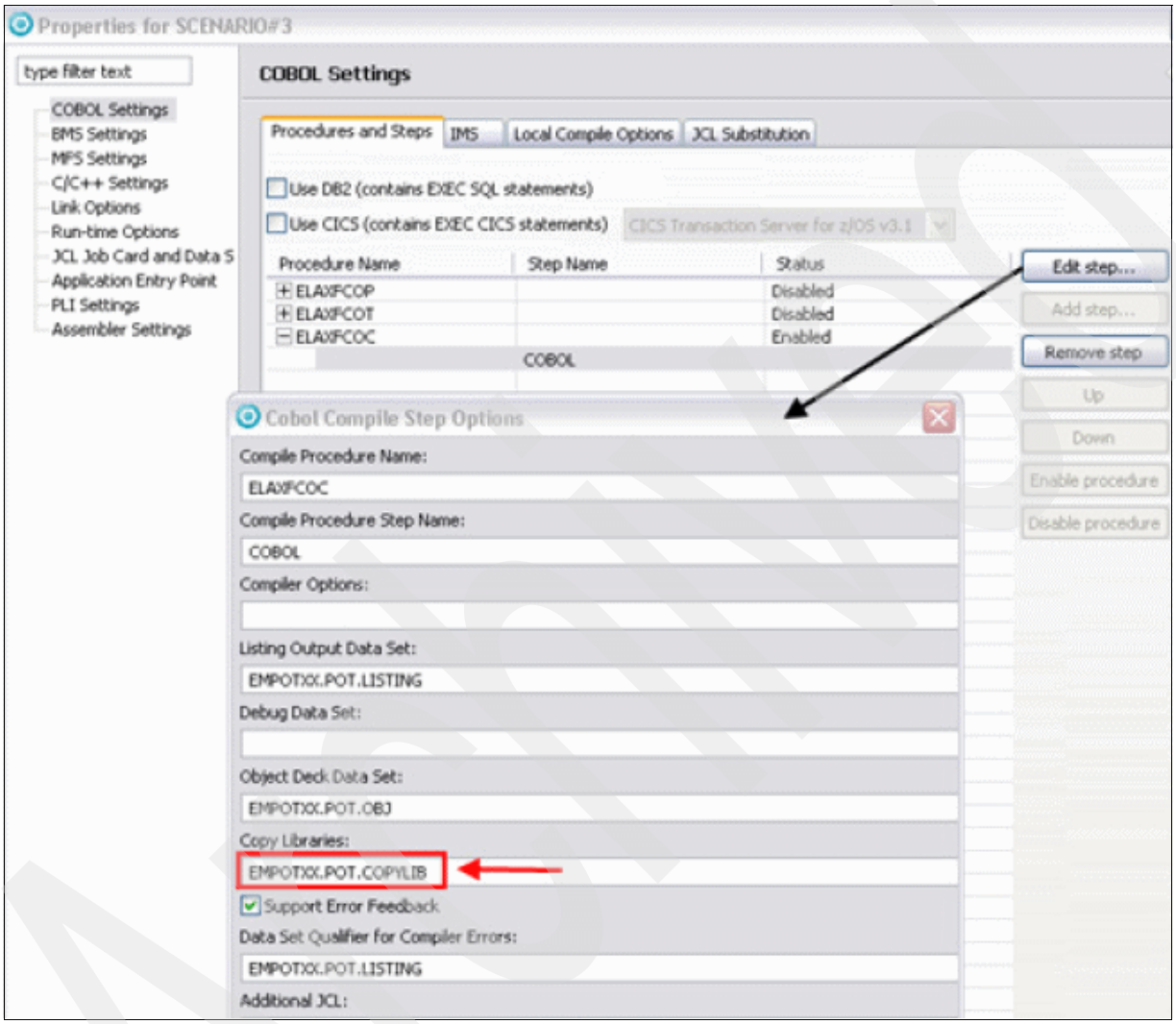


Figure 1-6 COBOL settings that are required for COBOL using copybooks

5. To perform the Remote Syntax check, right-click the **P1VSAM** program and select **Syntax checking** → **Remote**. A job is submitted to z/OS. If you have adjusted the project settings to reflect your environment, you should see no JCL errors; the COBOL program is compiled. Remote syntax checking uses the Developer for System z JCL procedure support to submit resources to a remote compiler for syntax checking.
6. If you use the provided P1VSAM COBOL program, you should see no syntax errors. Warnings are acceptable. To see the statement causing the warning, double-click the warning message listed in the Remote Error List view; you see the COBOL program positioned by the statement that caused the warning. Make sure that the COBOL copybooks were successfully invoked from EMPOTXX.POT.COPYLIB and that the JCL cards were generated during the remote syntax check. You can check the JES queue to see the generated JCL.

1.5.2 Scenario 3.2: Local syntax check using the local compiler

This scenario is similar to the previous one in that you use a z/OS project with the P1VSAM COBOL program added to an MVS subproject. However, in this scenario, you use the local compiler instead of the one on the z/OS system. This work style is preferred because it saves CPU cycles on the z/OS and it can help you achieve greater development productivity.

To perform a local syntax check, the source file you want to check must be in an Developer for System z MVS subproject. The subproject may be in either an offline or online state. In this scenario, it will be in online state, because the user is connected to z/OS.

The P1VSAM program has a copybook, defined in XXXX.POT.COPYLIB and specified in the project properties, as shown in Figure 1-7 on page 18. When you request a local syntax check for a remote COBOL program, Developer for System z searches the remote system for all dependencies (such as copy books) and caches these files in a temporary location in your workspace. The list of files is also stored as a property of the remote file. This search and temporary caching enables Developer for System z to complete a syntax check without requiring you to add the dependent files to your MVS subproject.

Before checking syntax locally, you might need to set local build options. You need to set these options only once and only if you want to override the default values. This procedure defines the default behavior for local syntax check. You can also request Developer for System z to look in your subproject for dependent files. To enable this behavior, select the **z/OS Solutions** and **Syntax Check** options in the preferences window (select **Window** → **Preferences**).

To create the z/OS project and perform a COBOL syntax check using the provided assets and the local compiler, do these steps:

1. Connect to z/OS system.
2. To perform the Local Syntax check using the z/OS projects view, right-click **P1VSAM** and select **Syntax Check** → **Local**.
3. The Check Dependencies window opens, prompting you to refresh dependencies for the remote file.
4. If you have reason to believe that the dependencies have changed since the last syntax check, click **Refresh dependencies and then perform a syntax check**; otherwise, click **Syntax check only**. To review the list of dependencies before proceeding, click **Dependent files**. The window expands to show the list of dependent files and the date and time of the last refresh.

Archive

5. Click **OK**. The local compiler is started to perform the syntax check. After the syntax check is complete, any errors or warnings are displayed in the Remote Error List. In this example, you should have only one warning and no errors. If you see errors, be sure that you have followed the steps above correctly. Figure 1-7 shows the results of the sample program using the local syntax check after you double-click the warning message.

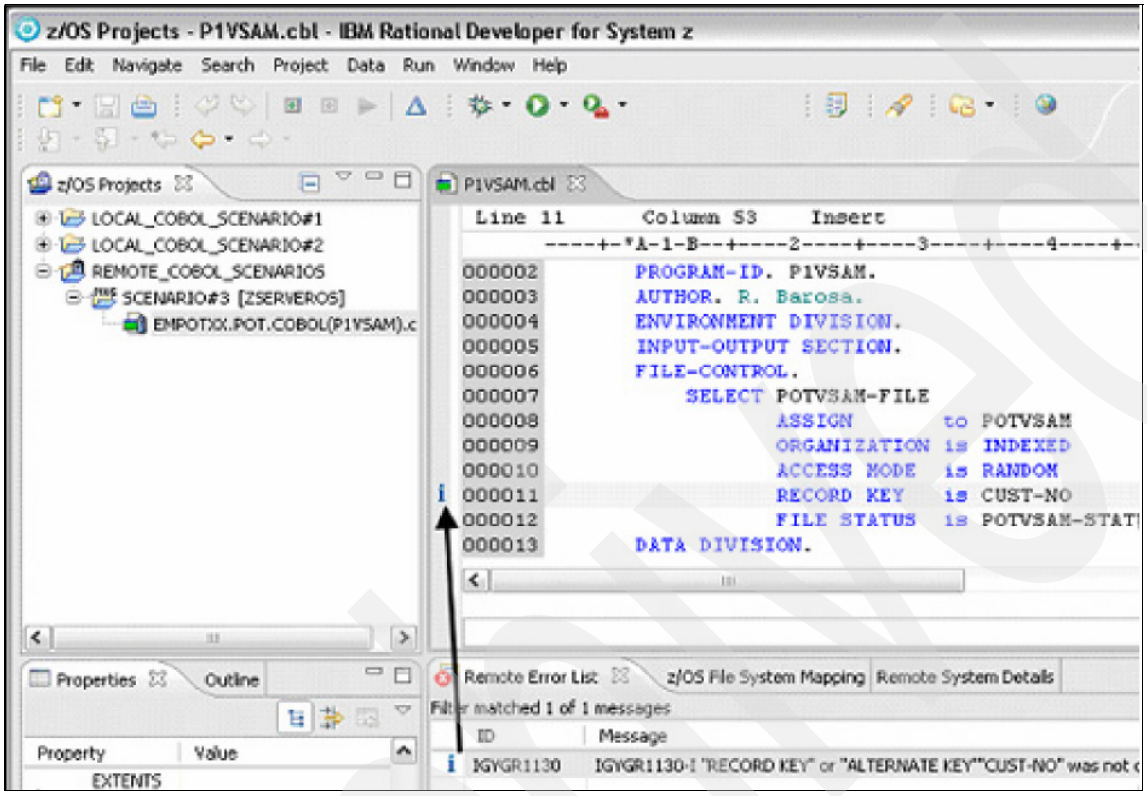


Figure 1-7 Results from local syntax check using a COBOL program stored on z/OS

The following error messages show some common problems with local syntax check of remote files. These messages display in the Refresh Dependencies Problems window. Click the **Details** button to display the specific error message:

- "JES subsystem is not connected. Connect to the subsystem and retry the operation." This message displays if the JES subsystem is not connected.
- "The submitted job has ended with a JCL error. Check the build properties and retry the operation." This message displays if, for example, you have specified a PDS in the SYSLIB that does not exist.

In this scenario, we had similar behavior as though the compilation was done on z/OS, but it was using the local compiler.

1.5.3 Scenario 3.3: Working offline - move to local and use the local compiler

You can work offline in a z/OS project if circumstances do not permit you to remain connected to the remote system. When you move a z/OS project to the offline state, you can select specific subprojects, data sets, and PDS members to be taken offline. When you are ready to work online again, you can reconnect to the remote system and upload the resources you changed.

Working offline in a z/OS project

Before you can move a z/OS project to an offline state, you need to have completed these tasks:

1. Defined a z/OS system and connected to the system in the Remote System view.
2. Created a z/OS project and one or more MVS subprojects in the z/OS Projects view.
3. Populated the MVS subproject(s) with your personal data sets or PDS members as well as some shared copybook data sets.
4. Correctly defined the SYSLIBs for the remote system.

To work offline in a z/OS project, do these steps:

1. Connect to the z/OS system.
2. Using the z/OS Projects perspective and the z/OS Projects view, right-click the project you want moved offline and select **Work Offline**. For this scenario, select the **REMOTE_COBOL_SCENARIOS** project.
3. Select the subprojects, data set members, or files that should be moved offline. In our example, we have only one program named P1VSAM.
4. Remember that this program needs a COBOL copybook and, to avoid compiler errors when working offline, you need to add the dependencies of this COBOL program. To perform this task:
   - In the Working Offline dialog, click **Next**.

Archived

– Click **Show Dependencies**. The dependencies display in the list box, as shown in Figure 1-8. This action submits a job to z/OS, which might take a while to run. If the preferences are set correctly, you see the required COBOL copybook, VSAMCOPY, listed.

– In the dependencies list, select the check box next to those copy members and include files you want to add to the subproject before it goes offline.The checked files are also taken offline. If any of your programs depends on a file that cannot be found, an error message is issued and the file is skipped. To ensure that files are not skipped, run a local syntax check on your program; if the syntax check does not produce any errors, then all the dependencies can be located and no files will be skipped.

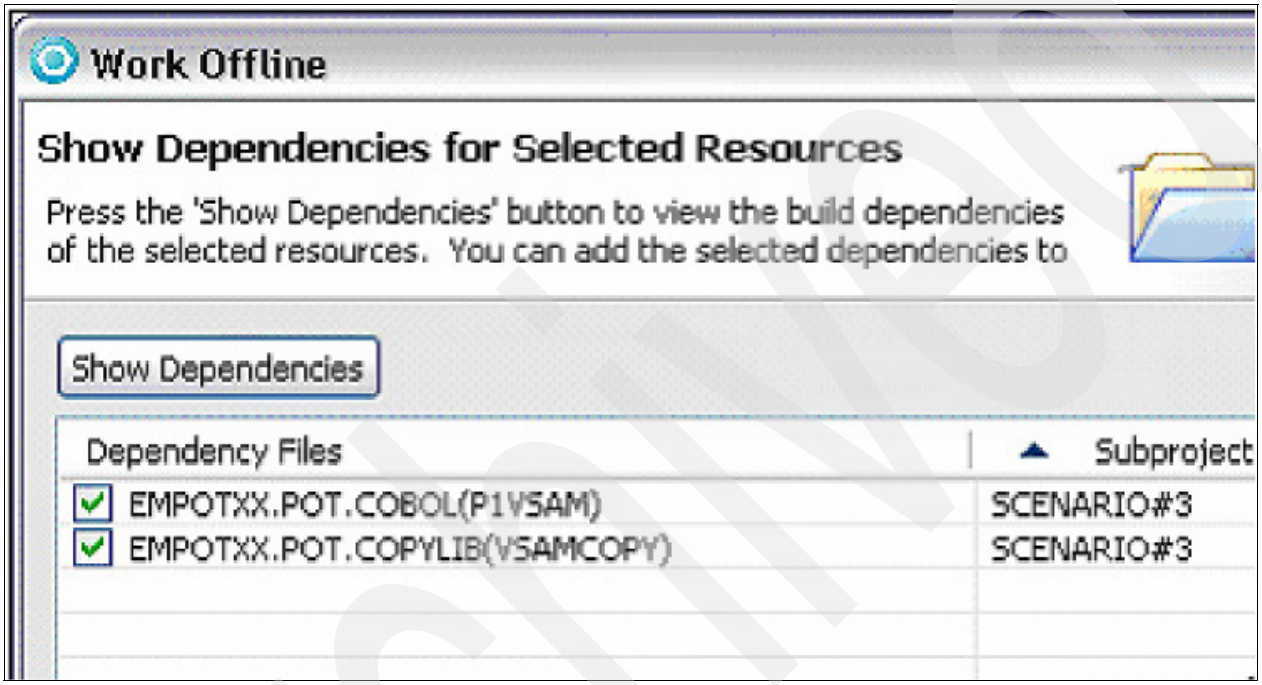


Figure 1-8 Identifying dependent resources for working offline

5. Click **Finish**. The selected resources are taken offline and the project is labeled [Offline] in the z/OS Projects view.
6. You can now disconnect from the remote system and work offline on the downloaded resources.

When the project is taken offline, the library properties used to build a file are transferred to the offline subproject. In addition to transferring the SYSLIB to reflect a local version, the additional JCL is processed into SET statements for the Environment variables field. Translating these properties locally helps you continue development activities such as syntax check, open copy member, and content assist.

The work offline feature is a convenience that lets you edit your source code, check the syntax of the code, and save your local changes when a remote system connection is not available. You *can* perform the following actions on an offline project:

- Create a PDS member.
- Copy and paste a PDS member from another project, subproject, or file system into an offline data set.
- Rename a PDS member.

- Delete a PDS member.
- Edit and make changes to the source or copybook member.
- Local syntax check.

These tasks are similar to those you can do in Scenarios 1 and 2 where you have no connection to z/OS. However, the following special considerations and limitations are in effect when you work offline in a z/OS project:

- Resources on the remote system are still accessible to others. You should take precautions to ensure that others do not overwrite the remote resources while you are working on your local copy.

Important: When you select **Work Online** and choose to overwrite the remote resource, any changes that others have made to the resources on the z/OS system will be lost.

- For MVS, remote operations, including remote syntax check, project builds, allocation or deletion of data sets, creation or deletion of a member, move, copy and paste, rename, save as, generate JCL, debug, compress, and migration of data sets, are not supported when you work offline, even if the system associated with the project is connected. For UNIX® System Services, project builds, file level compile and assemble, and debug are not supported in an offline state.
- New folders and files (sequential data sets) cannot be created in an offline subproject.

Returning a z/OS project to an online state

Important: While a project is offline, resources on the remote system are still accessible to others. Take precautions to ensure that others do not overwrite the remote resources while you are working on your local copy. When you return an offline project to an online state, and choose to overwrite the remote resource, any changes that others have made to the resources on the z/OS system are lost.

To reconnect a z/OS project to the remote system and to work online, do these steps:

1. Connect to the z/OS system.
2. Right-click the project to be moved online and click **Work Online**. If any of the associated systems are not connected, you are prompted to connect to them before the project is moved to an online state. Developer for System z attempts to resynchronize your local resources with the remote system. A progress indicator displays while collecting out-of-sync resources. Resynchronizing can take a few minutes if the offline project is large. After resynchronizing, the Work Online window opens showing the resources that have been changed since the project was taken offline. If you have renamed an offline resource, this change shows up as two separate update operations: a deletion of the old resource and an addition of the newly renamed resource. For this scenario, when the project was offline, we added one comment line to the COBOL program P1VSAM after the procedure division and, to force a possible problem, we also added another comment line on the program to the z/OS version. This simulates the typical case in which someone changes the z/OS program and also the copied version.

Figure 1-9 shows the example after the synchronization.

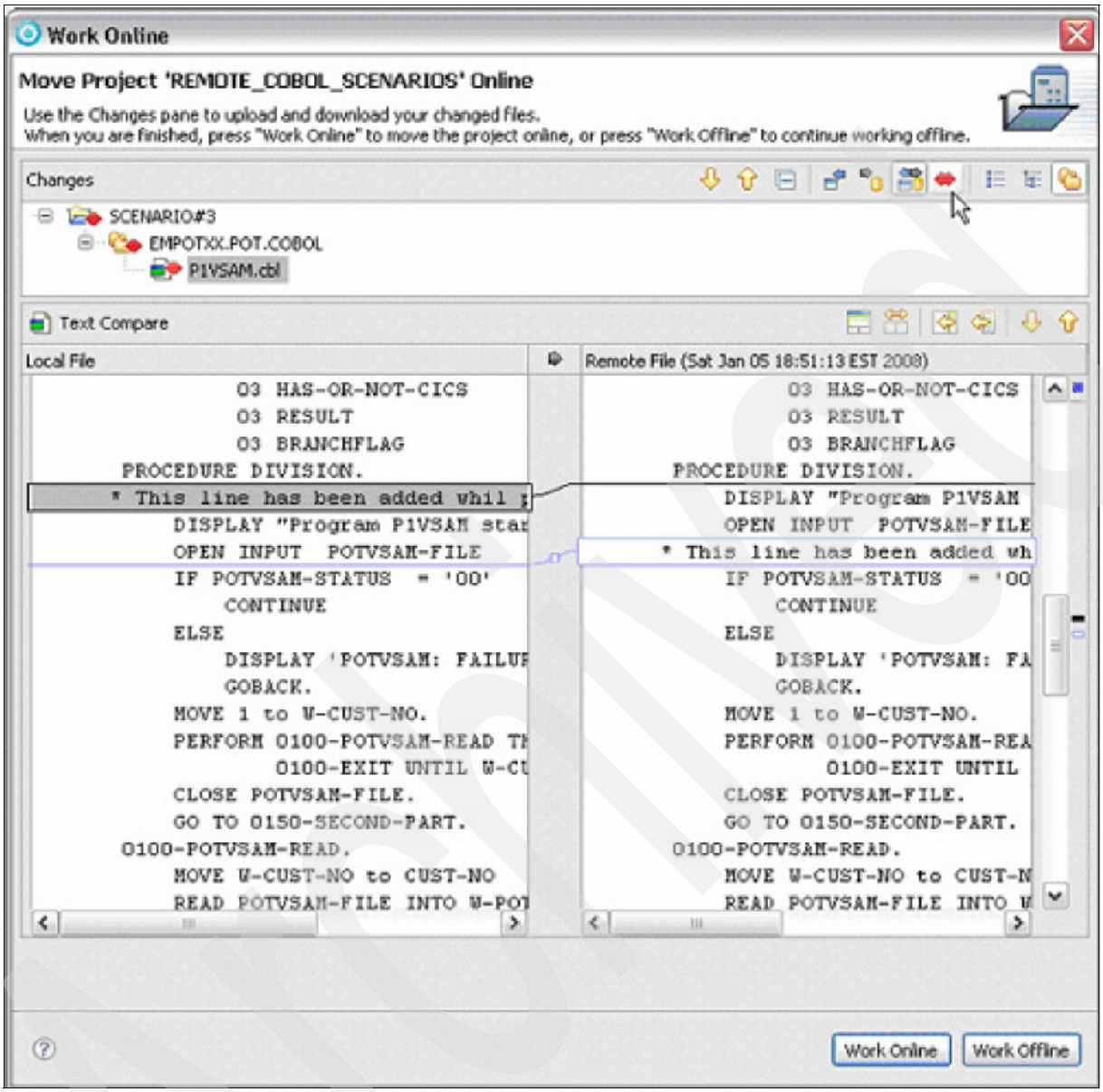


Figure 1-9 Work Online dialog

Tip: If you do not see the results shown in Figure 1-9, you might need to change the Preferences. Make sure that the option **Ignorewhite space** is selected when you select **Preferences** → **General** → **Compare**.

3. You can make updates as follows:
   – To view differences between a local and remote resource, select the resource of interest in the Changes pane. A Text Compare pane opens showing the differences between the local and remote resources (see Figure 1-9 on page 22). You can handle conflicts as follows:
     • If a file is modified both locally and on the host, it is marked with a red diamond. To copy changes from the remote system to your local workspace, click the right arrow button between the two panes.
     • After fixing the conflicts, save the local file by right-clicking it and selecting **Save**.
     • To indicate that all conflicts have been resolved, select the resource in the Changes pane and click **Mark as Merged**.
     • To download a resource that has changed on the remote system, select it and click **Download**.
     • To upload a resource that you have changed locally, select it and click **Upload**.
     • To upload the entire subproject, select the subproject name in the Changes pane and click **Upload**.
4. After you have uploaded or downloaded all the resources you want, click the **Work Online** button. You can work as you did before going offline.

1.6 Scenario 4: Code on z/OS that uses CICS but not DB2

In this scenario, you work with a COBOL program that uses CICS statements but does not access DB2 tables. The COBOL program has two COBOL copybooks with data definitions. To perform syntax check capabilities, the compiler needs to have access to the COBOL copybooks; otherwise, the compiler flags errors indicating missing defined data names.

1.6.1 Scenario 4.1:- Remote syntax check using the z/OS compiler with a remote COBOL/CICS program

For this scenario, you use the P4CICS COBOL program and the COMAREA and VSAMCOPY copybooks. This program has CICS statements, and VSAM I/Os on z/OS; it does not access DB2. Part of this program is shown in Example 1-3.

Example 1-3 P4CICS program used in CICS scenarios

```
IDENTIFICATION DIVISION.
PROGRAM-ID. P4CICS.
AUTHOR. Reginaldo Barosa.
INSTALLATION. IBM Dallas.
DATE-WRITTEN. 01/21/07
***************************************************************
* Receive customer #, read VSAM, move data to COMMAREA
*
* ENVIRONMENT : CICS, COBOL II
*
* CICS TRANSACTION NAME : NONE
*
* VSAM FILES: EMPOTXX.RDZV7.POT
*
***************************************************************
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-PROGRAM PIC X(08) VALUE 'P4CICS'.
01 WS-LITERAL-WS PIC X(48) VALUE
' WORKING STORAGE STARTS HERE'.
01 WORK-VARIABLES.
05 WS-RESP PIC S9(8) VALUE 0 COMP.
* Below is the VSAM record area
COPY VSAMCOPY.
01 WS-LITERAL PIC X(48) VALUE
' LINKAGE SECTION STARTS HERE'.
LINKAGE SECTION.
01 DFHCOMMAREA.
COPY COMAREA.
PROCEDURE DIVISION.
0100-MAIN-ROUTINE.
move CustNo to xxx.
IF CustNo > 10 OR CustNo < 1 THEN
MOVE CustNo TO CUST-NO
MOVE SPACES TO DFHCOMMAREA
MOVE CUST-NO TO CustNo
MOVE 'Should be > 0 and < 11' TO LastName
MOVE -1 to RetCode
EXEC CICS RETURN
END-EXEC.
0300-POTVSAM-READ.
MOVE CustNo TO CUST-NO .
EXEC CICS READ
FILE ('POTVSAM')
INTO ( POTVSAM-RECORD-REC )
LENGTH (LENGTH OF POTVSAM-RECORD-REC )
RIDFLD ( CUST-NO )
EQUAL
RESP (WS-RESP)
END-EXEC.
0400-CHECK-IFOK.
IF WS-RESP NOT = DFHRESP(NORMAL)
MOVE SPACES TO DFHCOMMAREA
MOVE 'INVALID READ POTVSAM ' TO LastName
EXEC CICS RETURN
END-EXEC.
IF WS-RESP = DFHRESP(NOTFND)
MOVE SPACES TO DFHCOMMAREA
MOVE 'POTVSAM NOT FOUND ' TO LastName
EXEC CICS RETURN
END-EXEC.
0500-DATA-RETURNED-OK.
MOVE CUST-LN TO LastName.
MOVE CUST-FN TO FirstName.
MOVE CUST-ADDR1 TO Address1 .
MOVE CUST-CITY TO City .
MOVE CUST-ST TO State .
MOVE CUST-CTRY TO Country .
```

Archived

```
MOVE ZEROES TO RetCode .
EXEC CICS RETURN
END-EXEC.
```

This program has a error. The statement `move CustNoto xxx.` is incorrect because the field “xxx” is not defined. When you compile this program, this line must be flagged as an error.

This scenario assumes that you have already implemented the scenario with remote assets (see 1.5.1, “Scenario 3.1: Remote syntax check using the z/OS compiler” on page 15). Therefore, you should have already allocated the required z/OS PDSs and moved the P4CICS COBOL program to XXXX.POT.COBOL and the COMAREA and VSAMCOPY copybooks to XXXX.POT.COPYLIB.

To create the z/OS project and perform a COBOL syntax check using the provided assets, do these steps:

1. Connect to z/OS.
2. Expand **MVS Files** to see the data sets that you allocated earlier.
3. Using any text editor, examine and adjust for the environment the properties that are defined in SCENARIO#4_MVS_PROPERTIES.xml; these properties are required for this scenario. For example, the CICS library defined in our example is CICSTS.V3R1.CICS.SDFHLOAD.
4. Using the Remote Systems view, right-click **MVS Files** and import SCENARIO#4 MVS PROPERTIES.xml.
5. Create an MVS subproject named SCENARIO#4 under the previously created REMOTE_COBOL_SCENARIOS project. Add the P4CICS COBOL member located at EMPOTXX.POT.COBOL to this subproject.

Archivo

Tip: The data set with the required COBOL copybooks named VSAMCOPY and COMAREA do not need to be part of the project. If you adjusted the environment properties, the imported properties should be adjusted for CICS. The Use CICS option is specified, as well as the data set where we have the COBOL copybooks (EMPOTXX.POT.COPYLIB), and CICS Copy Libraries are listed in the properties. Figure 1-10 on page 26 shows the required COBOL settings to perform our scenario.

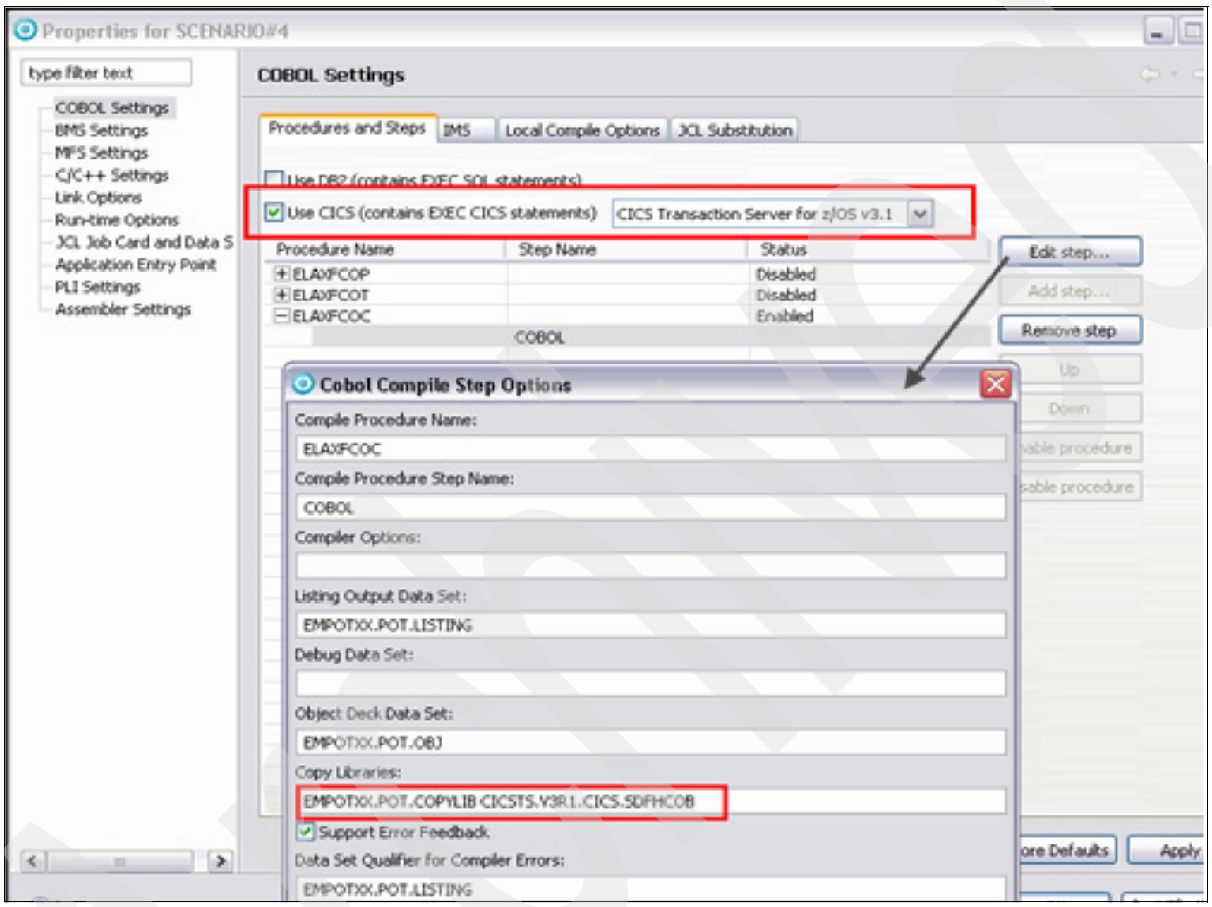


Figure 1-10 COBOL settings for a remote program with CICS

6. To perform the Remote Syntax check, right-click the program **P4CICS** and select **Syntax checking** → **Remote**. A job is submitted to z/OS. If you successfully specified the project settings to reflect your environment, you should have no JCL errors, and the COBOL program is compiled. Remote syntax checking uses the Developer for System z JCL procedure support to submit resources to a remote compiler for syntax checking.

Using the P4CICS COBOL program, you see a syntax error due to the bad statement mentioned earlier (`move CustNo to xxx`). Double-click the warning message listed in the Remote Error List view to edit the COBOL program, which is next to the statement causing this warning.

COBOL copybooks were successfully invoked from EMPOTXX.POT.COPYLIB and the JCL cards that were generated during the Remote Syntax check. Check the JES queue to see the generated and submitted JCL.

1.6.2 Scenario 4.2: Local syntax check with a remote COBOL/CICS program

This scenario is similar to the previous one; you have created a z/OS project and added the P4CICS COBOL program to an MVS subproject, but you use the local compiler capabilities instead of the one on the z/OS system. This is the suggested scenario because it saves z/OS CPU and can increase your development productivity.

To perform a local syntax check, the source file you want to check must be in an MVS subproject. The subproject may be in either an offline or online state. In this scenario, it is in an online state because you are connected to z/OS. The P4CICS program has two copybooks that are stored in XXXX.POT.COPYLIB, which is specified in the project properties, shown in Figure 1-10 on page 26.

In this scenario, the copybooks do not have CICS statements, so it does not require the CICS pre-processor. Section 1.6.3, “Scenario 4.3: Local syntax check using local compiler, remote COBOL/CICS program, and copybook with EXEC CICS” on page 29 shows how to handle the case where you have a copybook with CICS statements.

When you request a local syntax check for a remote COBOL program, Developer for System z searches the remote system for all dependencies (such as copybooks) and caches these files in a temporary location in your workspace. The list of files is also stored as a property of the remote file. This search and temporary caching enables Developer for System z to complete a syntax check without requiring you to add the dependent files to your MVS subproject.

Before checking syntax locally, you may need to set local build options. You need to set these options only once and only if you want to override the default values. This procedure outlines the default behavior for local syntax check. You can also request that Developer for System z to look in your subproject for dependent files by selecting the **z/OS Solutions** and **Syntax Check** option in the Preferences window.

To create the z/OS project and perform a COBOL syntax check using the provided assets on the local compiler, do these steps:

1. Connect to the z/OS system.
2. To perform the Local Syntax check, using the z/OS projects view, right-click **P4CICS** and select **Syntax Check** → **Local**.

Archive

3. The Check Dependencies window opens, prompting you to refresh dependencies for the remote file. Click **Refresh dependencies and then perform a syntax check.** To review the list of dependencies before proceeding, click **Dependent files**. The window expands to show the list of dependent files and the date and time of the last refresh. Figure 1-11 shows the result of this dialog. You usually need to refresh at least once. For the next syntax check, you can select **Syntax check only**.

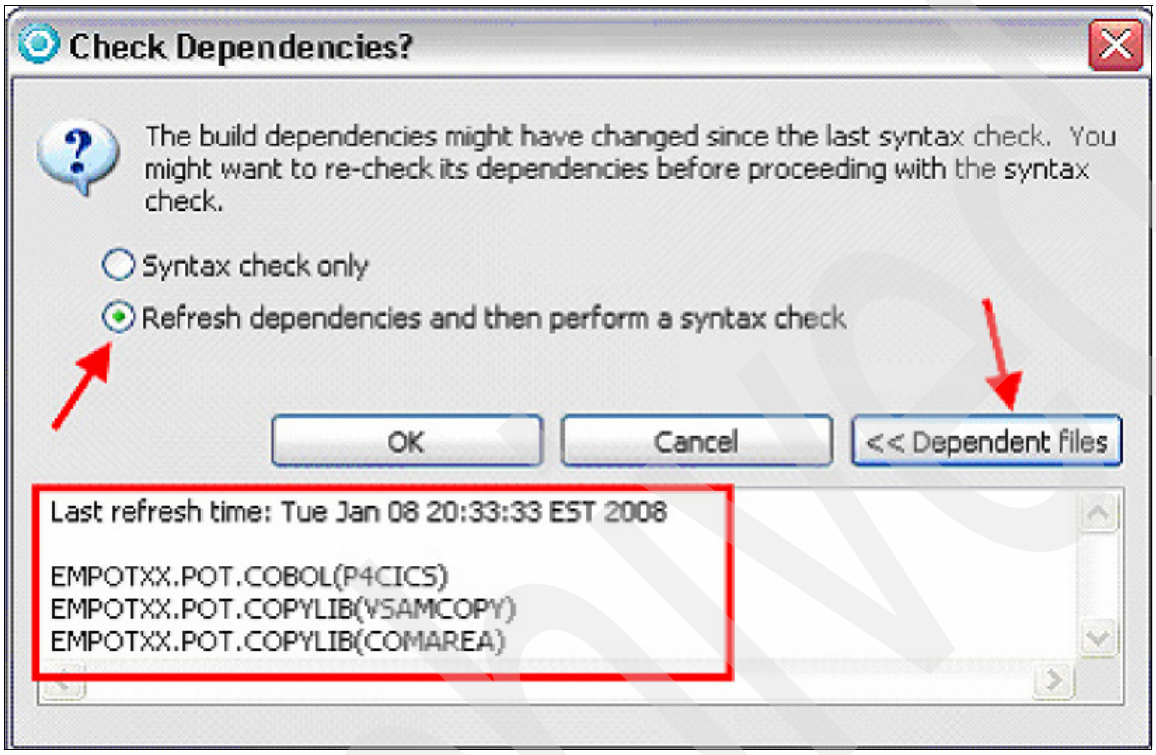


Figure 1-11 Dependent files for program P4CICS

4. Click **OK**. The local compiler starts and performs the syntax check. When the syntax check is completed, any errors or warnings display in the Remote Error List. In this example, you should have only one warning and no errors. If you get errors, make sure that you have followed the steps above correctly.
5. Double-click the warning message. Figure 1-12 on page 29 shows the results of the sample program using the local syntax check.

The following error messages represent some common problems with a local syntax check of remote files. These messages may display in the Refresh Dependencies Problems window. Click the **Details** button to display the specific error message:

- "JES subsystem is not connected. Connect to the subsystem and retry the operation." This message might display if the JES subsystem is not connected.
- "The submitted job has ended with a JCL error. Check the build properties and retry the operation." This message might display if, for example, you have specified a PDS that does not exist in the SYSLIB.

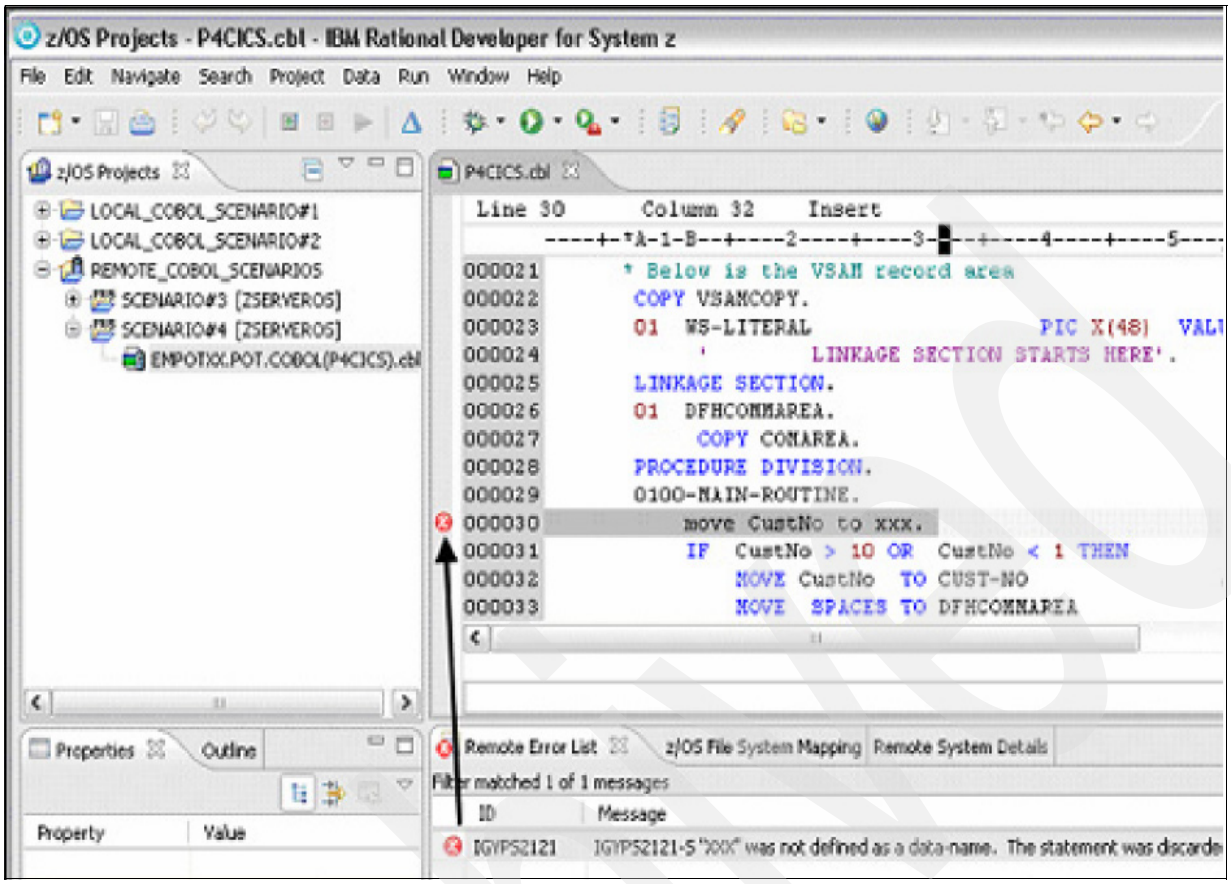


Figure 1-12 Results from local syntax using a remote COBOL program with CICS

In this scenario, you saw behavior similar to that when the compilation was run on z/OS; however, this scenario used the local compiler.

1.6.3 Scenario 4.3: Local syntax check using local compiler, remote COBOL/CICS program, and copybook with EXEC CICS

In this scenario, the developer is working with a COBOL program that uses CICS statements but does not access DB2 tables. The COBOL program has two COBOL copybooks with data definitions and another COPYBOOK that has EXEC CICS statements. To perform the syntax check, the compiler needs to have access to the COBOL copybooks; otherwise, the compiler will flag errors for missing defined data names. Also, because the program uses copybooks with CICS statements, those statements need to have been interpreted by the CICS translator before the compilation. For this scenario, you use the P43CICS COBOL program and the CUSTREC, CUSTIORQ and COPYCICS copybooks. This program has CICS statements but no DB2 statements; the COPYCICS copybook has EXEC CICS statements. Part of this program is shown in Example 1-4.

Example 1-4 P43CICS program used in CICS scenarios

```
IDENTIFICATION DIVISION.
PROGRAM-ID. P43CICS.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WORK-AREAS.
```

```
05 RESP-FLD PIC S9(8) COMP.
05 RESP-FLD-TWO PIC S9(8) COMP.
05 APP-ID.
10 Filler PIC X(9) VALUE
'P43CICS_'.
10 The-Date PIC X(8).
10 Filler PIC XX Value ': '.
05 CSMT-MSG PIC X(121).
05 DISPLAY-NUMBER PIC 9999.
05 END-DATA PIC XXX VALUE 'NO'.
88 END-OF-DATA Value 'YES'.
05 Start-Index PIC S9(8) COMP.
05 Size-Index PIC S9(8) COMP.
05 Abs-Time PIC S9(15) COMP-3.
05 The-Time PIC X(8).
COPY CUSTREC .
LINKAGE SECTION.
01 DFHCOMMAREA.
COPY CUSTIORQ .
PROCEDURE DIVISION.
Perform p8400-Get-Format-Current-Time.
COPY COPYCICS.
MOVE '00' TO RET-CODE.
String APP-ID 'Contents of input COMMAREA is:'
DELIMITED BY SIZE INTO CSMT-MSG.
PERFORM P7200-WRITE-TO-CSMT-QUEUE.
….…
….
p8400-Get-Format-Current-Time.
EXEC CICS ASKTIME ABSTIME(Abs-Time) END-EXEC.
EXEC CICS FORMATTIME ABSTIME(Abs-Time)
DATE(The-Date) DATESEP('/')
TIME(The-Time) TIMESEP(':')
END-EXEC.
P9000-RETURN-TO-CICS.
EXEC CICS RETURN END-EXEC.
```

This scenario assumes that you have already implemented the scenario with remote assets (see 1.5.1, “Scenario 3.1: Remote syntax check using the z/OS compiler” on page 15); therefore, you have already allocated the required z/OS PDSs, and moved the P43CICS COBOL program to XXXX.POT.COBOL and the COMAREA copybook to XXXX.POT.COPYLIB.

To create the z/OS project and perform a COBOL syntax check using the provided assets, do these steps:

1. Connect to z/OS.
2. Expand **MVS Files** to see the data sets that you allocated earlier.
3. Using the Remote Systems view, right-click **MVS Files** and import the required properties defined for this scenario. These properties are defined in SCENARIO#4_MVS_PROPERTIES.xml. Adjust these properties for your environment. For example, the CICS library defined in this example is CICSTS.V3R1.CICS.SDFHLOAD. The generated JCL feature of Rational Developer for System z uses those properties as a model.

Archived

4. Create an MVS subproject named SCENARIO#43 under the previously created REMOTE_COBOL_SCENARIOS project. Add the P43CICS COBOL member located in EMPOTXX.POT.COBOL to this subproject. (The data set that holds the COBOL required copybooks named CUSTREC, CUSTIORQ, and COPYCICS does not need to be part of the project.)
5. To perform the Local Syntax check, using the z/OS projects view, right-click **P43CICS** and select **Syntax Check** → **Local**.
6. The Check Dependencies window opens, prompting you to refresh dependencies for the remote file. Click **Refresh dependencies and then perform a syntax check.** To review the list of dependencies before proceeding, click **Dependent files**. The window expands to show the list of dependent files and the date and time of the last refresh. Figure 1-13 shows the results. You usually need to refresh the dependencies at least once; after that, you can select **Syntax check only**.

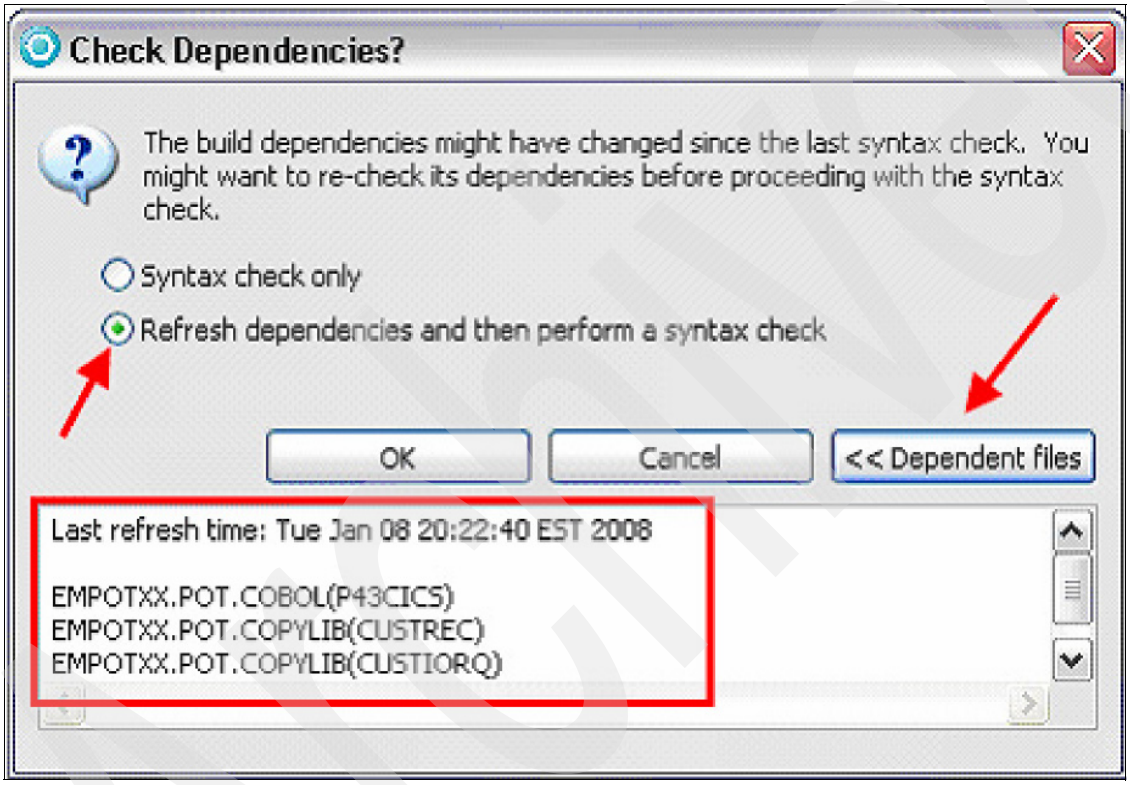


Figure 1-13 Dependencies for the P43CICS program

1.7 Conclusion

Rational Developer for System z can help developers that are working with COBOL (as well PL/I) programs to be more productive. Instead of submitting compilations to z/OS, you can use the local compiler. When you use the z/OS compiler, you can easily find the syntax errors in the Remote Error List view.

Archived

2

Scenario for problem determination tools with Rational Developer for System z

In this chapter, we introduce the IBM tools that support application development on System z:

- Application Performance Analyzer for z/OS
- Debug Tool for z/OS and Debug Tool Utilities and Advanced Functions
- Fault Analyzer for z/OS
- File Manager for z/OS
- Optim™ MOVE
- Workload Simulator
- Rational Developer for System z

These tools provide you with a comprehensive suite of products that can help improve application delivery and enhance production availability and performance. The collection of tools for this chapter are referred to as the problem determination tools (PD Tools).

© Copyright IBM Corp. 2009. All rights reserved.

2.1 Introduction

As businesses and the critical IT systems that support them grow increasingly complex, application developers are constantly struggling to meet the demands placed upon them. Service-oriented architecture (SOA) has become the hottest topic in IT today, because it holds the promise of addressing these demands. The ability to reuse existing assets is the cornerstone of SOA. This possibility is significant because reusing assets can be significantly less expensive than rewriting them. With the vast amount of existing applications running on the IBM System z platform, it only makes sense that System z applications can be a big part of the move to SOA.

IBM Application Performance Analyzer for z/OS, IBM Debug Tool Utilities and Advanced Functions for z/OS, IBM Fault Analyzer for z/OS, and IBM File Manager for z/OS, along with Optim MOVE and Rational Developer for System z, provide a robust suite of problem determination tools that can help improve application delivery throughout the application life cycle.

You can use these tools to help increase productivity and IT effectiveness across source code generation and debugging, application abend analysis, data management, and application-performance analysis. IBM problem determination tools do much more than support traditional applications. They include capabilities that enable you to build SOA applications. They are also tightly integrated with other tools in the IBM problem determination tools portfolio. All the tools also continue to support and make the most of the latest subsystem levels.

2.2 IBM Application Performance Analyzer for z/OS

Application Performance Analyzer for z/OS helps programmers and systems personnel to identify constraints and to improve the performance of their applications. It is a non-intrusive performance analyzer that helps you with design, development, production, and maintenance cycles.

The product's key function is to measure and report on how system resources are used by applications running in a z/OS address space, such as Time Sharing Option (TSO) and batch, and online subsystems, such as IMS, CICS, or WebSphere Application Server, as well as DB2 stored procedures.

With a single interface, you can monitor applications in test, as well as in production, and in multiple source languages, including Assembler, C/C++, COBOL, PL/I, and Java™. Optimized code support for COBOL and PL/I is provided to enable you to monitor production applications.

Archive

2.2.1 Highlights

Here we summarize the main features of Application Performance Analyzer:

- Provides an easy-to-use function that helps isolate application performance problems.
- Helps pinpoint performance bottlenecks affecting online transaction response times.
- Assists in reducing batch-application turnaround time.
- Supports Assembler, C/C++, CICS, COBOL, DB2, IMS, Java, PL/I, WebSphere MQ, and WebSphere Application Server, including all the latest versions.
- Increases application understanding during stress and regression testing.
- Shares side files with Fault Analyzer and Debug Tool.
- Provides support for UNIX System Services.
- Provides support for IBM Parallel Sysplex® Coupling Facility.
- Can be invoked from other programs, such as IBM Tivoli® OMEGAMON® and IBM Tivoli Enterprise Portal.
- Provides the ability to compare two observation reports to see the relevant differences.
- Supports threshold monitoring.
- Supports Java source mapping, along with support for a wide range of Java applications.
- Provides recommendations that can help you to improve the performance of your DB2 applications.

Using Application Performance Analyzer helps you maximize the performance of your existing hardware resources and helps you improve the performance of your applications and subsystems. It lets you evaluate application prototypes in the design phase, review the impact of increased data volume or changes in business requirements on performance, and generate historical data and reports to analyze performance trends and evaluate program changes for cost-effectiveness.

Running in a separate address space, Application Performance Analyzer non-intrusively collects resource utilization, wait time, and statistical samples from the address space being monitored. This data is then analyzed and documented. The reports generated help you to identify the key performance bottlenecks that require examination and resolution. This data is available in online and printed reports that you can choose to create as PDF files for viewing on a workstation.

Archived

2.2.2 Support for IBM subsystems

Application Performance Analyzer provides application-monitoring support for all of the major IBM subsystems, as shown in Figure 2-1.

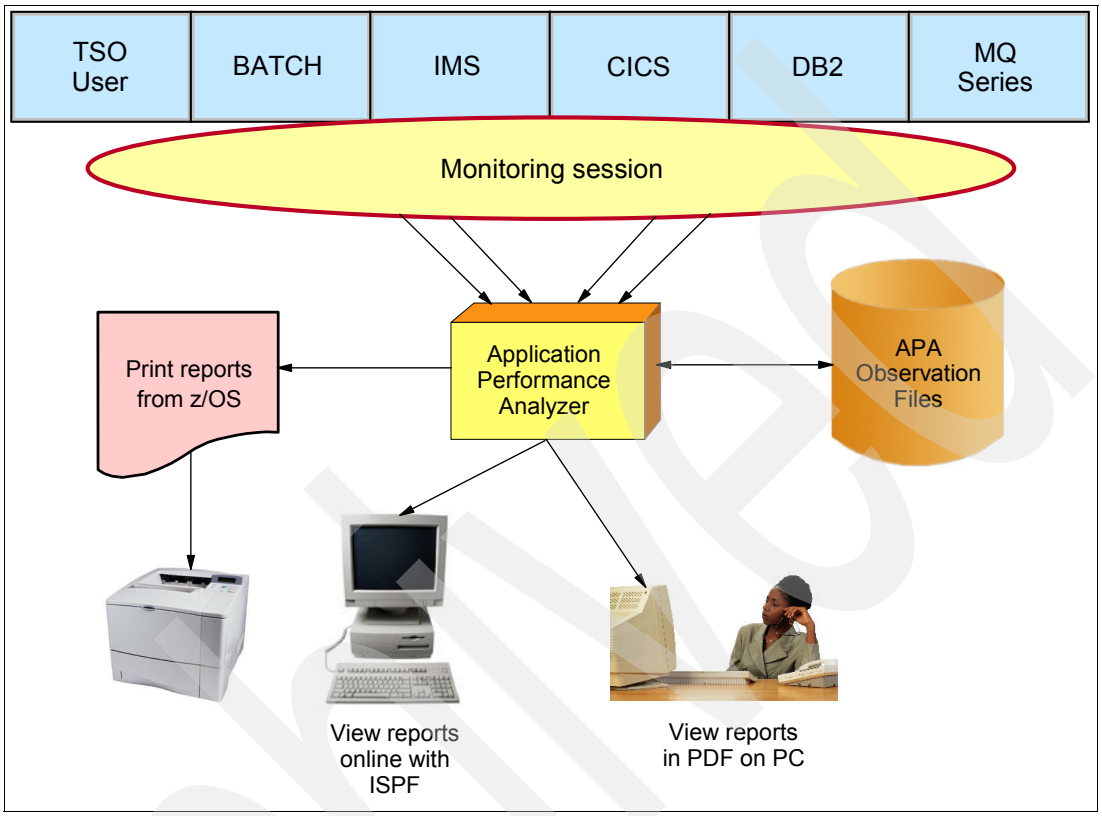


Figure 2-1 The APA environment

CICS

Support for IBM CICS Transaction Server enables you to monitor specific CICS transactions using wildcard transaction prefixes or by termID selection. With this support, you can trace transactions during critical situations, rather than waiting to review data that is collected periodically. Java application code running in the Java 2 Platform Standard Edition (J2SE™) environment under CICS Transaction Server is also supported.

DB2

Support for DB2 delivers relevant information for performance analysis and tuning, including SQL statements and processor usage by SQL statement as well as for IBM DB2 stored procedures written in a traditional language or in Java.

IMS

Support for IMS applications means that you can have IMS application-performance data-on-call time and service-call time for DL/I. You can also choose to trace all IMS calls.

WebSphere MQ

WebSphere MQ support provides information about CPU usage by queue, by request, and by transaction. Application Performance Analyzer also provides service time by queue, request and transaction, as well as wait time, for the same categories.

2.3 IBM Debug Tool Utilities and Advanced Functions for z/OS

Debug Tool Utilities and Advanced Functions for z/OS provides debugging capability for applications running in a variety of environments, such as IBM CICS, IBM IMS, IBM DB2 stored procedures, and IBM UNIX System Services. To meet the requirements of IBM DB2 Universal Database™ for z/OS, Versions 8 and 9 and IBM CICS Transaction Server for z/OS, Version 3, Debug Tool Utilities and Advanced Functions also includes features to help you identify old OS/VS and VS COBOL II source code and to upgrade the code to IBM Enterprise COBOL.

To effectively build and service applications, you require robust, easy-to-use tools to compile, test, and debug them. IBM Debug Tool Utilities and Advanced Functions for z/OS Version 8.1 software provides a complete solution that can help you reduce application development cycle times. IBM Debug Tool for z/OS is included with Debug Tool Utilities and Advanced Functions for z/OS.

2.3.1 Highlights

Here we summarize the main features of IBM Debug Tool Utilities and Advanced Functions for z/OS:

- Provides a single debugging tool for batch, TSO, CICS, DB2, DB2 stored procedures, and IMS applications written in COBOL, PL/I, C/C++, Assembler, and Java.
- Offers more productivity enhancements when used with IBM Rational Developer for System z. With Debug Tool Utilities and Advanced Functions, you can use Rational Developer for System z to provide a fully integrated development, test, and debugging environment for all applications running on z/OS, whether traditional or Web-based.
- Includes tools to identify OS/VS COBOL code and convert it to the ANSI 85 standard.
- Supplies tools to help you determine how thoroughly your code has been tested.
- Helps you debug an application while it runs in a host environment, such as a batch application, TSO, Interactive System Productivity Facility (ISPF), CICS, IMS, or DB2 Universal Database (including IBM DB2 stored procedures) environments. Debug Tool Utilities and Advanced Functions can help you debug almost any application in almost any host environment, including COBOL, PL/I, C/C++, and Assembler applications running on z/OS systems. With Debug Tool Utilities and Advanced Functions, you can compile and link your COBOL, PL/I, C and C++, and Java programs, and assemble and link Assembler programs, as well as preprocessing and compiling your CICS and DB2 programs.
- Provides a CICS utility transaction that enables you to control debugging in the CICS environment.
- Provides an interactive, full-screen 3270 system-based terminal interface with four windows that enable single-step debugging, dynamic patching, and breakpoints:
  - A monitor window displays the status of items you select, variables, and registers. You can view, monitor, and alter application variables or storage in real time.
  - A source window displays the program code, highlighting the statement being run. In the prefix area of this window, you can enter commands to set, display, and remove breakpoints.
  - A log window records and displays your interactions with Debug Tool and can show program output. The information that you see in this window is included in the log file.

  - A memory window (swappable with the log window) that helps you display and scroll through sections of memory. You can update memory by typing over existing data with new data. The memory window keeps track of addresses for easier navigation.

- Using the Debug Tool Utilities and Advanced Functions source window, you can monitor application code while an application runs. You can also debug applications written in a mix of COBOL, C, C++, PL/I, or Java languages without leaving the tool. You can include Assembler programs in this mix and, using the disassembly view, you can debug programs compiled with the NOTEST compiler option or applications that include other languages. You can use commands to dynamically patch or alter the value of variables and structures and to control the flow of an application.

With Debug Tool Utilities and Advanced Functions, you can debug Enterprise COBOL and Enterprise PL/I applications that have been compiled with standard or full-optimization compiler options. You can also analyze your load modules to help you identify candidate OS/VS COBOL programs for conversion and then convert these OS/VS COBOL applications to Enterprise COBOL. These applications then can be compiled and debugged to extend the life of your existing code. Debug Tool Utilities and Advanced Functions software also provides coverage tools that enable you to conduct analysis on your test cases to determine how thoroughly they validate your programs.

2.4 IBM Fault Analyzer for z/OS

Fault Analyzer for z/OS provides the information you require to determine the cause, and assist with the resolution, of application and system failures. Integrated support for Java and IBM WebSphere Application Server for z/OS gives Fault Analyzer expanded application coverage and related business value beyond traditional applications. You can use this one tool to assist in composite-application abend analysis, including 64-bit DB2 Universal Database support. It helps you repair failures quickly by gathering information about an application and its environment at the time of failure.

2.4.1 Highlights

Here we summarize the main features of Fault Analyzer for z/OS:

- Provides a detailed report about program failures to help you resolve them quickly.
- Includes a fault-history file that enables you to track and manage application failures and fault reports.
- Offers a view of storage contents, trace tables, and terminal screen images at the time of failure to help speed corrective action.
- Provides the ability to customize message descriptions to be used in application-failure reports.
- Supports all z/OS subsystems and compilers.
- Provides integration with Rational Developer for System z.

When an application abend occurs, Fault Analyzer captures and analyzes real-time information about the application and its environment, then generates an analysis report detailing the cause of the failure. The report describes the failure in terms of the application code, so you no longer lose time reviewing cumbersome, low-level system error messages. Fault Analyzer allows you to choose a report format to locate the information more easily.

Archive

Each application abend is recorded by Fault Analyzer in a fault-history file by job name, failure code, and other details, along with the analysis report and storage pages referenced during the analysis. This information can later be retrieved to reanalyze the failure.

Through the inclusion of softcopy versions of selected manuals, Fault Analyzer can extract message and failure-code descriptions and insert them into the analysis report where applicable. You can also provide your own descriptions for messages.

You can write your own user exits. For example, you can write a user exit to access compiler listings that are compressed or available only through a proprietary access method.

Integration with Rational Developer for System z allows application developers to work with fault entries directly from their development environment, and also allows Debug Tool and Fault Analyzer to share common source files without redundancy.

2.4.2 Support for IBM subsystems

In the following sections, we discuss the support available for IBM subsystems.

CICS

Fault Analyzer functions are available from within the CICS transaction-based environment in a manner that is consistent with the ISPF usage, permitting you to review and analyze fault entries in history files without the necessity for a TSO logon.

The ability of Fault Analyzer to detect and analyze dump formatting and storage violations is particularly helpful for system programmers who have to debug CICS system problems. Options are provided to speed processing of duplicate dumps and to skip analysis of repeated abends.

DB2

Details of the last SQL activity is provided, along with plan and security information.

IMS

Fault Analyzer provides general information about an IMS region along with details of the last DL/I call parameter list, information for all PCBs in the order of their relative PCB number, and, if available, JCB call trace information. IMS accounting information is also provided.

You can optimize the processing of dumps within the IMS environment, and speed up the processing of duplicate dumps.

WebSphere MQ

API information and return-code diagnostics are provided to assist in problem determination of WebSphere MQ applications.

WebSphere Application Server

Support for the latest versions of WebSphere Application Server and Java is provided.

Archived

2.4.3 Analysis options

Fault Analyzer provides three modes to help you better track and analyze application and system failure information.

Real-time analysis

When an application failure occurs, the Fault Analyzer exit starts real-time analysis. After failure processing, you can view the analysis report in the job output or through the Fault Analyzer ISPF or RD/z interface.

Batch reanalysis

Batch reanalysis generates a new analysis report. This report is based on the dump and information gathered in real time, but with potentially different options specified, or with compiler listings or side files made available. You can submit a Fault Analyzer batch-mode job using either the Fault Analyzer ISPF or RD/z interface or your own job control language.

Interactive reanalysis

Interactive reanalysis runs under ISPF and enables you to navigate through a formatted, structured view of a fully detailed reanalysis. This Fault Analyzer mode enables you to view working storage and control blocks at the time the dump was written. The interface has many point-and-click fields for easy navigation through the interactive reports.

2.5 IBM File Manager for z/OS

File Manager for z/OS offers comprehensive, user-friendly tools for working with z/OS data sets, CICS resources, DB2 data, and IMS data.

Extending the standard browse, edit, copy, and print utilities of ISPF, File Manager not only contains tools that support personnel will find useful, but also delivers enhancements that are designed to address the requirements of application developers working with files containing structured data. Also, because the basic features of the File Manager editor and the ISPF/PDF editor are deliberately almost identical, you can take advantage of extra features without having to relearn fundamental skills.

2.5.1 Highlights

Here we summarize the main features of File Manager for z/OS:

- Includes a File Manager editor that maintains the same general look and feel across both view (read-only) and edit functionality in all of the key areas, whether they are traditional z/OS data sets, CICS resources, or data stored in a DB2 or IMS database.
- Provides the ability to define record and field properties in the same manner as your application programs, that is, by using your existing COBOL and PL/I copybooks stored in partitioned data sets or library-management systems.
- Enhances usability by enabling you to customize which fields to display, copy, or print.
- Enhances usability by providing the ability to define fields describing your data even when you do not have an existing copybook by using dynamic templates.
- Supports development and production through extensive editing, browsing, and batch and interactive utility capabilities.

- Provides the ability to find, create, browse, copy, edit, print, format, or reformat data files in the most-popular z/OS file formats. Not only can the same data mapping be applied as your applications use, but by using File Managers templates, filtering can easily be applied to these operations based on the content of fields in your data. The template concepts of identification and selection criteria provide a simple to use, yet flexible interface for providing the simplest to the more complex of criteria.
- Provides data scrambling support to provide for certain forms of data protection.
- Provides integration with Rational Developer for System z.

2.5.2 File Manager components

In this section, we describe the main components of File Manager.

Base component

The base component manages z/OS data sets, such as queued sequential access method (QSAM), VSAM, partitioned data sets (PDS and PDSE), and IBM z/OS UNIX System Services hierarchical file system (HFS) files.

The File Manager base component helps speed the application-development process by identifying the structure of your records and displaying each field in a human readable format, according to its data type.

This component supports QSAM, VSAM (including IAM - Innovation Access Method files), PDS, and HFS data sets, including support for double byte character set (DBCS) data in these data sets.

You can edit entire files (regardless of size) and use a template or copybook for formatting and record selection during an edit or browse session.

DB2 component

Whether you are a DB2 DBA, application programmer, or just wan to retrieve information stored in a DB2 database, the File Manager DB2 component provides something for you.

Database management abilities (such as create and drop objects, copying data within DB2, handling privileges, and import/export) and an editor for DB2 tables and views (in either read only or update mode), encompasses all the usual Insert, Delete, and Update functionality one typically requires in a database application.

Full tables, views, or the results of a limited SQL query that you have customized yourself are provided, with the ability to save the query you made, in a File Manager template.

If you are writing SQL to be used in your applications, another handy File Manager utility is the ability to be able to refine and test an SQL query by using the prototyping and analysis feature (which includes an explanations feature).

Getting data out of, or back into DB2 to or from QSAM or VSAM files, is also provided by a powerful utility that allows the export of DB2 data in a number of formats, including the external format used by the DB2 UNLOAD utility, the format used by the DB2 sample unload program DSNTIAUL, a tailorable File Manager export format with multiple options to control handling of NULL values and options for various column data types, and a delimited text format, such as comma-separated value output.

You can also generate batch JCL for the most commonly used DB2 utilities to save time.

Archived

IMS component

With File Manager's IMS component, accessing data stored in an IMS database also becomes very easy. Although the database storage concepts are different than DB2, File Manager still delivers a similar look when viewing or editing your data, and creating and saving customized queries is just as easy as using the other components.

Using record structures defined in COBOL or PL/I copybooks, just like the BASE component, the File Manager IMS component enables you to view or edit IMS segments displayed with formatting according to their individual field data types. You can find and change data across an entire database hierarchy or in specified segments and fields.

Data displays for IMS always include the segment name to help you identify where you are in the database (handy in cases where you are viewing a large result set, or an entire hierarchy).

Database navigation commands are available with flexible parameters and scoping, to allow very selective navigation or applicability, or a more generic approach, depending on your requirements.

You can use flexible criteria to select IMS database segments containing specific field values, and extract the segments into a sequential data set for later use, or you can use the IMS component to load them into a different database.

CICS component

Those users wishing to manage or query data in CICS VSAM file resources, transient data queues, and temporary data queues, have been provided with the File Manager for CICS feature, which allows access to CICS resources under a CICS transaction. The CICS resources supported for view, edit, and certain File Manager utilities (such as data create, copy, and compare) are VSAM files, temporary storage queues, and transient data queues.

You can authorize users to view and change the status of supported local and remote CICS resources and perform File Manager base and File Manager IMS tasks from the File Manager CICS transaction.

The same display and record selection capabilities in the CICS environment as in the BASE product, providing for quick easy access to CICS data. Also very useful is the ability to create full copies of open CICS files (or even TS or TD queues) to another QSAM, VSAM, HFS, or even PDS file.

Another nice benefit of File Manager for CICS is that the majority of File Manager related processing happens offline from the CICS task, so it runs little risk of adversely affecting other CICS users, even when (for example) copying an entire file somewhere else.

File Manager for CICS even maintains the same look and set of editor commands that you can possibly use in an ISPF environment. Logging off of your CICS environment to log on to your ISPF is not even necessary if all you want to do is use the File Manager BASE or IMS components.

2.5.3 SOA support

File Manager enables you to generate XML data from files. A File Manager template that describes the data-record layouts is required. The XML tags are generated based on the field names from the template, and the XML content comes from the data. A number of options are provided for the handling of invalid and unprintable data.

File Manager provides a plug-in for integration with Rational Developer for System z, allowing all aspects of Web service (and traditional application) development to be undertaken from the same developer tool.

File Manager is enhanced to provide a scrambling algorithm that modifies data in a field while maintaining its system data type. Scrambling is intended to de-identify (disguise) personal information in different ways for different data types. The goal of repeatable scrambling is that application relationships based on equality tests can be maintained, if desired, even after the data is scrambled. A number of standard algorithms are provided to give the user complete control over the type of scrambling performed.

2.6 Optim MOVE

IBM Optim MOVE for DB2 and IBM Optim for Legacy for z/OS are components of the IBM Optim solution for the z/OS environment. Optim enables you to assess, classify, subset, archive, store, and access enterprise application data. Optim uses the relationships defined in the DB2 Catalog, where available, and supplements these relationships with those defined in the Optim (OPT) Directory. Optim runs as a TSO/ISPF application and incorporates familiar ISPF commands. Optim handles any number of tables and relationships, regardless of their complexity.

Using the features in Optim, you can:

- Isolate historical data from current activity and safely remove it to a secure archive.
- Access archived historical data easily, using familiar tools and interfaces.
- Restore archived data to its original business context when it requires additional processing.
- Create realistic test data.

Optim helps you achieve these benefits with the following components: ACCESS, ARCHIVE, MOVE, and COMPARE. This section introduces MOVE and Move for Legacy. A reference to Move includes Move for Legacy.

2.6.1 Test data management

IBM Optim MOVE offers comprehensive test data management capabilities for creating realistic, right-sized test databases, protecting data privacy in vulnerable development and testing environments and validating test results. You eliminate costly cloning processes and correct defects early in the development process, where they are cheapest and easiest to fix. Most importantly, Optim scales to meet your development and testing requirements, across applications, databases, operating systems, and hardware platforms.

The Optim test data management capabilities provide an efficient alternative to database cloning, allowing you to create development and testing environments that are sized appropriately.

MOVE is a relational facility used to copy or move sets of relationally intact data from multiple source tables to corresponding destination tables. Move for Legacy allows you to perform the same processes, using data in IMS, Virtual Storage Access Method (VSAM), or sequential files.

Archived

MOVE is indispensable when:

- Creating test data that is a relationally intact subset of existing production data.
- Copying related sets of data to an “area” where any problems can be resolved. Then, the corrected data can be re-introduced to the production data.
- Migrating subsets of data that require data transformations as part of the migration.

Programmers and database administrators (DBAs) are no longer required to spend hours writing programs to accomplish these tasks. Instead, MOVE provides an interactive utility to prompt for the necessary information and perform the task.

Because MOVE runs as a TSO/ISPF application, the Help and Tutorial facility in ISPF is also supported. Menu-driven prompt windows are used to specify which data to obtain and which process to perform. Intelligent screen handling technology provides simultaneous display of multiple tables, pop-up windows, cursor-sensitive online help, and tutorials.

Tables

Throughout the Optim documentation, the term “tables” refers to not just to DB2 tables, but also to views, aliases, and synonyms, as well as to Legacy Tables that describe data in IMS, VSAM, or sequential files.

2.6.2 Processing flow

To copy a set of related data, you define the source of the data and the destination. The source is defined in the Extract Process and the destination in the Insert Process.

Extract process

An Extract is the process of copying data from one table or a set of related tables to an “Extract File.” An Extract File is a sequential file that contains the extracted data and information about that data. The supporting information defines the characteristics of the data, and optionally includes the object definitions for that data. (Object definitions are the definitions for tables, keys, indexes, and so on.) An Extract File can be used repeatedly to create new databases or to refresh existing databases.

To perform an Extract Process, you must specify:

- The names of the tables from which data is extracted and the name of the Start Table or the table used as the starting point for the process.
- The relationships between the tables to be used for the extract. By default, all relationships are used. Note that if an unload program is available, you also can indicate whether it is to be used to extract the data.

To identify or limit the rows to be extracted, you can use:

- Selection criteria for any table
- A list of rows in the Start Table
- A selection factor and maximum row limit

Table definitions (DB2 and Legacy) and column definitions are always extracted. You can also extract subordinate definitions as follows:

- Primary Keys • Column Field Procedure Names
- Relationships • Triggers
- Indexes • User Defined Types
- Views • User Defined Functions
- Aliases • Stored Procedures

- Synonyms

Access Definitions

The information that identifies the data for the Extract Process (table names, relationships, and selection criteria) is stored in an Access Definition, which is copied to the Extract File.

A named Access Definition is available to all components of Optim. You can also define an unnamed, temporary Access Definition to be used for a single Extract Process.

Next, we describe the various Access Definitions.

Insert Process

After an Extract File is created, the data in the file can be inserted into the destination tables. These tables might or might not reside in the DB2 subsystem from which they were extracted.

Table Maps

A Table Map matches source tables to destination tables. Using a Table Map, you can exclude individual tables from processing and map tables with dissimilar names. You can use one or more existing Table Maps (APPLY command) as the basis for a new Table Map, or define the Table Map at the same time you define other specifications for the process.

When the names and attributes of columns in the destination table match those of columns in the source table, MOVE automatically inserts the data. If the columns do not match, you can use one or more Column Maps to tell MOVE how to insert the data.

Column Maps

A Column Map matches source and destination columns that have different names or attributes. You can also use a Column Map to provide default values, or specify literals, constants, expressions, and exit routines used to populate destination columns. Although MOVE automatically handles many transformations, you can use custom exit routines for more complex processing.

Create Object Definitions

MOVE generates the appropriate SQL DDL statements to create destination tables that do not exist. These tables can be created as part of the Insert or in a separate step, and the created tables can be identical to the source tables and columns, or you can modify the table definitions.

Other object definitions, such as primary keys, relationships, indexes, and son on, that have been extracted with the data can be created by request also.

Insert processing

You can choose the type of processing for all tables as a group or by table:

- If you select Insert processing: A row in the Extract File is inserted when it does not already exist at the destination. However, if the row exists, the row to be inserted is discarded.
- If you select Update processing: A row in the Extract File replaces or updates a matching row in the destination. However, a source row that does not match a destination row is not inserted.
- If you select both Insert and Update processing: New rows are inserted and existing rows are updated.

Other processes

In addition to Extract and Insert, MOVE includes the following other processes or facilities:

- Load Process: To insert large volumes of data, the Extract File can be transformed into a DB2 Load or other load format to be used with a Load Utility.
- Create Process: The object definitions in the Extract File can be used to create any and all objects (tables, primary keys and relationships, indexes, and so on) without inserting source data.
- Convert Process: The data in the Extract File can be transformed according to parameters in the Table Map and Column Maps and saved as modified. Conversion is useful for masking source data or creating consistent, refreshable test data.
- Retry: Retry can be executed for one or more rows in the Extract File that cannot be inserted. A row that cannot be processed is marked as discarded. The Retry Process attempts the previous operation for the discarded rows only.
- Restart: Restart can be executed when a process does not complete successfully. A process might terminate unexpectedly due to resource restrictions, such as space allocation or time limitations. Using Restart, the process is restarted from the last commit point. MOVE tracks the most recently committed row in the Extract File and restarts the process with the succeeding row.
- Browse: The Extract File can be browsed. Browsing is useful for reviewing the error status of individual rows or for verifying the data in an Extract File.

2.7 Workload Simulator

Workload Simulator (WSim) provides the ability to simulate terminals and the associated messages. It also lets the user alter message loads during a run. It can be used to generate a large volume of messages to evaluate the reliability and approximate performance characteristics of a network under expected operating conditions. Simply stated, anything that a real user can do at a terminal, Workload Simulator can do faster, more reliably, and typically at less cost.

IBM Workload Simulator has these benefits and features:

- Helps prepare your networks for peak transaction volumes
- Enables testers to conduct reliable tests on stress, performance, and capacity as well as regression and function tests
- Simulates different terminals, terminal features, and terminal operator actions
- Provides support for enhanced TCP/IP support, SNA, and CPI-C (LU 6.2)
- Helps manage the test process
- Offers several options for creating scripts to use in simulations
- Provides screen images, data, and reports during simulation
- Helps testers compute and analyze test results

You can run WSim on MVS in batch mode, as a procedure, using the more user friendly WSim/ISPF Interface, or under TSO.

Archived

2.8 Rational Developer for System z

Rational Developer for System z (RD/z) speeds the development of traditional mainframe, Web, and composite applications. Built on an Eclipse platform, it lends itself readily to integration with other products, both IBM and non-IBM.

2.8.1 Support for the IBM problem determination tools

In this section, we discuss the IBM problem determination (PD) tools.

Remote compile generation

With remote editing, compiling, and debugging, you can develop or enhance many types of applications, including CICS, batch, TSO, or IMS Transaction Manager applications. These applications can access many forms of data, such as DB2, VSAM, DL/I, and QSAM data. You can save time by editing, compiling, and debugging host applications remotely. When editing, compiling, and debugging host files from the workstation, you work in a cooperative Windows and TCP/IP-based development environment, avoiding lengthy downloads and uploads unless explicitly desired.

Debug Tool Utilities and Advanced Functions (DTU & AF)

COBOL and PL/I applications, EGL applications, and Java applications can be debugged through a common interface using the local debugger. The remote debugger supports debugging of code that runs in the following z/OS environments:

- Batch
- TSO
- CICS
- IMS, both IMS Database Manager and IMS Transaction Manager, with or without Batch Terminal Simulator (BTS)
- DB2 (including stored procedures)
- WebSphere Application Server

The debugging sessions are cooperative. The remote distributed debugger resides on the workstation and interfaces with the IBM Debug Tool Utilities and Advanced Functions, which runs on the host with your application. The workstation interface communicates with the host z/OS products through TCP/IP.

File Manager

File Manager (FM) integration enables access to:

- VSAM KSDS files for browsing and updating
- Template-driven display of VSAM, PDS members, and sequential file data

Fault Analyzer

Fault Analyzer (FA) integration allows users to:

- Browse FA abend reports
- View dump selections relating to abends
- Annotate reports for future reference or to share comments with other users who browse the same reports

Archived

2.9 Scenario for PD Tools with RD/z

For the purposes of this chapter, we assume that the PD Tools and RD/z have already been installed and customized.

In this scenario, we are using a subset of the PD Tools to demonstrate how the tools work together to quickly and efficiently detect an error, correct it, and test it. We do that with Fault Analyzer, Debug Tool, and File Manager, all under the umbrella of RD/z. The same scenario can be accomplished in batch (green screen) without the use of RD/z.

We are not providing source code, data, or any other resources for the reader to exactly duplicate this scenario. However, the steps taken and screen captures provided should be sufficient when used as a guide for performing similar tasks in your environment. We are assume that you have completed the setup procedures to establish connection to the mainframe, and gained access to the data sets required.

Archived

Submitting the job

The first step in this scenario, as well as most processes, is to submit your JCL. We access the JCL through the Remote Systems Connection, in our case DEMOMVS, and locate the JCL in the MVS files (Figure 2-2).

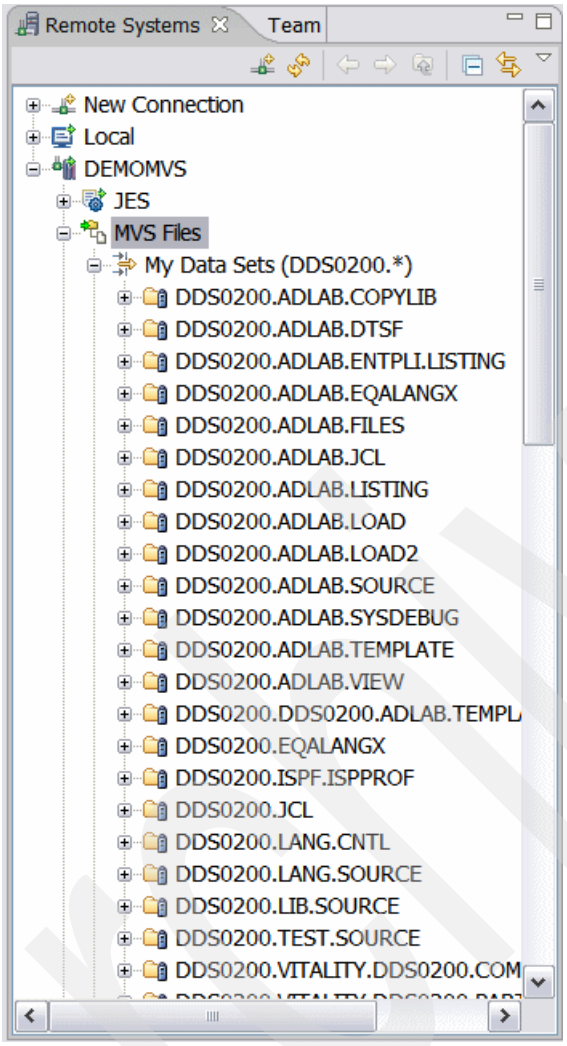


Figure 2-2 Remote Systems view

Archived

After the JCL is located, we can submit the job. Right-clicking the JCL brings up the menu for actions that can be taken on the JCL. Select **Submit** (Figure 2-3).

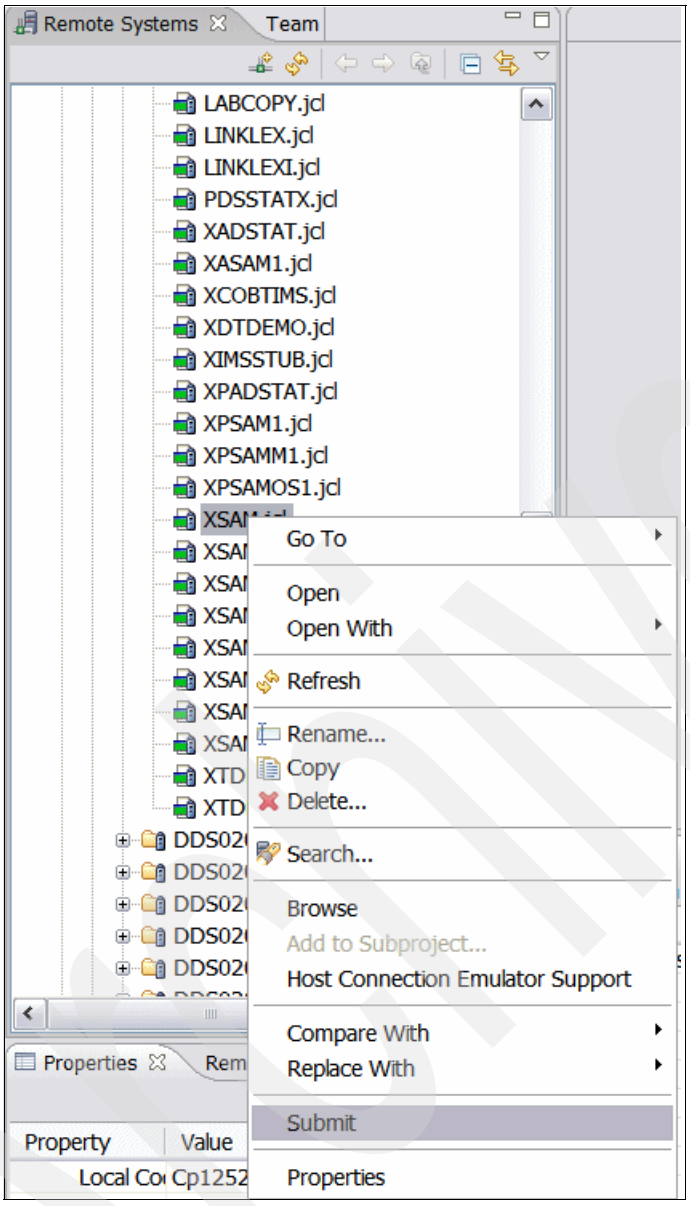


Figure 2-3 Submit JCL from Remote Systems view

Validating your job submission return code

By scrolling to the top of the box under the Remote System tab, locate the + next to the item labeled JES. Click the **+** next to JES, then click the **+** next to the filter that contains your submitted jobs, in our case, the **My Jobs** filter. There you should be able to find the job that was submitted; if not, you might have to right-click the JES and select **Refresh**.

Properties window

As you use RD/z, notice that as you highlight an item, the box with the Properties tab in the lower left hand corner will automatically populate with information for whatever item is highlighted. If we highlight our batch job output, we can see the properties for this item. You might have to scroll down the Properties box to find the Return Code for this job. You should expect to see a return code of 0C7 (Figure 2-4 on page 51).

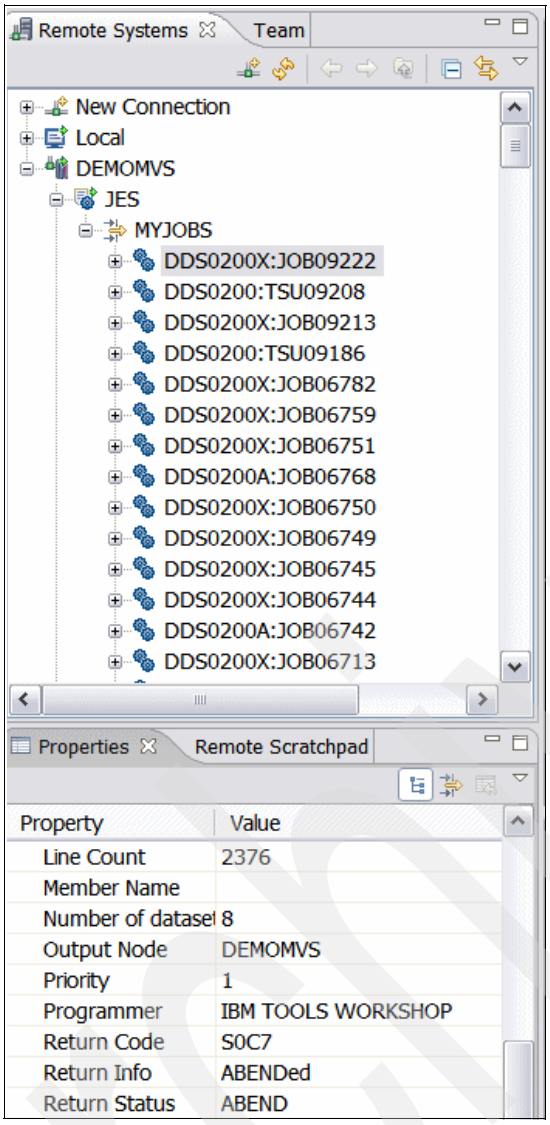


Figure 2-4 Properties window

Now that you know you have an abend, you are ready to begin analyzing the Fault. Double-click the JES output, and the results will appear in the box next to the Remote System Explorer, under the tab labeled jobname.out, where jobname is the label name of the JES output. Look for the line that looks similar to `+IDI0003I Fault ID F01385 assigned in history file FAULTANL.V8R1.HIST` (Figure 2-5).

```
JOB09222  +IDI0002I Module SAM2, program SAM2, source line # 89: Abend S0C7 (Dat
JOB09222  +IDI0003I Fault ID F01405 assigned in history file FAULTANL.V8R1.HIST
JOB09222  IEF450I DDS0200X RUNSAM1 - ABEND=S0C7 U0000 REASON=00000007  805
                TIME=14.47.13
JOB09222  -                                        --TIMINGS (MINS.)--
JOB09222  -JOBNAME  STEPNAME PROCSTEP    RC    EXCP     CPU     SRB  CLOCK    SERV
JOB09222  -DDS0200X          RUNSAM1  *S0C7   1130     .00     .00   3.13    9325
JOB09222  IEF404I DDS0200X - ENDED - TIME=14.47.13
JOB09222  -DDS0200X ENDED.  NAME-IBM TOOLS WORKSHOP   TOTAL CPU TIME=    .00  TOT
JOB09222  $HASP395 DDS0200X ENDED
```

Figure 2-5 JES output view

Archived

Using the Ctrl+F key combination brings up a find menu that can be used to find this line by finding the string “hist”. If you highlight the history file name (ADPOT00.HIST) in this example, we can use this in specifying the history file we will be using in the next section.

Analyzing 0C7 with Fault Analyzer

Now we start using the Fault Analyzer perspective. Switch perspectives to Fault Analyzer, right-click **FA Artefacts** in the top left box, and select the **Add New History File** option from the context menu. A pop-up box will appear; in the input line labeled History File Name, enter the name of the history file found in the job output. When the history file is specified, click the **OK** button (Figure 2-6).

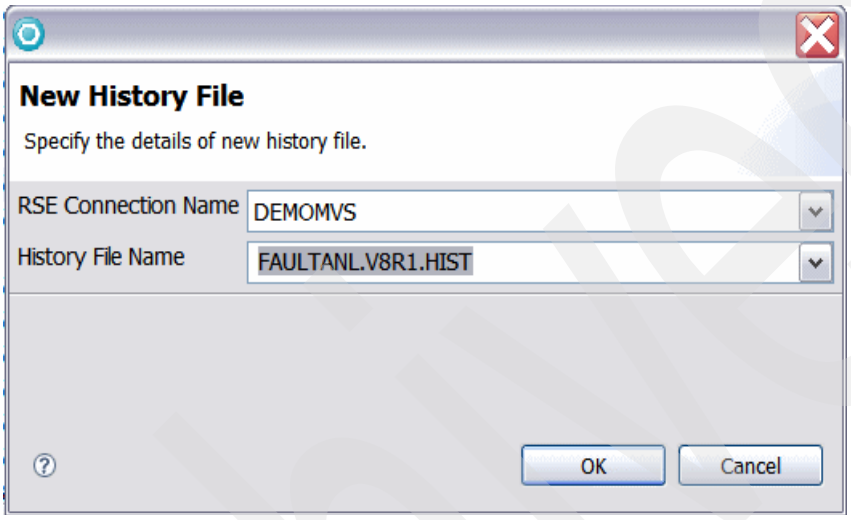


Figure 2-6 New History File window

This adds the history file name under the FA Artefacts label. Right-click the history file and select **Populate/Refresh history file** from the context menu. In the bottom right of the window, you see a matrix of the most recent abends. One of them should be the 0C7 just produced from our batch job. To start analyzing the problem, right-click our abend and select **Browse Report** from the context menu (Figure 2-7).

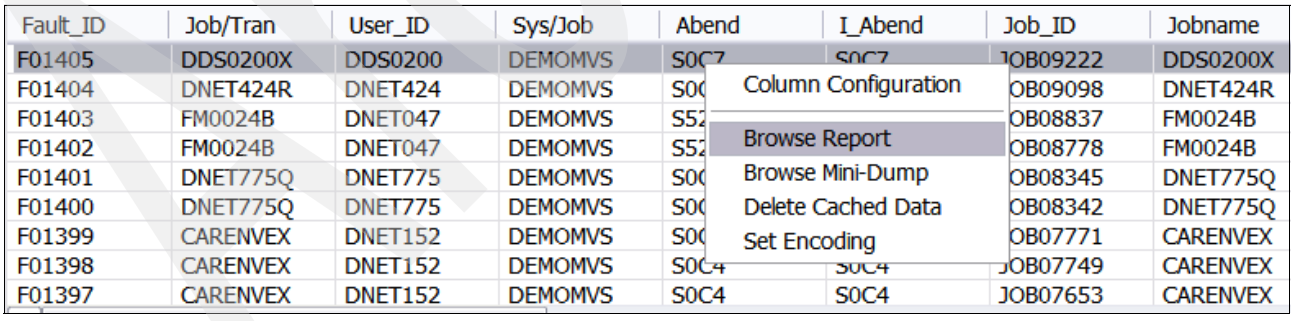


Figure 2-7 Contents of Fault History File

When the report comes up, you can see all the tools that are available in the Fault Analyzer perspective. The FA Artefact in the top left allows us access one to many history files from one or many mainframes. Below this box is the outline that can make navigating the history file quick and easy. The bottom section shows the list of faults and details about the faults in the columns. These columns can be customized and sorted to make finding a specific abend quick and easy. Finally, the FA report is in the main part of the window that contains the actual Fault History details.

The Fault History details are presented in five tabs:

- The Initial tab gives a summary of the problem (Figure 2-8).
- The Event Summary tab shows all the events (the call path) leading up to the problem and indicates in which event the original problem occurs.
- The Abend Job Information tab contains all the information about this specific abend (similar to the information shown in the abend matrix in the bottom section). In addition, it has the runtime environment settings and link edit information for the problem program.
- The System-Wide Information tab has control blocks that pertain to the problem. This is specifically important when the abends occur in a subsystem such as CICS, IMS or DB2, because this section will display values for the control blocks for the specific subsystems.
- Finally, the Misc. Information tab contains the specifics about how Fault Analyzer is invoked and some of the customizations used.

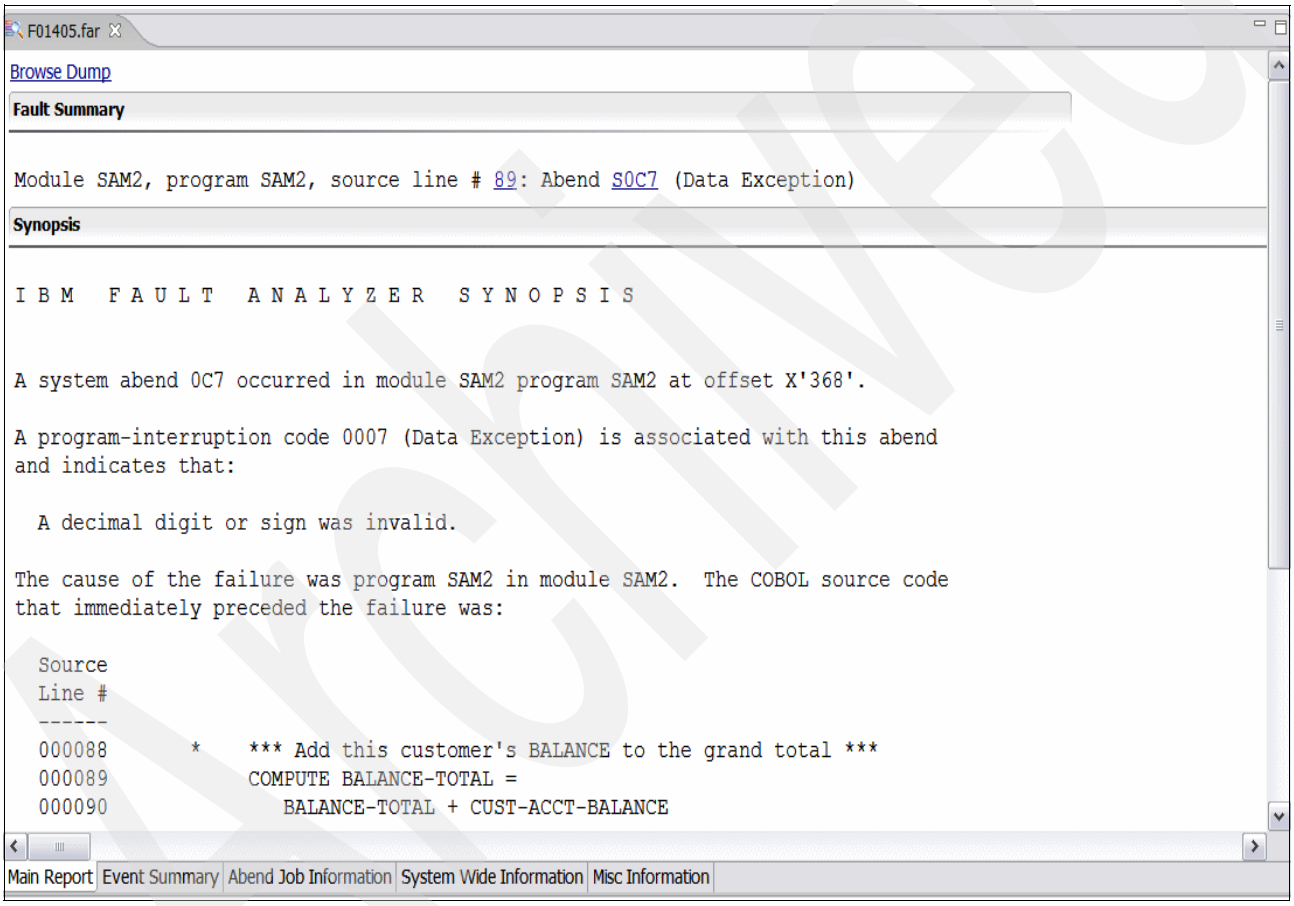


Figure 2-8 Fault Analyzer Fault Summary

To start analyzing the abend, we use the Main Report tab. All items underlined and in blue are links to other areas of the report. Clicking the **S0C7** abend invokes the Lookup part of the Fault Analyzer tool. While the summary page gives a brief explanation of the abend code, Lookup gives a full explanation. Lookup can be used independently of the rest of the fault analyzer tool.

In Figure 2-8 on page 53, the Fault Summary page shows the COMPUTE statement on line 89. Further down, the variables used on line 89 and the definitions within the program are shown, and at the bottom, the values of those variables are shown. Notice that the value of CUST-ACCTBALANCE is shown in hex because the data is not a valid packed number. This is also flagged with `*** Invalid Numeric Data ***` text denoting that this value caused the problem (Figure 2-9).

```
The COBOL source code for data fields involved in the failure:

  Source
  Line #
  ------
  000059            05  CUST-ACCT-BALANCE      PIC S9(7)V99  COMP-3.
  000066            05  BALANCE-TOTAL        PIC S9(7)V99  COMP-3.

Data field values at time of abend:

  BALANCE-TOTAL     = 10948.44
  CUST-ACCT-BALANCE = X'7C7B5B6C50' *** Invalid numeric data ***
```

Main Report | Event Summary | Abend Job Information | System Wide Information | Misc Information

Figure 2-9 Fault Analyzer summary data fields involved in abend

Clicking the **Event Summary** tab, we can see the call path. This tells us that the problem occurred in a program named SAM2, and the program SAM2 is called by SAM1. Scrolling down to this part of the history report, we see all the events within the report as well as that the Event containing the problem is already expanded and highlighted. Scrolling down to view event 3, you can see the same type of information as in the Summary tab, but in a lot more detail.

At this point, we know enough to start a debug session to continue our discovery. We know that the problem occurs in program SAM2 with a variable named CUST-ACCT-BALANCE, and the value in this variable is X'7C7B5B6C50' (Figure 2-10 on page 55).

Archive

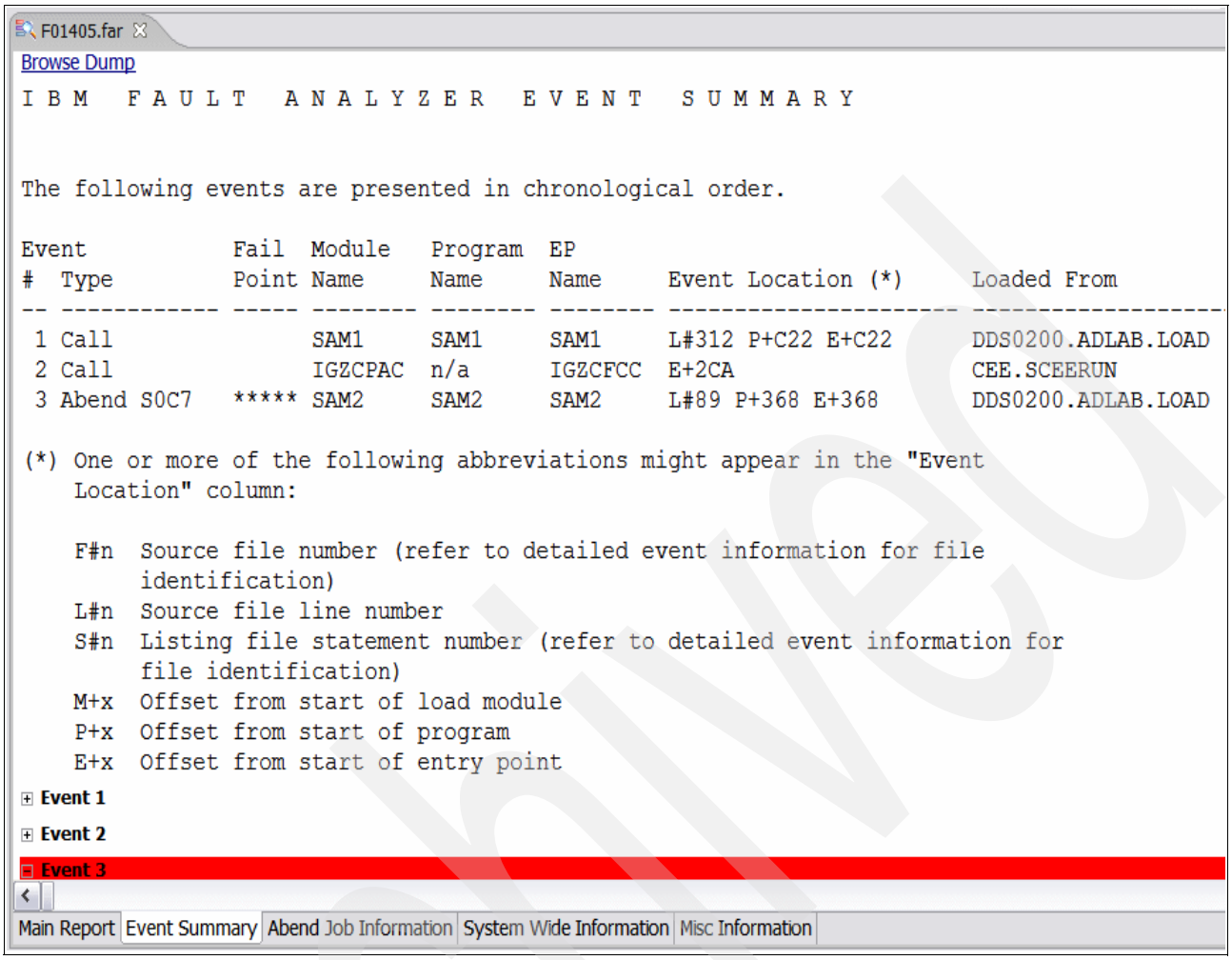


Figure 2-10 Fault Analyzer Event Summary

Debugging the program with OC7 With Debug Tool

In order to use an interactive debugger, we have to tell the debugger where to interact with us (the user). We could use a VTAM® screen (also known as a green screen) but this is not necessary because we can route the interactive debug session to RD/z. To do this, you would go to the debug perspective and ensure that RD/z is listening for an interactive debug session and determine the IP address of our client to direct the debug session to our RD/z client. To switch to the debug perspective, select **Window** → **Open Perspective** → **Other** → **Debug**.

In the Debug Perspective, the top left of the window will have an icon that looks like an inverted tee. If the RD/z is listening, the ICON will be green; if not, it will be red, as shown in Figure 2-11.

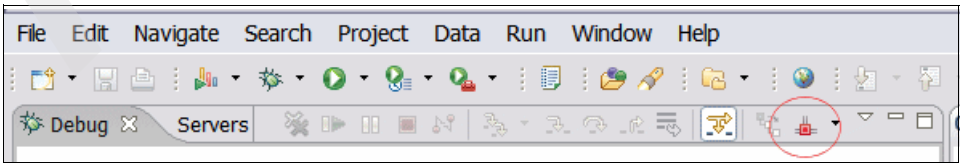


Figure 2-11 Debug Listening Icon

If RD/z is not listening, use the drop-down menu and select **Start listening on port: ####**, where #### is the port number, usually 8001.

To have the debug session come to RD/z, we have to tell the running job to have the interactive debug session come to our client IP address. To determine the client's IP address, use the same drop-down menu and select **Get Workstation IP…**. You can highlight the IP address and copy it for later use (Figure 2-12).

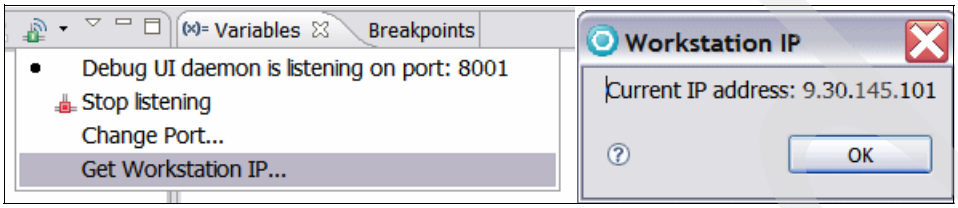


Figure 2-12 Get Workstation IP

To start the debug session, we want to modify the JCL to tell the run time to use the debugger. We go back to the Remote System Explorer, and navigate back the XSAM.jcl member of the PDS "userid. ADLAB.JCL". We double-click the XSAM.jcl member so we can edit this member. You must prepare your JCL with debug test parameters. Add the TCPIP parameter and IP address to reflect the IP number discovered by acquiring the Workstation IP address above (Figure 2-13).

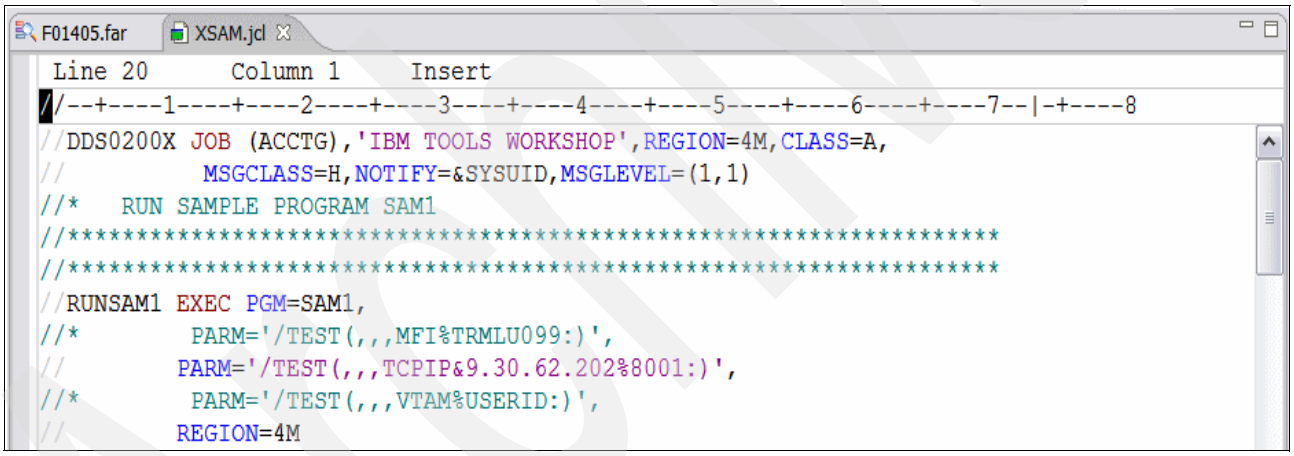


Figure 2-13 JCL modification

After the JCL change, save the source by right-clicking within the JCL source and selecting **Save** from the context menu. After saving, right-click within the source once more and this time select **Submit** within the context menu (Figure 2-14 on page 57).

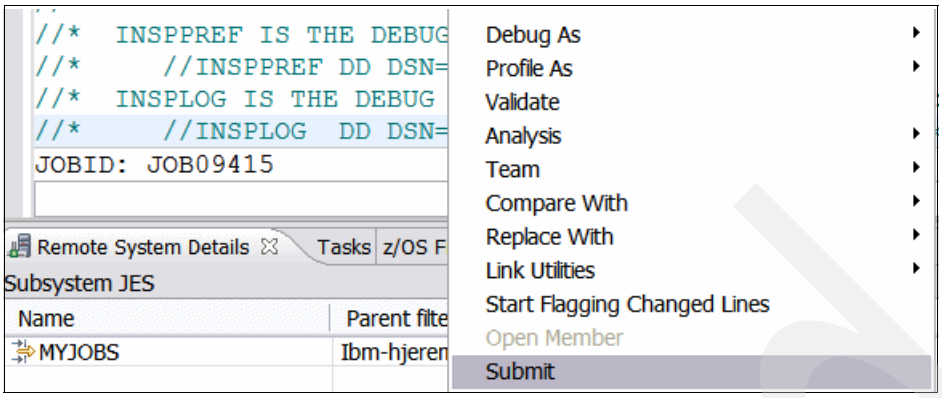


Figure 2-14 Submit JCL from editor

When the debug session starts, a pop-up menu is presented, titled Confirm Perspective Switch, to validate that you want to change to the debug perspective. Click the **Yes** button and you will enter the Debug Perspective.

From the Fault Analyzer report, we know the problem occurs in SAM2, so we should navigate to this program. In this example, we set a deferred breakpoint for the load module of SAM2 with the entry point of SAM2. In the top right of the window, select the **Breakpoints** tab. Within the Breakpoints window, right-click and select **Add Breakpoint** → **Entry** (Figure 2-15).

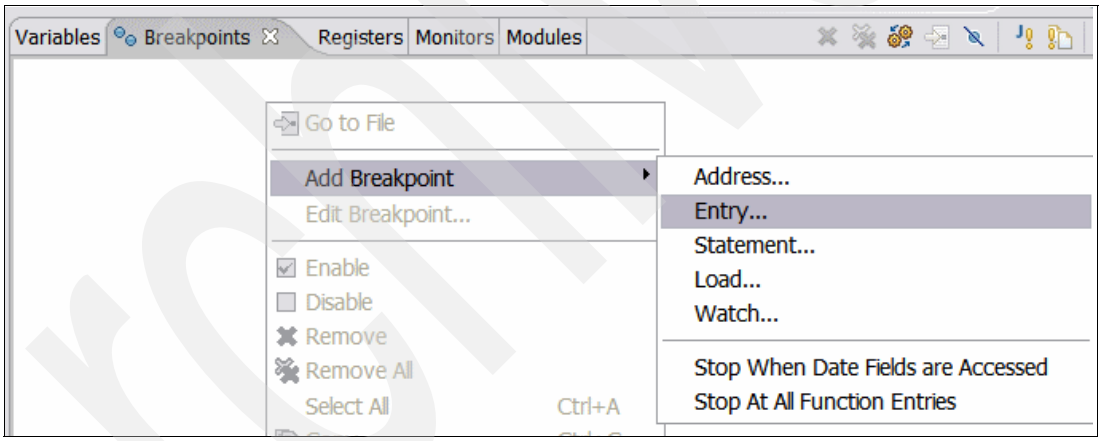


Figure 2-15 Adding a breakpoint

Archived

Specify SAM2 in all the text boxes and make sure that the box for **Defer breakpoint until executable is loaded** is checked. Then click the **Finish** button (Figure 2-16).

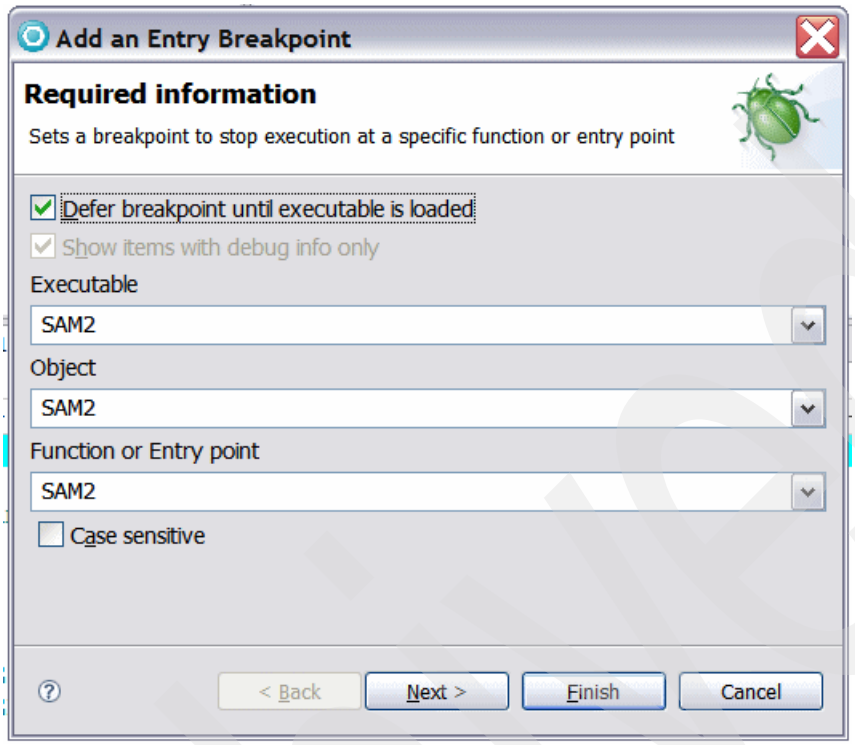


Figure 2-16 Specifying the function for a breakpoint

With the breakpoint set, use the **Resume** button to run into the SAM2 program. The resume button is a green arrow located in the step tool bar (Figure 2-17).

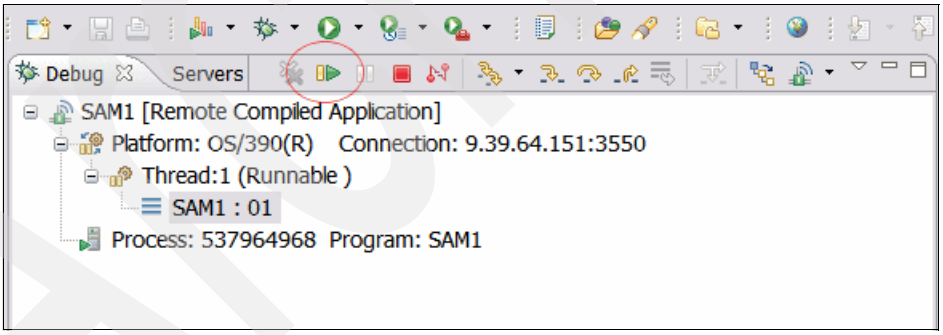


Figure 2-17 Debug Toolbar - Resume button

Referencing the Fault analyzer report, we see that we want to watch the problem variable CUST-ACCT-BALANCE. To work with variables, we have to navigate in SAM2. To do this, we can use the Step icon to step over the working storage initialization and into the procedure division (Figure 2-18 on page 59).

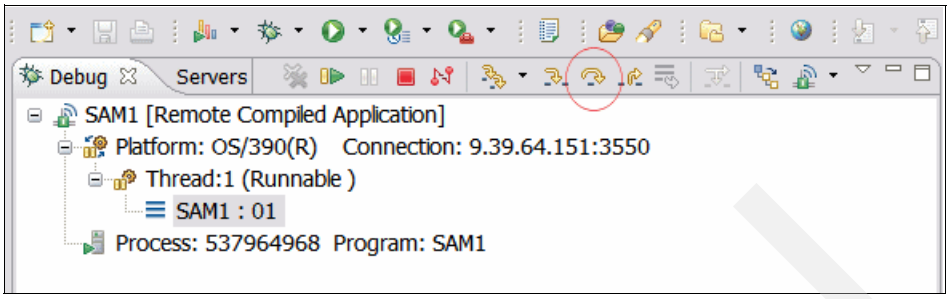


Figure 2-18 Debug Toolbar - Step function button

Now we can set a watch breakpoint and a monitor on CUST-ACCT-BALANCE. Once again, right-click in the breakpoint area, but this time select **Add Breakpoint** → **Watch**. In the field labeled Address or expression, type in CUST-ACCT-BALANCE, and then click the **Finish** button (Figure 2-19).

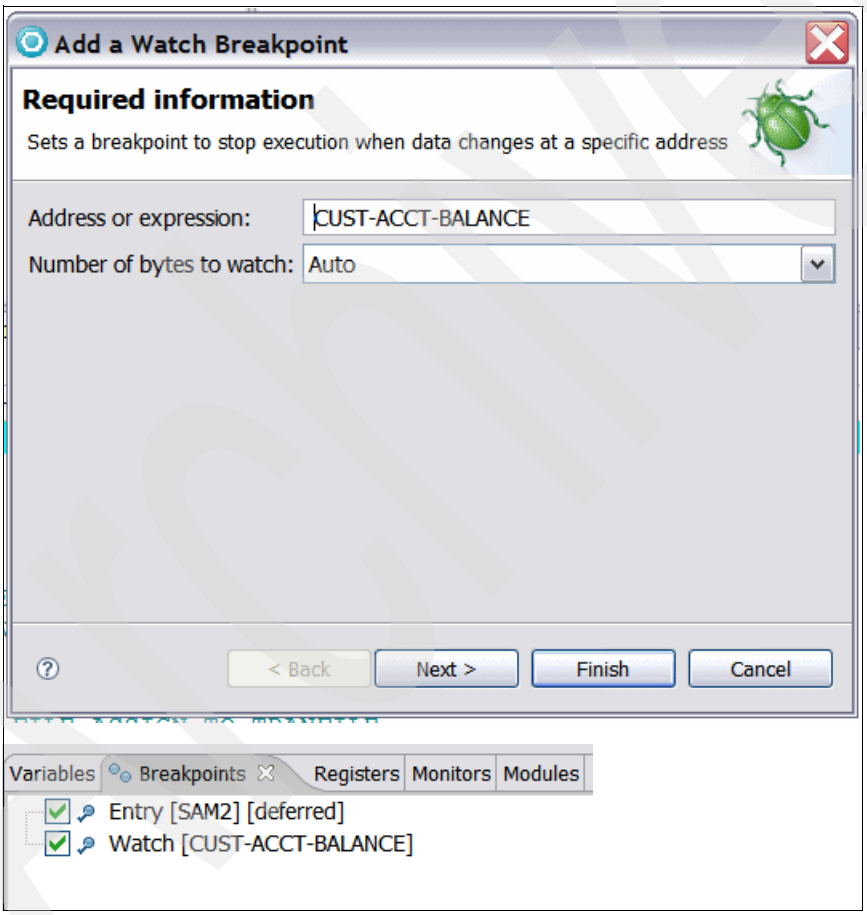


Figure 2-19 Add a Watch Breakpoint

At this time, we should have two entries in the Breakpoints window. Now select the **Monitors** tab within this same window. Right-click in the monitor box and select **Monitor Expression®** from the context menu, then specify CUST-ACCT-BALANCE in the text box. The name and value should appear within the monitor box (Figure 2-20).

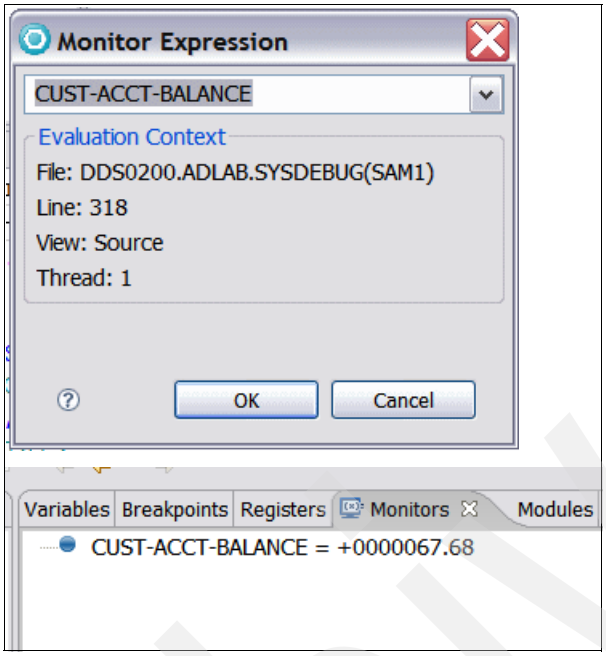


Figure 2-20 Expression monitoring

We are ready to figure out when the value of CUST-ACCT-BALANCE contains an invalid value. Click the **Resume** button until CUST-ACCT-BALANCE contains an invalid value. Optionally, we can change the representation of the value to show the value that matches the invalid value in the FA report (Figure 2-21).

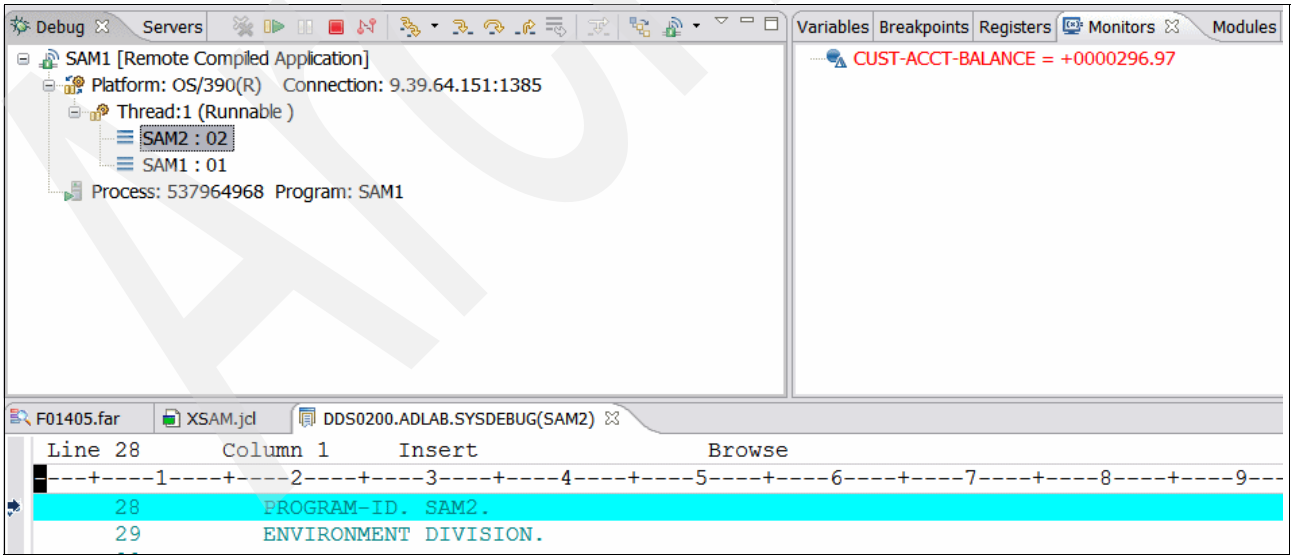


Figure 2-21 Call stack and expression monitoring

Notice that we are stopped at the entry of SAM2. Because we are at the beginning of SAM2, we can infer that the bad data came from SAM1. In the top left box, we can see the call path.

Archived

In the Fault Analyzer Event report, we saw that CUST-ACCTBALANCE is part of CUST-REC, which is the first structure passed from SAM1 to SAM2.

If we select SAM1, we can view the call to SAM2 (Figure 2-22).

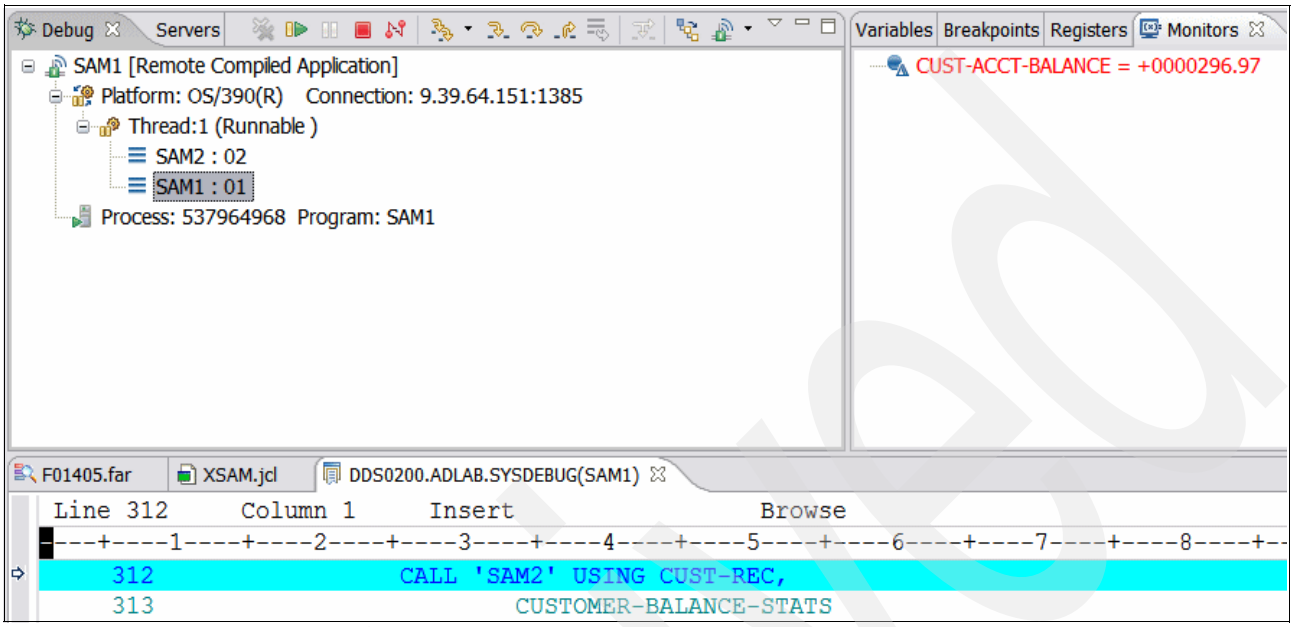


Figure 2-22 Call stack - entry point to SAM2

Scrolling to the top of SAM1 and using the Find command, we can see that CUSTREC is part of the CUSTMER-FILE FD, and scrolling up further we can see that CUSTOMER-FILE is associated with a CUSTFILE DD statement within the JCL. This confirms that the bad data comes from a data set allocated to the CUSTFILE DD statement. Selecting SAM2 in the call chain, we can temporarily fix the variable data, disable the breakpoints, and continue with our test.

To temporarily fix the variable, double-click the value of CUST-ACCT-BALANCE within the monitor window. Type in a valid number (we use 10 in this example) and then press Enter (Figure 2-23).

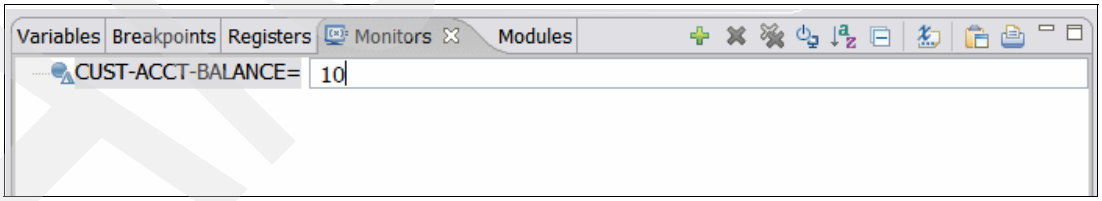


Figure 2-23 Changing a variable dynamically

Now select the **Breakpoints** tab and disable both breakpoints by clicking in the square box next to each breakpoint (Figure 2-24).

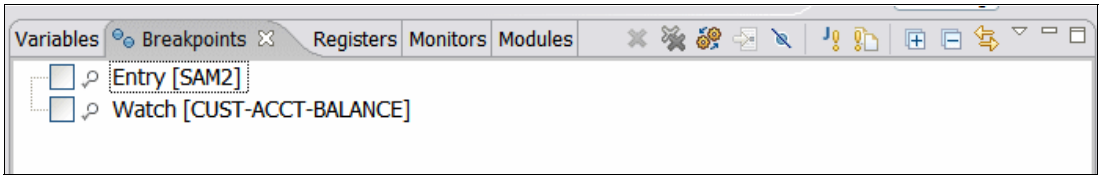
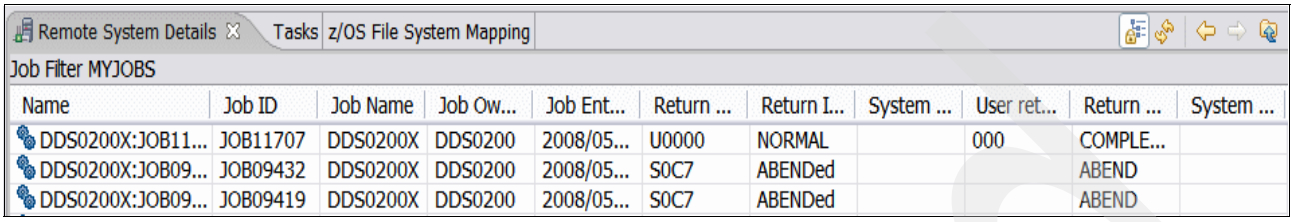


Figure 2-24 Turning off breakpoints

Using the **Resume** button, we can let the application run to termination. We can check the return code by switching back to the Remote System perspective. In the Remote Systems tab on the top left of this window, we can expand **JES**, right-click **My Jobs**, and select **Show in Table**. At the bottom of the window, we can now see all jobs run and a return code of zero of the debug session we just ran (Figure 2-25).



Name	Job ID	Job Name	Job Ow...	Job Ent...	Return ...	Return I...	System ...	User ret...	Return ...	System ...
DDS0200X:JOB11...	JOB11707	DDS0200X	DDS0200	2008/05...	U0000	NORMAL		000	COMPLE...	
DDS0200X:JOB09...	JOB09432	DDS0200X	DDS0200	2008/05...	S0C7	ABENDed			ABEND	
DDS0200X:JOB09...	JOB09419	DDS0200X	DDS0200	2008/05...	S0C7	ABENDed			ABEND	

Figure 2-25 Remote System Details

The last step is to update erroneous data using File Manager.

Fixing a data problem with File Manager

We can figure out what file we want to edit and fix our JCL at the same time. In the Remote Systems view, select the **JCL** tab in the top center window where we edit files, and comment the TEST parameter line at the same time. Note that the CUSTFILE DD is using the data set named `userid.ADLAB.SAMFILE1` (Figure 2-26).

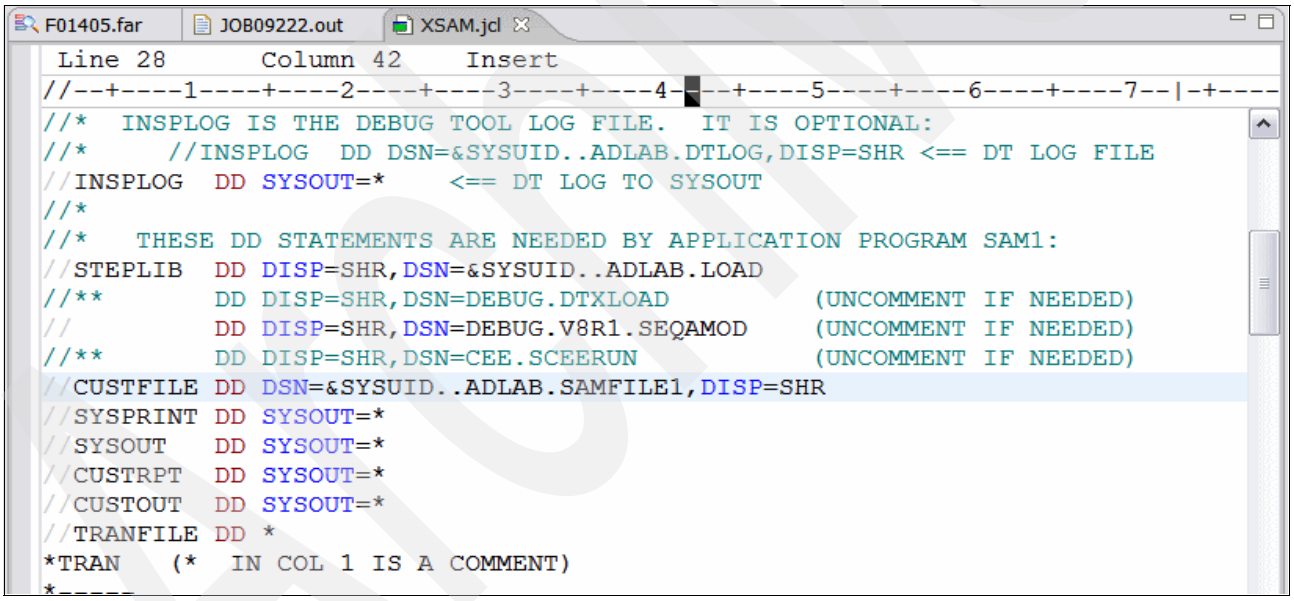


Figure 2-26 JCL File Viewer

Using Remote Systems on the left, we can scroll down and locate our file. Note that data sets are sorted in two lists in alphabetic order. The first list is all PDS data sets followed by all other data sets. RD/z uses two criteria to determine the contents of a data set in order to open an appropriate editor. The primary criteria to determine a file's content is the naming convention of the data sets.

For example, any data set ending in .JCL will be interpreted as JCL files. The other criteria to determine a file's content is the data set organization. If the data set is VSAM, we would automatically know it is a data file. Because our file is sequential and data files do not normally have a naming convention, we can identify our data set as data and at the same time identify the copybook to be used to map the data. Right-click userid.ADLAB.SAMFILE1 and select **Properties** from the context menu (Figure 2-27).

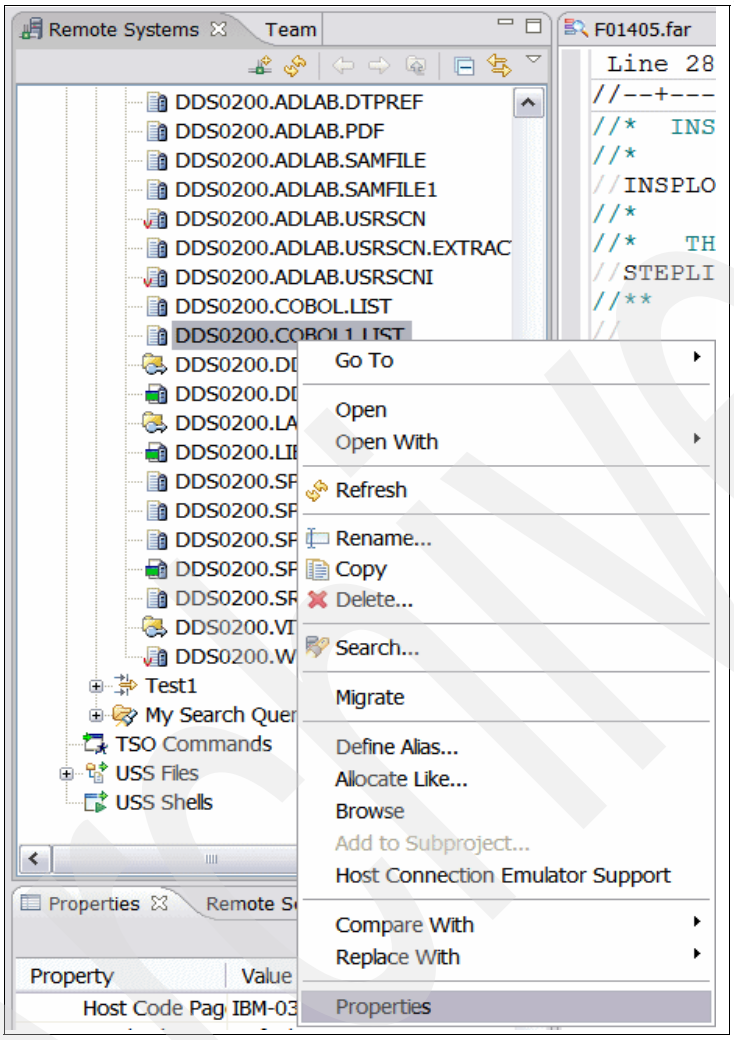


Figure 2-27 Remote Systems data set properties

In order to tell RD/z that the content of the data set is data, select **Mapping**, and in the Extension section, select the **Other** radio button and specify tdat in the text box (Figure 2-28).

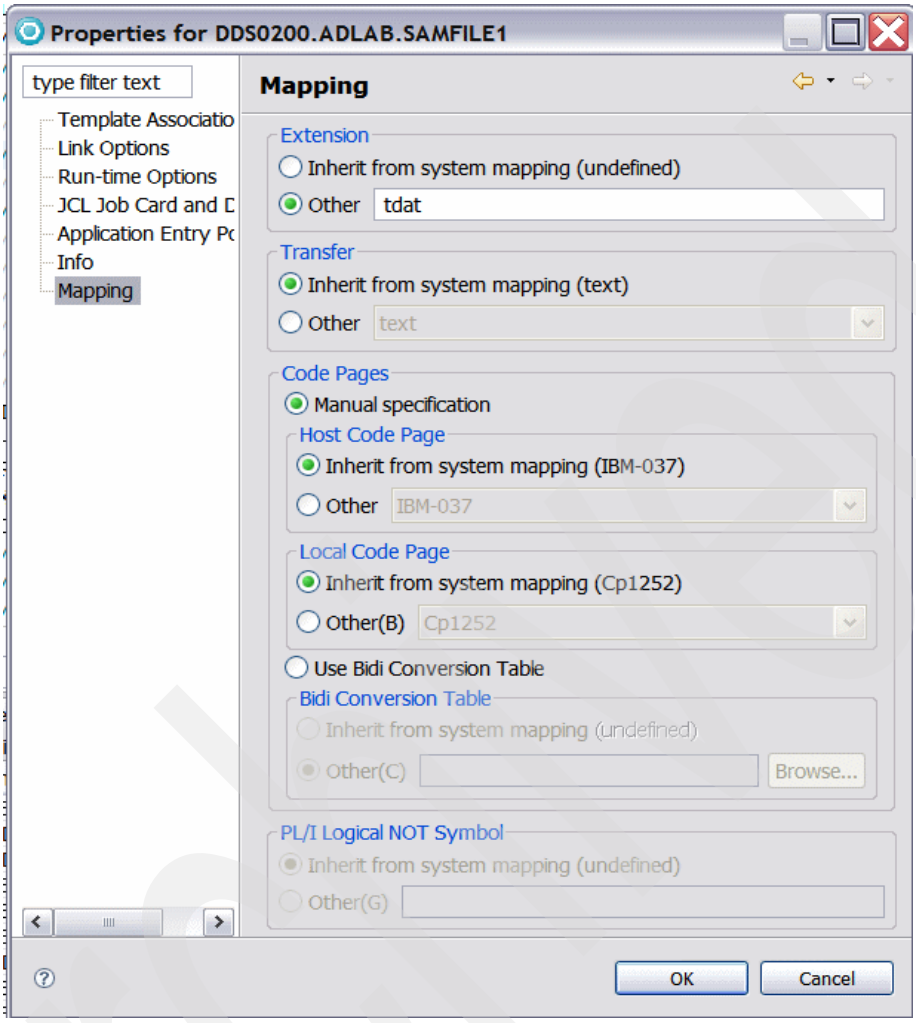


Figure 2-28 Properties - Mapping

To associate the copybook to be mapped with the data, click **Template Association** and use the drop-down menu under the title of Browse for Template/Copybook. Expand the first data set userid.ADLAB.COPYLIB and the member of CUSTMAST.cpy. Click the **Apply** button and then click the **OK** button (Figure 2-29 on page 65).

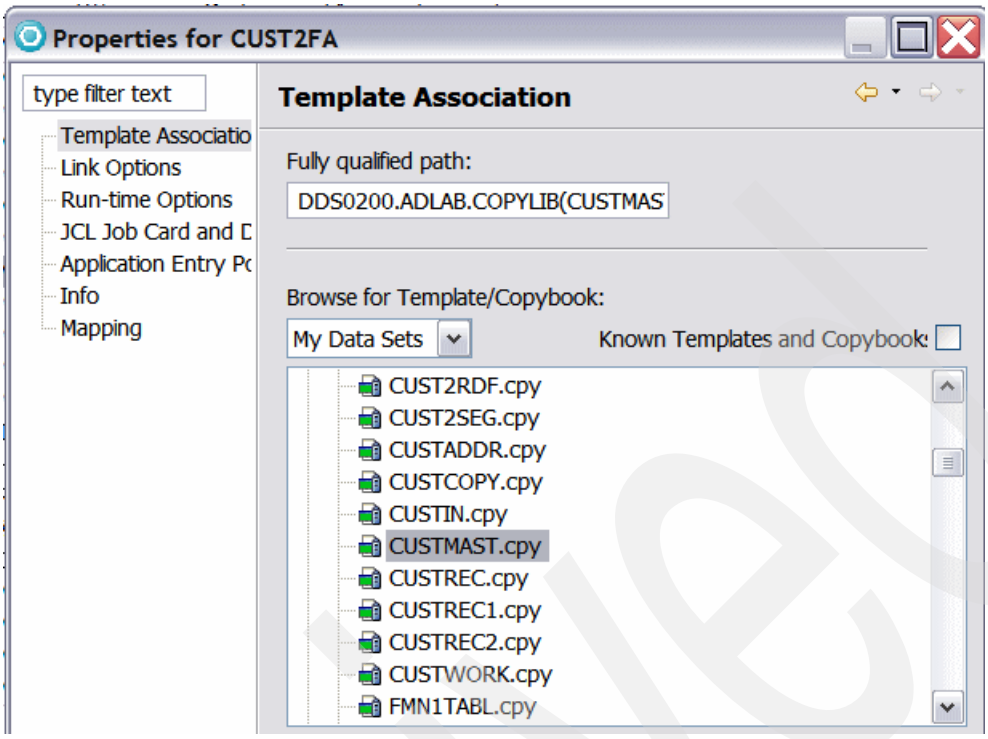


Figure 2-29 Properties - Template Association

This will cause the icon next to our file to change. We can double-click this file to edit the file with File Manager (Figure 2-30).

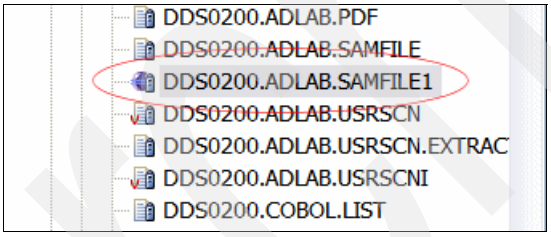
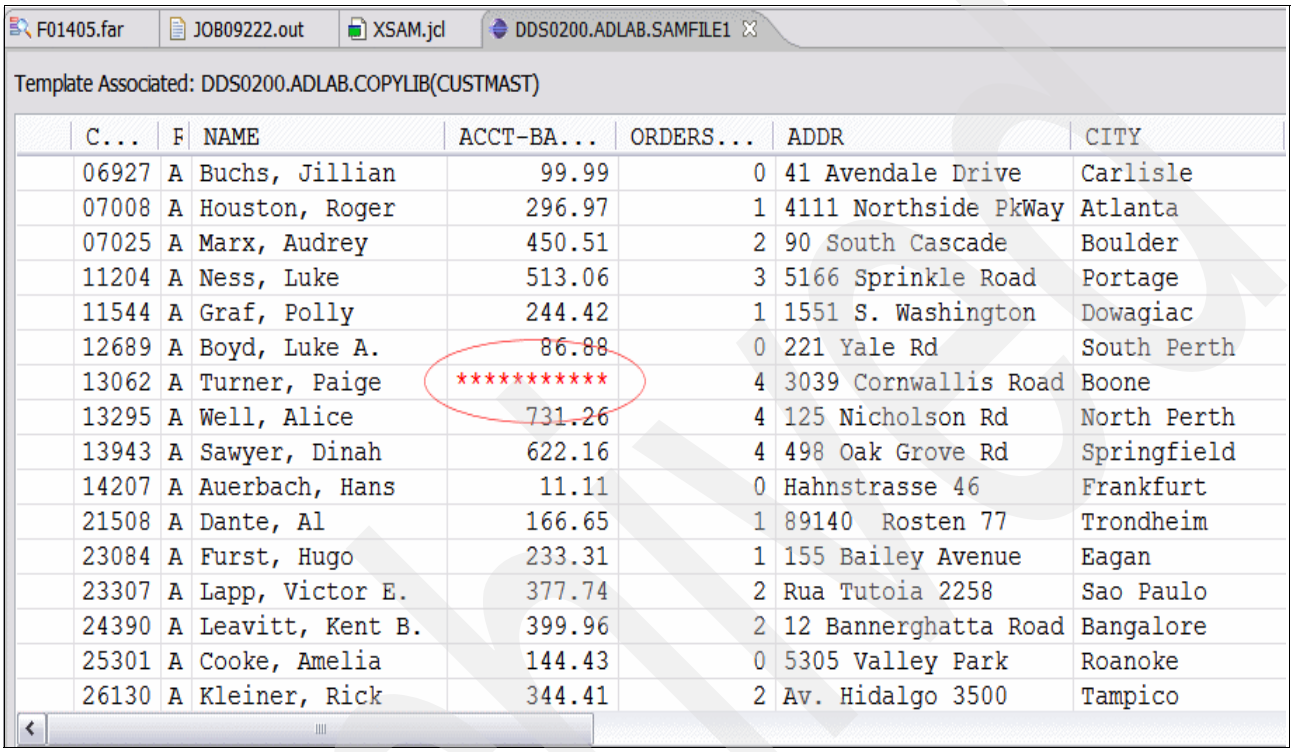


Figure 2-30 Data File with Template Icon

By double-clicking the tab for the editor of the file, we can see our file in full screen. This allows us to see a lot of data. For those familiar with the green screen version of File Manager, you will notice that we can see both the table mode and single mode in the same window. Selecting any record in the table section activates the Single record section and synchronizes the record. It is easy to see that the record containing the name of “Turner, Page” has bad data. This is denoted by the red asterisks in the account balance column (see Figure 2-31 and Figure 2-32 on page 67).



C...	F	NAME	ACCT-BA...	ORDERS...	ADDR	CITY
06927	A	Buchs, Jillian	99.99	0	41 Avendale Drive	Carlisle
07008	A	Houston, Roger	296.97	1	4111 Northside PkWay	Atlanta
07025	A	Marx, Audrey	450.51	2	90 South Cascade	Boulder
11204	A	Ness, Luke	513.06	3	5166 Sprinkle Road	Portage
11544	A	Graf, Polly	244.42	1	1551 S. Washington	Dowagiac
12689	A	Boyd, Luke A.	86.88	0	221 Yale Rd	South Perth
13062	A	Turner, Paige	**********	4	3039 Cornwallis Road	Boone
13295	A	Well, Alice	731.26	4	125 Nicholson Rd	North Perth
13943	A	Sawyer, Dinah	622.16	4	498 Oak Grove Rd	Springfield
14207	A	Auerbach, Hans	11.11	0	Hahnstrasse 46	Frankfurt
21508	A	Dante, Al	166.65	1	89140 Rosten 77	Trondheim
23084	A	Furst, Hugo	233.31	1	155 Bailey Avenue	Eagan
23307	A	Lapp, Victor E.	377.74	2	Rua Tutoia 2258	Sao Paulo
24390	A	Leavitt, Kent B.	399.96	2	12 Bannerghatta Road	Bangalore
25301	A	Cooke, Amelia	144.43	0	5305 Valley Park	Roanoke
26130	A	Kleiner, Rick	344.41	2	Av. Hidalgo 3500	Tampico

Figure 2-31 Invalid Data Notification Using File Manager

Archive

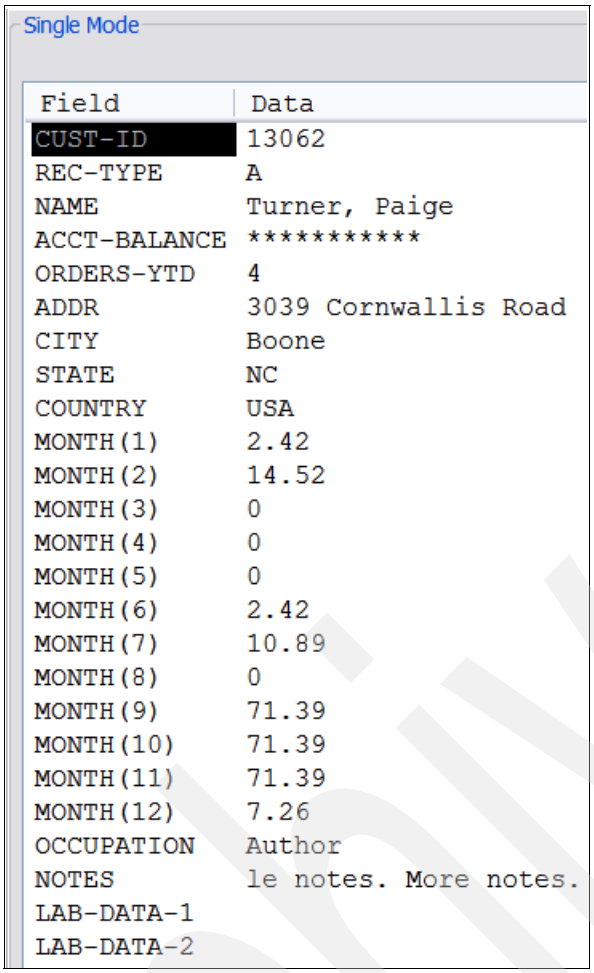
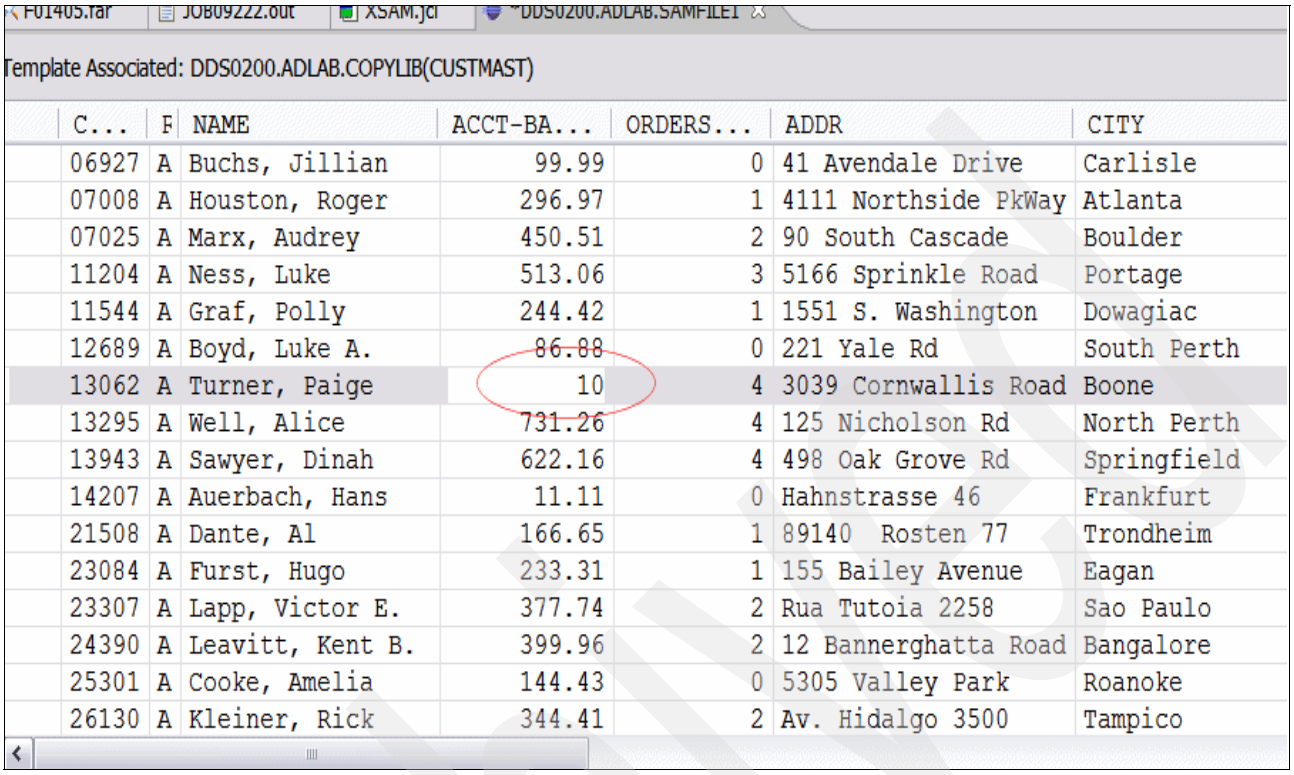


Figure 2-32 Single Data Entry using File Manager

Archived

To fix the data, simply highlight the column and overtype the asterisks with a valid number. In this case, we are using 10 (Figure 2-33).



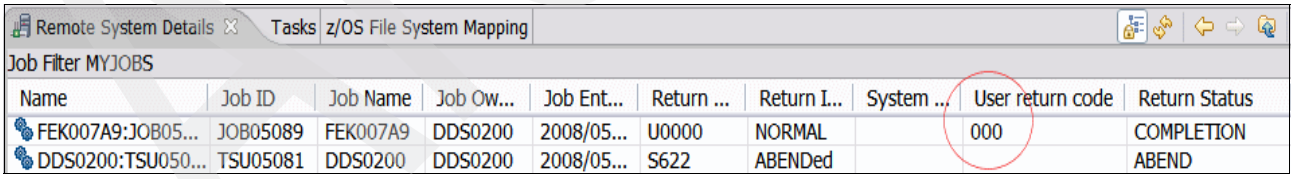
C...	F	NAME	ACCT-BA...	ORDERS...	ADDR	CITY
06927	A	Buchs, Jillian	99.99	0	41 Avendale Drive	Carlisle
07008	A	Houston, Roger	296.97	1	4111 Northside PkWay	Atlanta
07025	A	Marx, Audrey	450.51	2	90 South Cascade	Boulder
11204	A	Ness, Luke	513.06	3	5166 Sprinkle Road	Portage
11544	A	Graf, Polly	244.42	1	1551 S. Washington	Dowagiac
12689	A	Boyd, Luke A.	86.88	0	221 Yale Rd	South Perth
13062	A	Turner, Paige	10	4	3039 Cornwallis Road	Boone
13295	A	Well, Alice	731.26	4	125 Nicholson Rd	North Perth
13943	A	Sawyer, Dinah	622.16	4	498 Oak Grove Rd	Springfield
14207	A	Auerbach, Hans	11.11	0	Hahnstrasse 46	Frankfurt
21508	A	Dante, Al	166.65	1	89140 Rosten 77	Trondheim
23084	A	Furst, Hugo	233.31	1	155 Bailey Avenue	Eagan
23307	A	Lapp, Victor E.	377.74	2	Rua Tutoia 2258	Sao Paulo
24390	A	Leavitt, Kent B.	399.96	2	12 Bannerghatta Road	Bangalore
25301	A	Cooke, Amelia	144.43	0	5305 Valley Park	Roanoke
26130	A	Kleiner, Rick	344.41	2	Av. Hidalgo 3500	Tampico

Figure 2-33 Changing Data using File Manager and RD/z

When you press Enter, the number will be formatted. When you close the file, you will be prompted to save the file if you have not already done so.

Verifying results

To prove that we have fixed the problem, we can go back to our JCL and submit the job (do not forget to save the change to the PARM if you have not done so already). Check your JES output and verify that you get a return code of zero. Remember that you have to refresh the output display by clicking the **Refresh** button (Figure 2-34).



Name	Job ID	Job Name	Job Ow...	Job Ent...	Return ...	Return I...	System ...	User return code	Return Status
FEK007A9:JOB05...	JOB05089	FEK007A9	DDS0200	2008/05...	U0000	NORMAL		000	COMPLETION
DDS0200:TSU050...	TSU05081	DDS0200	DDS0200	2008/05...	S622	ABENDed			ABEND

Figure 2-34 Return Code "0"

3

The Rational Enterprise Software Configuration Management (ESCM) implementation

In this chapter, we present an approach we refer to as the Enterprise Software Configuration Model (ESCM). This chapter will present the elements of the solution and how they are combined to support an enterprise-wide Application Lifecycle Management solution. Note that this chapter assumes in-depth familiarity with the administrative features of ClearQuest, ClearCase, and Build Forge. You also need knowledge of Rational Developer for System z.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

Figure 3-1 shows an overview of the Rational solutions used in the development of this book.

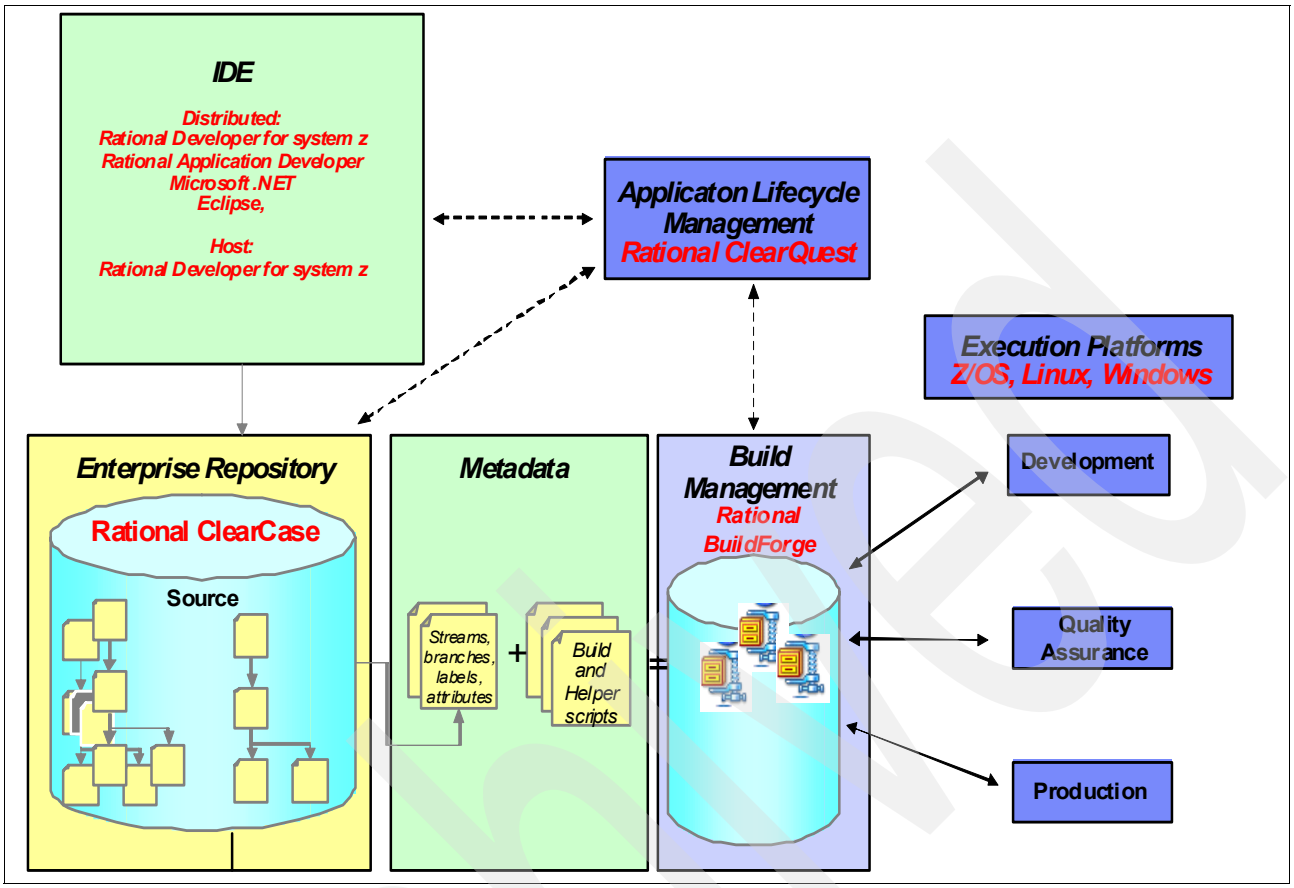


Figure 3-1 ESCM supporting architecture

Archived

3.1 Enterprise repository

If you have implemented a robust and effective SCM solution with IBM Rational ClearCase, in a distributed environment, you will find that there are more similarities between the mainframe and distributed environments then there are differences.

Most of the differences are in terminology, but once you have identified and cross-referenced those, we have found that the most successful implementations make use of some long-established, but little known features of ClearCase. The ClearCase Extensions for z/OS are a more recent addition to ClearCase (V2003.06 and later), and as their name implies, they enable you to create an Enterprise SCM that will support native z/OS builds, and provide direct access to a ClearCase repository from ISPF. In earlier chapters, we present an ESCM process as a concept. In this chapter, we describe the use of IBM Rational solutions to implement an ESCM centralized repository and life cycle. In our implementation, we combine the change and application management capabilities of IBM Rational ClearQuest, and the cross platform build management.

Regardless of the solution used, in order to support an enterprise level SCM process, the centralized repository should offer the following capabilities:

- Scalability to host thousands of files regardless of host or target platform
- Support for identifying and organizing groups of files as independently managed components, regardless of host or target platform
- Support and enforcement of a standardized application life cycle
- Support for role-based access
- Transparent access from host or distributed portals.

These characteristics enable the creation of a comprehensive application management infrastructure with the robustness to manage business critical applications, and the flexibility to be used effectively in the both the distributed and z/OS mainframe environments.

Let us now examine how we would use the capabilities of IBM Rational ClearCase to create a centralized repository that provides robustness and flexibility. This chapter assumes a working knowledge of IBM Rational ClearCase. We present a discussion of those features of ClearCase that are most relevant to deployment in an enterprise application SCM.

3.1.1 Build management

Building an enterprise application from distributed and host components has to be efficient and reliable. A critical difference between a distributed application and building a z/OS application is the build process, most particularly the methods used to configure the compile, link, and deploy steps.

Modern Java, C++, and .NET applications are built with the aid of sophisticated IDEs, such as IBM Rational Application Developer or Microsoft® Visual Studio® in the context of a project. The IDE automates and abstracts the compile, link, and packaging steps from the developer. These steps are performed either on a developer workstation or a managed build server, using information that can be stored in project files, makefiles, or environmental settings in the IDE.

Archive

z/OS applications are compiled, linked, and deployed with JCL build scripts, which are submitted to the host. All of the information about the compile and link process is contained in the build script. Modern IDEs like IBM Rational Application Developer for z have features for automating the creation and submission of these scripts.

This distinction is significant in that in the case of the distributed application, build rules and dependency relationships are maintained in the IDE and in local build scripts. In a ClearCase distributed environment, dependencies can be automatically inferred by using **`clearmake`** or invoking a build script in a clearaudit shell. In the z/OS application, the build script is the only way to make this information available to the z/OS batch environment.

In the z/OS environment, configuration of the build process and setting compiler and pre-link and link options can be performed by passing parameters to the build scripts, or in a similar fashion by invoking command-line invocation of a build script.

In order to deliver the benefit of a single repository, standard process approach, an ESCM environment must be able to manage these two different approaches with a consistent paradigm. In our project, we employed a straightforward approach.

3.1.2 Organization and identification of assets

At one level, the addition of z/OS mainframe assets to a ClearCase repository is straightforward. Practically any type of file can be stored in a ClearCase Versioned Object Base (VOB). COBOL, JCL, CICS-COBOL, and any other type of source member can all be stored in a VOB and accessed from any desktop application, as well as from ISPF.

ClearCase performs EBCDIC to ASCII translation transparently, enabling both the Eclipse user and the ISPF user to work in parallel with the same physical repository. The ClearCase paradigm of making all of the elements visible, but read only unless checked out to the user, is extended to the ISPF user. The usual developer actions of select view, check out, and check in all behave in similar or identical fashion.

This means that the primary considerations in planning the implementation of an ESCM solution with ClearCase are architectural and organizational, not technical.

So, we begin by analyzing the existing enterprise software asset base, and determining how these would be organized into a ClearCase repository. There is a considerable body of knowledge available to guide organizations in this process, There is no technical limit on the size of a Clearcase Versioned Object Base (VOB). Performance characteristics of the server and network and the nature of the access to the repository do impose some constraints.

Aside from the size constraints described above, there are no hard technical requirements from ClearCase that would influence this process. Architectural and organizational factors will dictate a logical structure. The build and release process will be a significant factor. The significant point here is that there is no technical requirement from ClearCase *per se*, to either segregate, or co-locate z/OS modules with other assets.

The main consideration will be architectural. We start off with the simplest approach, which is to allocate a top level directory for each subsystem. You can start off the same way. Assets that are shared across subsystems, such as copybooks, database access routines, and so on, need special consideration. Changes to shared assets like these can affect multiple subsystems, and as such may need to be more carefully managed. You can argue for maintaining them in a separate VOB or directory where you may choose to employ more stringent access and approval levels.

3.1.3 Using ClearCase meta data in z/OS builds

We used ClearCase meta data and a set of helper scripts, or alternately, a set of trigger scripts to manage the parameterization of the z/OS builds. An examples of a helper script can be found in Appendix A, “Source script for the build engine” on page 627.

When a build is invoked, the scripts use calls to the ClearCase Automation Library (CAL) to retrieve the meta data, set the relevant parameters, and construct a build script. We will describe this in more detail shortly.

Table 3-1 lists some of the project and element level attributes that we have used to implement the components of the build process. The names and data types are arbitrary, and were relevant to the implementation at the time. You should choose attribute names that are meaningful in your organization. There is no difference between project level and element level attributes as far as how they are created in ClearCase. The distinction simply designates what the attribute in question is used for.

Table 3-1 Project and element level attributes

Name	Type	Value	Function
PGMRNAME	String	SMITH.K	Annotate programmer name.
AUTHPARMS	String		A
CICS	String		A
COPYLIB1	String	TAX.CPY.INTRT	Specify required copybook1.
COPYLIB2	String	DATE.CPY.YEAR	Specify required copybook2.
COPYLIB3	String	N	
LOADLIB	String		
SRCLIB	String		
LANG	String	COBOL	Identifies the compiler.
COBOL390	String		Invoke COBOL390 option.
PGMDESC	String		Name or description of program.
DB2_BIND_MEMBER	String		
PGMSTAT	String		

Assigning element level attributes

In this approach, we assign build information on a member-by member basis, by assigning a set of attributes that varies depending on the characteristics of the source member. We also identify a set of what we refer to as “project level attributes”.

These are used to carry information that applies to the entire subsystem and z/OS infrastructure. Depending on the infrastructure, these attributes will be related to syslib concatenation relating to copybooks and load modules, DB2 subsystem, CICS regions, IMS regions, and so on; anything that is related to the deployment of artifacts to the proper z/OS environment. For example, in a simple life cycle based on Test, QA, and Production environments, the project level attributes would have stored the meta data to know each of the target libraries, CICS regions, DB2 subsystems, These provide all the information needed when building or moving to any of above environments.

Figure 3-2 gives an overview of this process of assigning element level attributes.

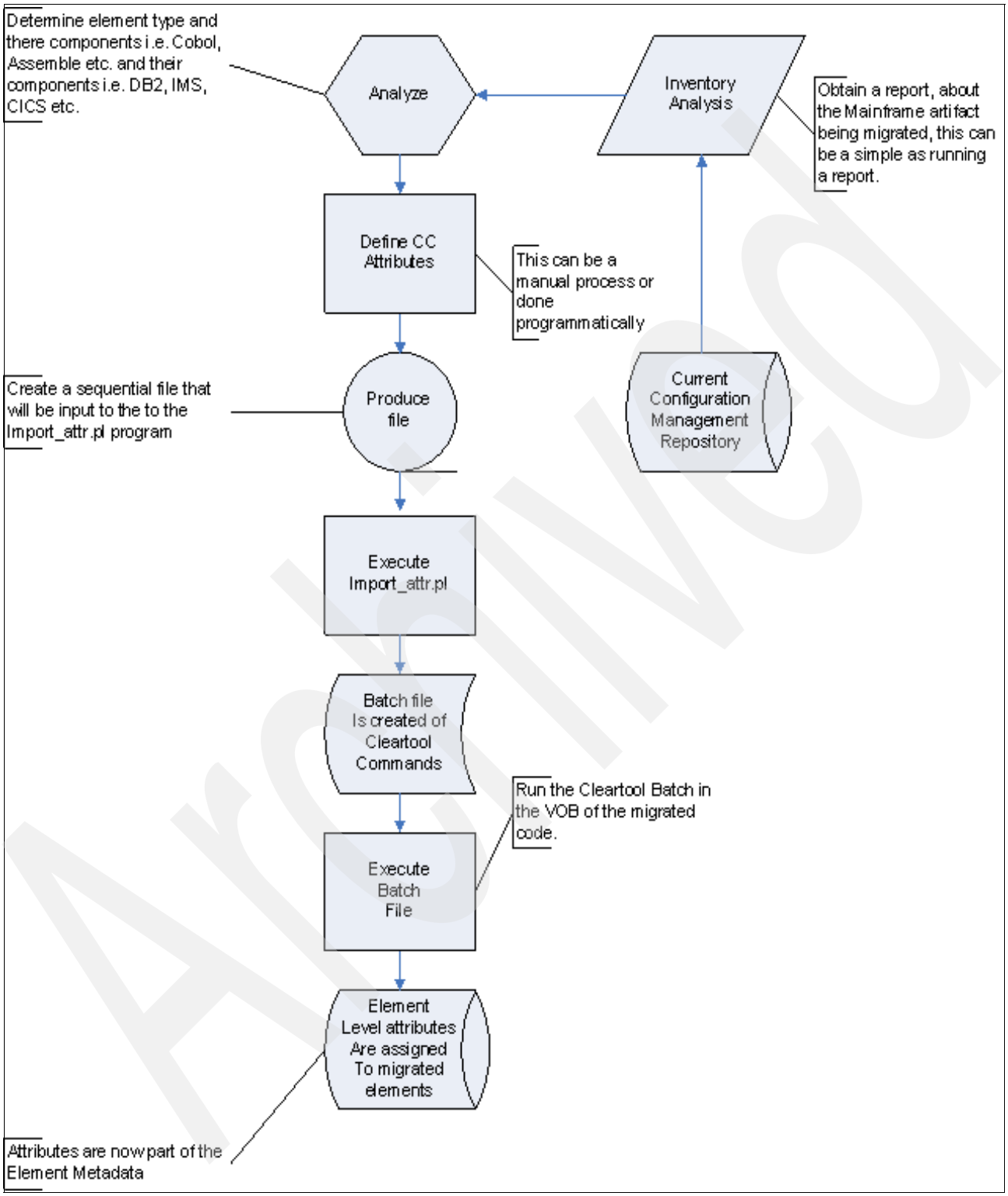


Figure 3-2 Schematic process for importing element-level attributes

Archive

Assigning project level attributes

We use a single ClearCase attribute called ZCCENV. We create a PERL script called SetZCCEnvAttr.pl that sets the value of the attribute. The purpose of this attribute is to define and map the z/OS infrastructure and life cycle on the mainframe. This data is then queried by the build process and used to determine where and how the project will migrate through the life cycle. If a change is needed or a different migration path is necessary as development proceeds, the ClearCase administrator needs only to change the values with the ZCCENV definition for the project utilizing the SetZCCEnvAttr.pl script. The build engine (described later) will use the new values and continue on in the new life cycle, or it will move to the new z/OS infrastructure for testing and QA. This process is illustrated in Figure 3-3.

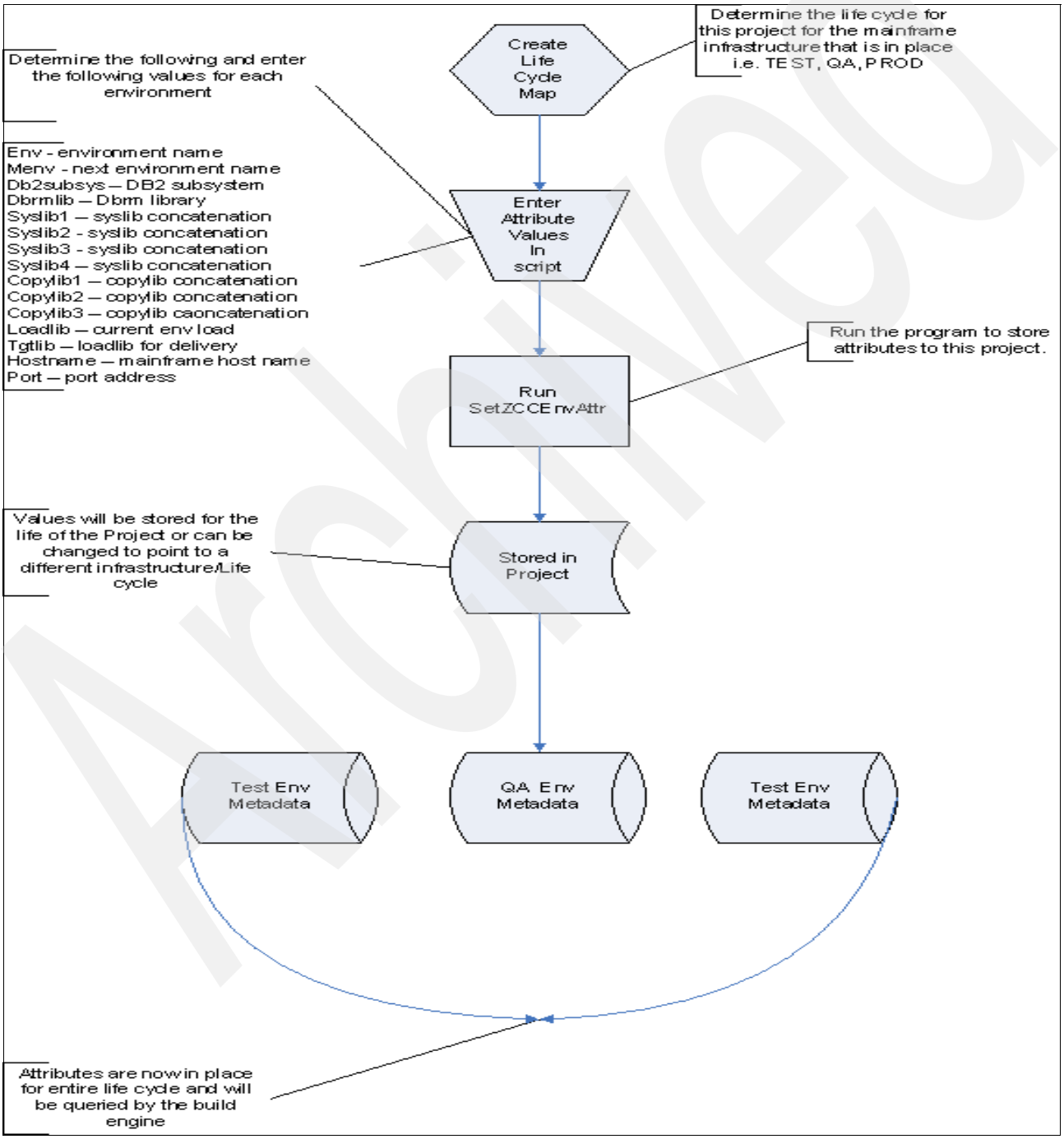


Figure 3-3 Process flow: creating and updating project-level attributes

Archived

Creating your repository

At this point, we have described the points you need to consider in order to determine the preliminary structure of your ClearCase repository. We should emphasize “preliminary”: In our experience, no matter how much effort is put into laying out the structure in advance, it will remain fairly dynamic throughout use, changing requirements as the organization begins to learn and optimize around the use of the solution.

You should plan for this scenario. ClearCase is sufficiently flexible such that in most cases, you can implement a fairly simple model, with few controls initially, and iterate towards a more managed approach, adding appropriate controls and approvals in stages as the team gains experience. We have found that the more successful organizations are those that use an iterative process adoption model.

With these considerations, you should be ready to begin to import assets into your new repository. The ClearCase TSO Client can be used to import a few files, as illustrated in Figure 3-4.

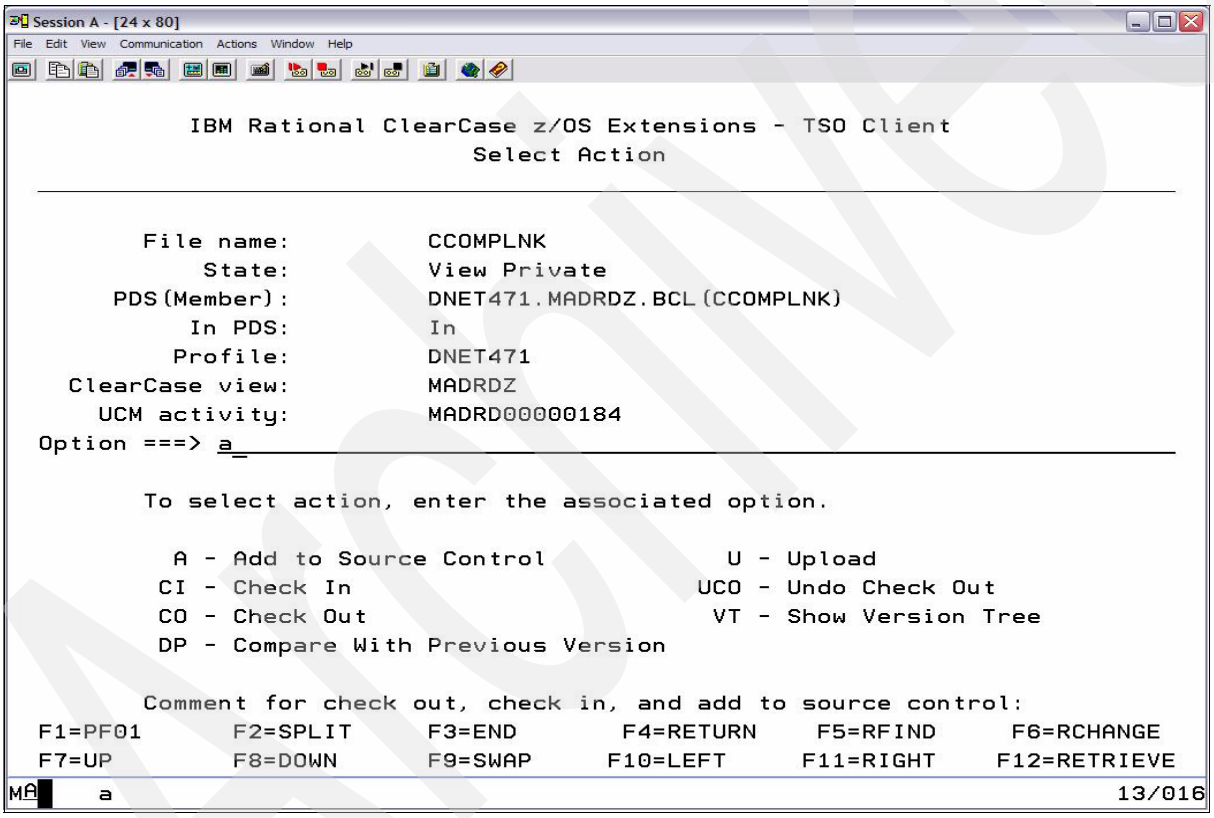


Figure 3-4 Adding PDS Member DNET471.MADRDZ.BCL(CCOMPLNK) to Source Control

Archived

You navigate to the member in question. The state will (or should) be view private. Selecting “A” adds the member to ClearCase, and the window will change to look as it does in Figure 3-5.

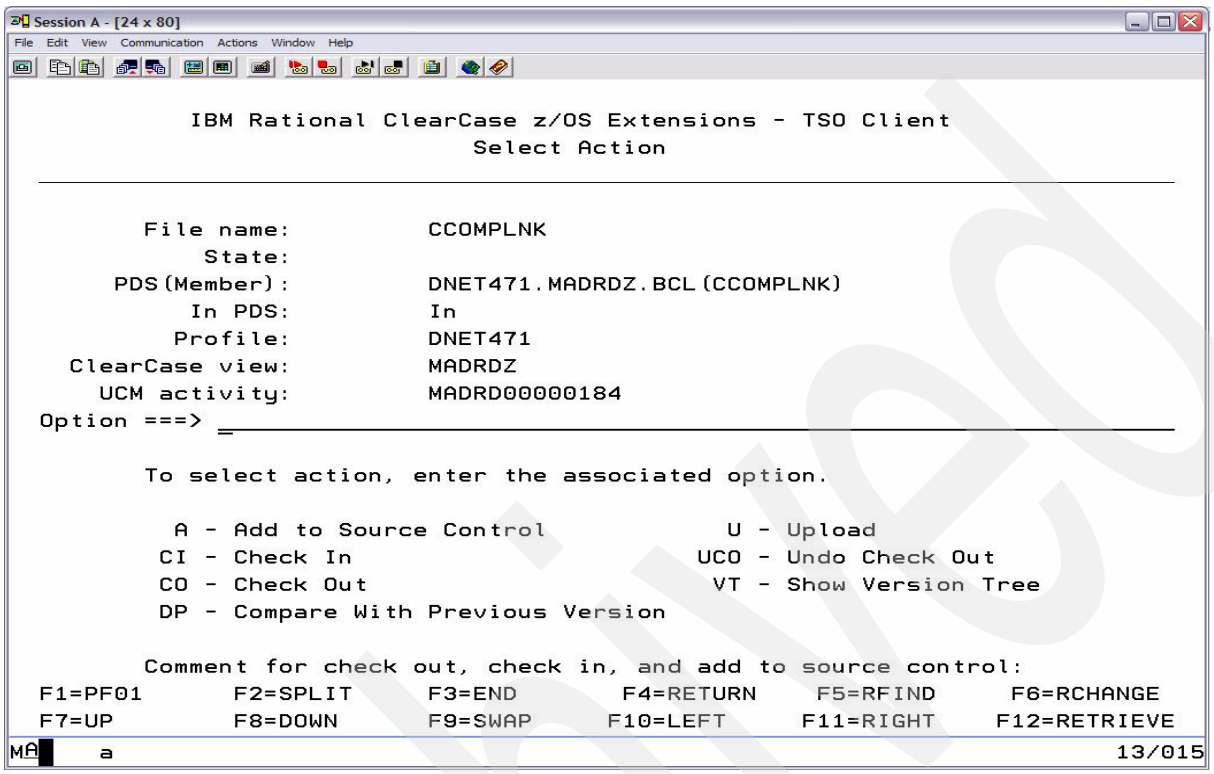


Figure 3-5 Member CCOMPLNK added to source control

Note that the state has changed from “view private” to blank, which means it is now a managed element.

In the most likely scenario, where you importing hundreds (or thousands) of modules, importing z/OS assets is done in three steps:

1. Determine the file system hierarchy.
2. Transfer the assets from z/OS to the distributed file system in the environment where ClearCase is installed.
3. Import the assets into the ClearCase repository.

Determine file system hierarchy

There are a host of references that can help you determine the file system hierarchy. If you are new to ClearCase, we recommend reading *Managing Software Projects IBM Rational ClearCase*, which you can find at:

ftp://ftp.software.ibm.com/software/rational/docs/v2003/cc_family/ccase/doc/all/cc_proj/wwhelp/wwhimpl/js/html/wwhelp.htm

Factors to consider in your file system hierarchy are:

- Organization: Which teams are responsible for which subsystems?
- Architectures: What subsystems are interdependent and need to be built or managed together? What subsystems can or should be managed separately?

Archived

- Security and access: Who should be allowed view or modify assets? Which assets should they be allowed to modify?

Define build and promote strategy

Although many mainframe shops we work with employ a build once and promote strategy, there are those who build at each stage. From the perspective of ClearCase, this determines what host environments you will be delivering into, and how you want the action recorded.

The definition and configuration of your host environments should be what drives your ClearCase stream or branching strategy. Regardless of whether you build once and promote the same binaries through the life cycle, or if you rebuild and rebind at each promotion step, we found that the most effective approach is to create a ClearCase integration branch or stream for each environment you deliver to.

If you build only once, you would identify a single stream or branch from which you deliver to your initial target environment and deliver to the initial target. Subsequent promotes can be performed on the host by the existing processes, or you can perform promotes in ClearCase with ClearCase deliver or merge actions on the load modules, then deliver into the target environment. For example, if you have environments called “STG”, “STG2”, “PREPROD”, and “PROD, using the same names for your ClearCase streams or branches makes life much easier for everyone on the team.

Figure 3-6 illustrates some of these alternatives. We will revisit this topic when we discuss build management in detail later in this chapter.

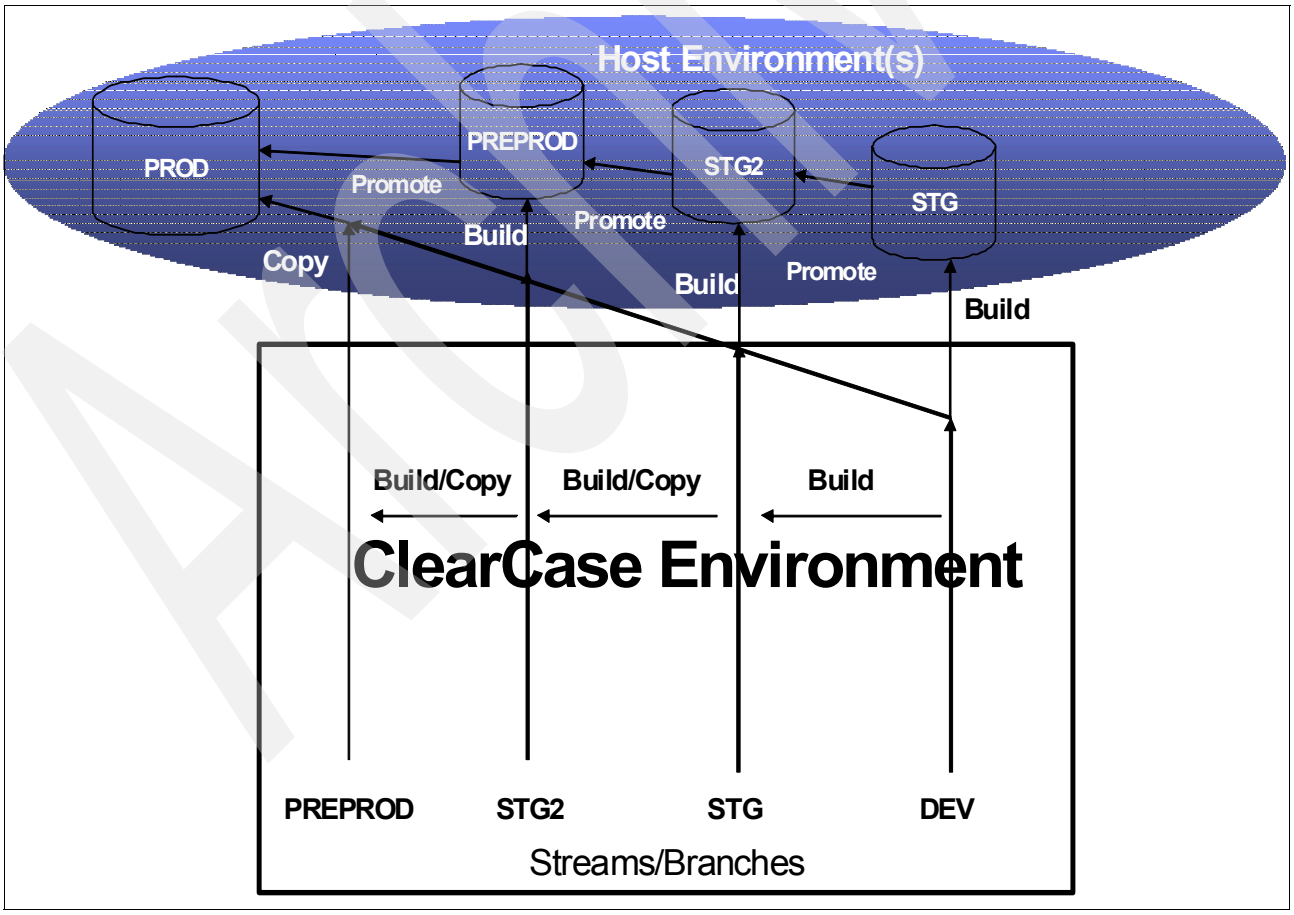


Figure 3-6 Stream strategies for delivery to z/OS

Transferring the assets

There are two steps to this; physically transferring the assets and converting them from EBCDIC to ASCII, as shown in Figure 3-7. There are many solutions available to make this a simple process. For example. IBM Rational Developer for z (V7.1 and later) contains a remote synchronizer that you can use to automate the transfer and synchronization of members between the local Rational Developer for System z workspace and the host.

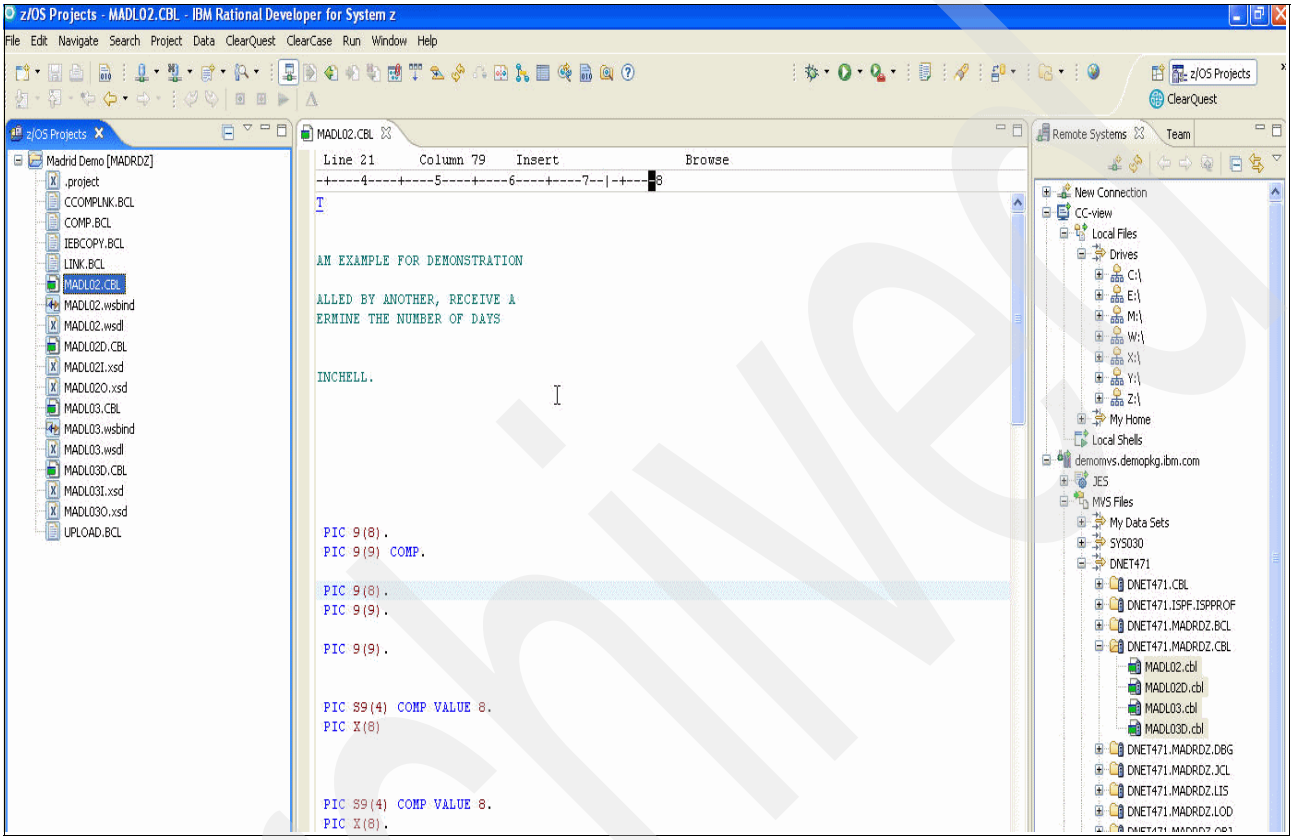


Figure 3-7 Transfer of assets

Note that in order to insure that binary files (object or load modules) are not corrupted, you may have to write a script that calls TSO XMIT.

Importing assets into IBM Rational ClearCase

Once you have transferred your assets to the ClearCase distributed environment and converted them to ASCII, you are ready to import them into your ClearCase environments.

At this stage, you will have already defined and created your VOB layout. The next step is to create a view and import the assets.

Rational ClearCase includes several features to automate the importation of large numbers of file members. With the **`clearfsimport`** command, you can import a hierarchical file system into ClearCase in a single step; consult the ClearCase manual page entry for **`clearfsimport`** for more details.

You can use the ClearCase Explorer GUI to drag and drop small numbers of files into a ClearCase VOB. More details about importing assets into ClearCase can be found in the *ClearCase Administration Manual*, found at:

ftp://ftp.software.ibm.com/software/rational/docs/v2003/cc_family/ccase/doc/all/cc_admin/wwhelp/wwhimpl/js/html/wwhelp.htm

3.2 Managing your z/OS assets with ClearCase

To enable work with z/OS assets, Rational ClearCase includes a facility called Remote Build. Remote Build enables you to submit z/OS batch jobs directly from the desktop. You can use it to build assets managed by ClearCase or host-based assets. Regardless of where the assets are, using the ClearCase Remote Build features enables you to create ClearCase audit records of builds run on the host. We will go into significant depth on this topic in subsequent chapters.

The second feature, the TSO Client, enables you to perform basic ClearCase SCM operations, like select activity, compare versions, and check out and in from a 3270 emulator. ClearCase manages a shadow PDS and populates it with the versions of assets selected by your view's config spec. You can work with both UCM and non-UCM views, snapshot or dynamic. As of ClearCase V7.01, there is no support for Web- or Eclipse-based views.

There is a separate manual, *Rational ClearCase Guide to Installing and Implementing z/OS Extensions*, GI11-6714, which explains the installation and configuration of the extensions in detail. This is part of the ClearCase product documentation set and will be referred to frequently throughout this section. If you are not familiar with this document, it can be found at:

http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=GI11-6714-01

3.2.1 Remote Build

As mentioned above, with the proper privileges, you use Remote Build to submit build scripts to your host system.

The Remote Build architecture is simple. There is a z/OS agent (or started task) that runs on the target z/OS system. Remote Build uses a secure socket and pipe protocol to communicate with and transfer data to and from the host. Remote Build does not rely on either NFS or FTP. It uses POSIX pipes and sockets to communicate with the ClearCase view server over a defined port.

Both z/OS and UNIX System Services versions of the Remote Build server are included as part of the standard ClearCase product. You can submit batch jobs directly to z/OS and you can remotely run UNIX System Services scripts; for this book, we focused on the z/OS version.

In the z/OS environment, this ability to run any JCL in this manner would present a risk of unintentional consequences, so we instead support a JCL-like subset called Remote Build Control Language (BCL). A complete description of BCL, and the differences and limitations is in Chapter 8, "Working with build scripts", of the *Rational ClearCase Guide to Installing and Implementing z/OS Extensions*, GI11-6714.

In the distributed ClearCase environment, Remote Build is a stand-alone executable (rccbuild.exe) that is installed as part of a ClearCase fat client install. In the simplest case, Remote Build can be executed from the command line in a ClearCase-managed directory. Remote Build manages the delivery of the source member and associated assets (build scripts and so on) to the host, and the return of the build products (listings, object, and load modules) to the repository where they are be registered as derived objects.

You can use existing build scripts with BCL, but they will most likely need rework. PROCs need to be expanded and inlined, for example, and there are limits on how you perform concatenations.

Figure 3-8 is an example of a simple BCL script that does a compile and link of a COBOL module.

```
//COMP         EXEC PGM=IGYCRCTL,REGION=4M,
//                PARM=('EXIT(ADEXIT(ELAXMGUX)),
//                ADATA,LIB,TEST(NONE,SYM,SEP),
//                LIST,FLAG(I,I)')
//STEPLIB   DD DISP=SHR,DSN=DNET100.ADLAB.LOAD
//          DD DISP=SHR,DSN=SYS030.EPS.LOAD
//          DD DISP=SHR,DSN=CICSTS.V3R1.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=COBOL.V4R1.SIGYCOMP
//          DD DSN=COBOL.V3R4.SIGYCOMP,DISP=SHR
//          DD DSN=RATD4Z.V7R1.SFEKLOAD,DISP=SHR
//          DD DISP=SHR,DSN=CICSTS.V3R1.CICS.SDFHLOAD
//          DD DISP=SHR,DSN=COBOL.V4R1.SIGYCOMP
//          DD DISP=SHR,DSN=DB2.V8R1.SDSNLOAD
//SYSLIB    DD DISP=SHR,DSN=SYS030.EPS.COBINC
//          DD DSN=CICSTS.V2R3.CICS.SDFHCOB,DISP=SHR
//          DD DSN=CICSTS.V2R3.CICS.SDFHMAC,DISP=SHR
//          DD DSN=CICSTS.V2R3.CICS.SDFHSAMP,DISP=SHR
//SYSIN DD DSN=&USR..MADRDZ.CBL(MADLO1),DISP=SHR,RCCEXT=CBL
//SYSLIN DD DSN=&USR..MADRDZ.OBJ(MADLO1),DISP=SHR,RCCEXT=OBJ
//SYSADATA DD DUMMY
//SYSPRINT DD RCCEXT=RCCOUT
//SYSUT1   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7   DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSDEBUG DD DSN=&USR..MADRDZ.DBG(MADLO1),DISP=SHR,RCCEXT=DBG
//SYSXMLSD DD DUMMY
```

Figure 3-8 Example of using BCL to compile and link a COBOL program

There are more examples of BCL scripts in Appendix B, “Sample rccbuild commands”, of the *Rational ClearCase Guide to Installing and Implementing z/OS Extensions*, GI11-6714, found at:

http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m1/topic/com.ibm.rational.clearcase.help.doc/zos_extensions.pdf

In most implementations, Remote Build is almost never called directly. Depending upon what other solutions are being deployed, there are several ways to automate the invocation of Remote Build:

1. ClearCase with Remote Build
2. Build Forge with Remote Build
3. ClearQuest with Remote Build

4. Rational Developer for system z with Remote Build

We will discuss each of these in the following sections.

ClearCase with Remote Build on z/OS

This is the most basic approach, and is not dependent on any other solution. This is also the only means to have direct interaction between the host and a ClearCase repository, and to have ClearCase manage and record the build step. Since this is frequently a requirement, the other solutions described below usually depend on this step.

Managed builds: The simple scenario

The core of the process is a build script, or fragment similar to the BCL sample above. Clearly the nature of the build depends on many factors, such as the target environment, binding to a database or CICS region, and so on. For simpler cases, managing the build process can be achieved with variable parameters. The &USR variable in Figure 3-8 on page 81 is an example of this. You specify the value of &USR on the command line like this:

```
rccbuild -h hostname@port <rest of parameters . . > &USR=FRED
```

When this script executes, it will expect to find a data set named FRED.MADRDZ.CBL, FRED.CBL.OBJ, and so on.

Managed builds: The build engine

We have come up with an approach where we take advantage of the meta data management features of ClearCase to automate the management of the builds. We create a template Perl script that we adapt at each implementation. We refer to this script as a “build engine”; it is a Perl script that is run against each source element. For each member to be built, the engine reads the values of some predefined ClearCase attributes (that we create in advance) and uses them to determine element-level build properties, such as compiler settings, and project level build properties, such as target load libraries, SYSLIB concatenations, DB2 subsystems, and so on.

The attribute values are inserted into BCL statements, which are in turn assembled into build scripts, which are submitted to the host with the ClearCase Remote Build Feature. Figure 3-9 on page 83 illustrates a high-level diagram.

Archived

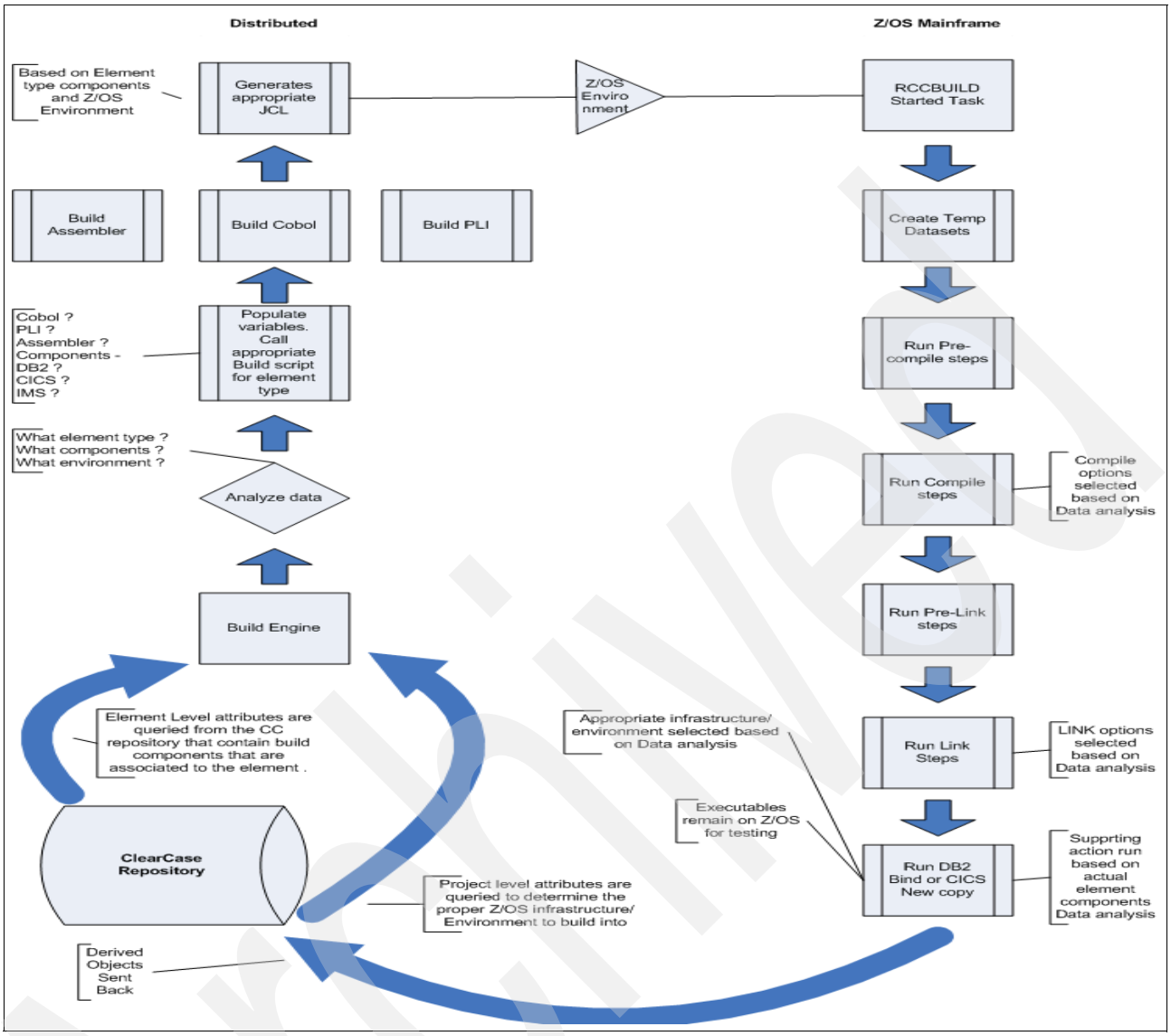


Figure 3-9 Build Engine: High-level overview

With this approach, the developers do not need to concern themselves with how to build or how to deploy; the build engine, using the information stored with the ClearCase element, knows this. The script is called Host-Build.pl. An example is provided in Appendix A, “Source script for the build engine” on page 627.

Archive

For performance reasons, we use the ClearCase Automation Library to retrieve the project and element level attributes of the source member (ClearCase element). The code fragment in Figure 3-10 illustrates this technique. Here we retrieve element attributes from a ClearCase element name that is passed in by the calling program. This could have been passed in from a ClearQuest hook, a ClearCase trigger, or from a customization of the ClearCase context sensitive menus.

```
my ($ccpn) = $ARGV[0];
my $CCApp= Win32::OLE->new ("ClearCase.Application") or die "Can't connect to
ClearCase object $!";
my $version = $CCApp->Element($ccpn) or die "Failed to retrieve element";
my $attlist = $version->Attributes;
my $attcount = $attlist->Count;
for ($i=1; $i<=$attcount; $i++) {
$myatt = $attlist->Item($i);
$myattname = $myatt->Type->Name;
$CCattr{$myattname} = $myatt->Value;
}
```

Figure 3-10 Retrieving an element attribute with the ClearCase Automation Library

Messages are sent to STDOUT. This is echoed to a command window in the user's desktop environment. At the end of the build process, the window remains open so that the developer can review the outcome.

The full script is available as part of a field implementation kit. A complete version of it is available in Appendix A, "Source script for the build engine" on page 627.

Working with the Host-Build.pl script

You cannot just copy the build engine into your environment and run it. We provide it here as an illustration. This version is a working prototype that runs in the IBM internal demonstration environment. We provide it as part of field service engagements to assist clients with this process. Much of the work is in modifying it to your build processes and environments.

In planning the implementation of your process, you should already know how you will be invoking this script: From a ClearQuest hook, from a ClearCase trigger, from a Build Forge step, from an Rational Developer for System z builder, or from some combination of them. You need to identify the build steps and any existing JCL you intend to use,

You should also know what target environments you intend to deliver to. If you are delivering to environments managed by other mainframe SCM solutions, you will need to ensure that the TSOID of the started task or (if you are running in authmode 2) of the user(s) running the build will have the appropriate authority to access or modify the target environments.

You need to know your build strategy. For example, will you build once into an integration environment, and then promote to succeeding environments, or will you build at each promotion step?

As you define each of your build steps, or processes, you modify and extend the Perl subroutines in the sample to create subroutines with the appropriate build steps, data set definitions, and concatenations for your environment.

The build engine script is designed to be callable from a ClearCase trigger, a ClearQuest hook, or by customizing the ClearCase Explorer menu to include a “compile” or “build” option, as shown in Figure 3-11 and Figure 3-12 on page 86.

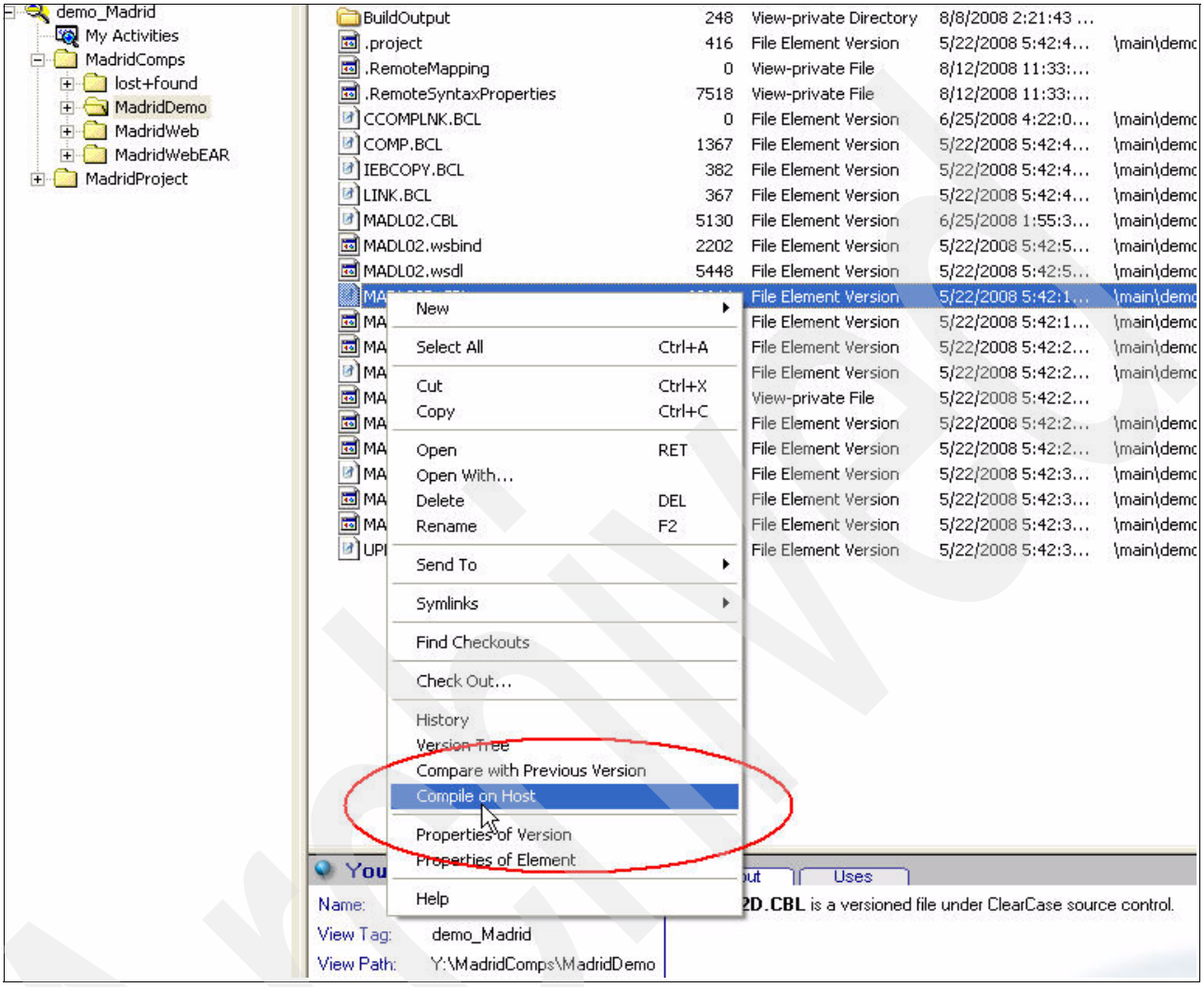


Figure 3-11 ClearCase Context menu customized to support z/OS compiles

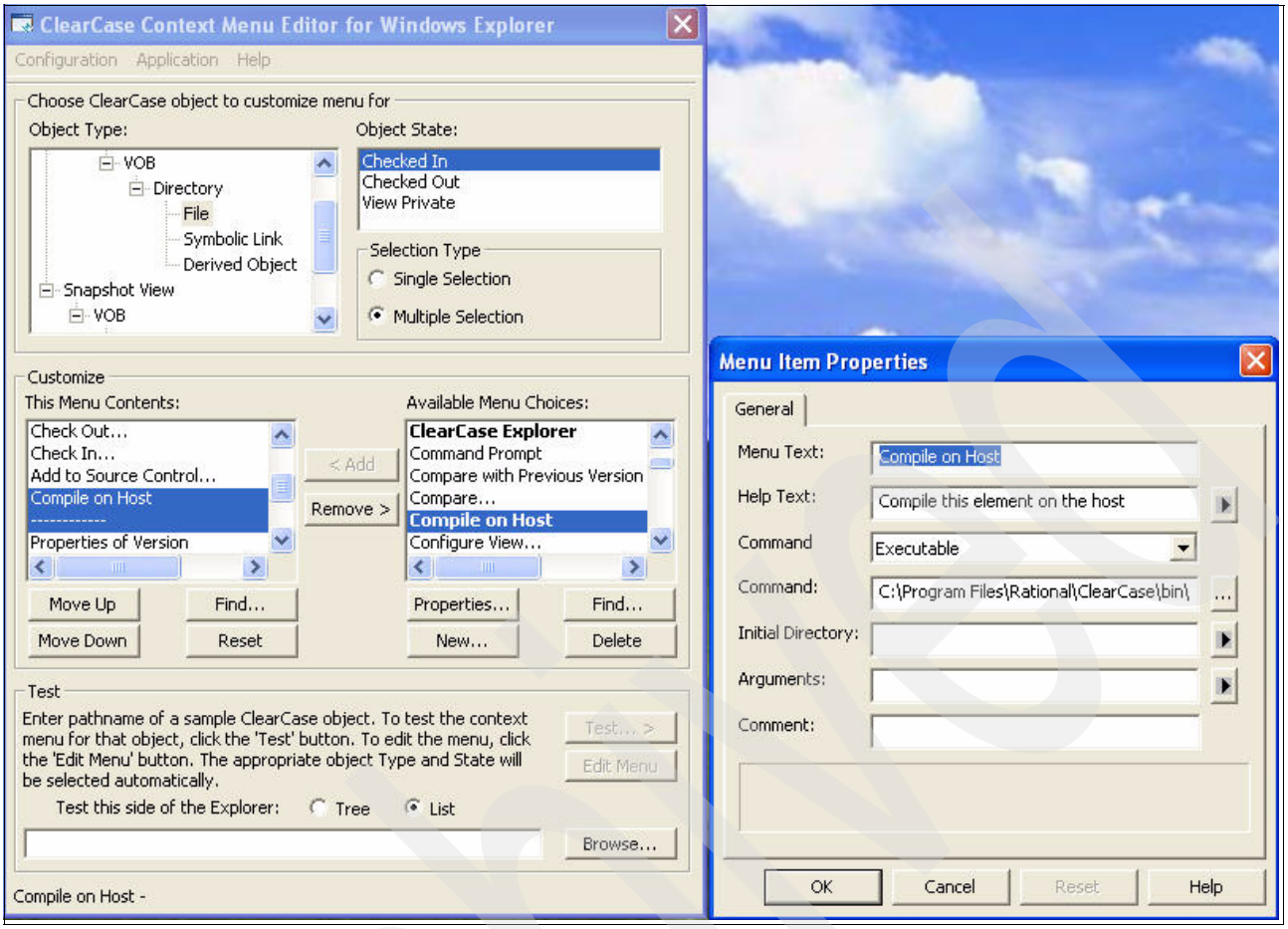


Figure 3-12 Customizing the ClearCase context menu

This is the approach we take when there is a requirement that a user be able to perform builds of a selected member or set of members from the desktop.

Bear in mind that Remote Build is really a synchronization and transport mechanism. As such, you can extend this approach to build an entire subsystem, or a cross-platform application, depending on how you choose to configure your build scripts.

ClearCase with Remote Build on UNIX System Services

If configured, you invoke the UNIX System Services Remote Build server the same way as the z/OS server, but the port number, and possibly the host name, will be different.

Instead of passing BCL scripts, you instead pass REXX, Perl, or shell scripts that execute in the UNIX System Services shell environment.

Remote Build: Hints, tricks, and security issues

Remote Build depends on a server task that runs on the host. This can be either a started task or a batch job. You need to consider the security implications when configuring the server task. This task must have authority to submit jobs, and to access whatever host resources that will be requested during the process.

The server task can be run in three security settings (authmodes). The security settings are:

- 0 - No user authentication. The user ID that starts the Remote Build server task on the host is used for build processes requested by all users.
- 1 - The user ID and password, passed by the client, are optional. If supplied, they are verified with the security product (that is, RACF® or others).
- 2 - The user ID and password, passed by the client, are required. The security product (that is, RACF, or others) verifies them.
- S. The user ID and password, passed by the client, are required. The security product (that is, RACF or others) verifies them, and an optional role can be supplied on the build request.

You need to carefully consider the privileges you grant to the TSOID that runs the Remote Build server task. The Remote Build server task needs RACF authority to create some scratch PDSs in the installation area, and to create members in the target PDS.

3.2.2 ClearCase TSO Client

As mentioned above, ClearCase includes an ISPF client that enables the developer to work with ClearCase managed assets locally in a private PDS. The TSO Client is similar in usage to the CC Web client, as shown in Figure 3-13.

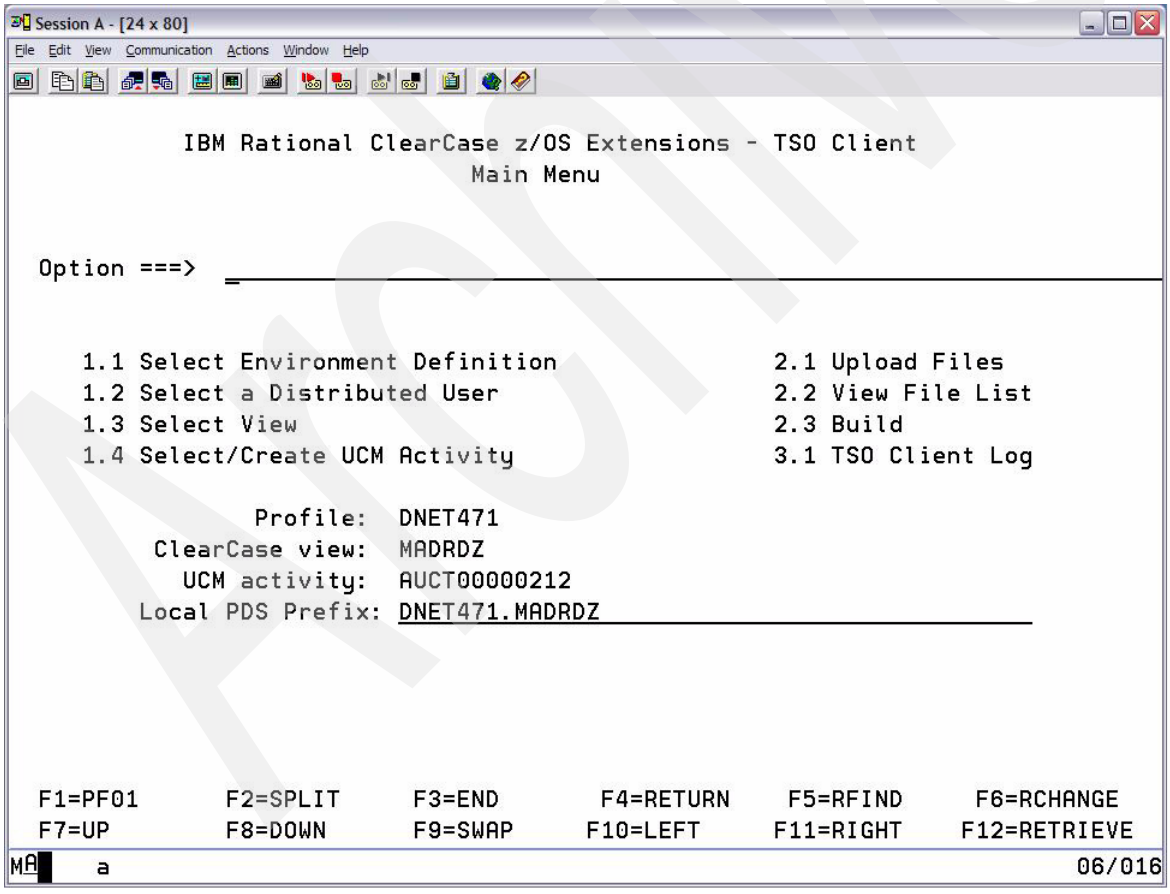


Figure 3-13 ClearCase TSO Client Main Menu

Archived

As with the Web client, you work with the TSO Client by identifying a set of controlled members that you want to work with, and uploading (or shadowing) them into a set local copy area (A set of PDSs, in this case). The versions you get are selected by the ClearCase view to which you attach your TSO Client session.

The TSO Client knows the state or the members you upload. As with any other ClearCase portal, members are read-only until you check them out.

Once your copy areas are populated, you perform developer level tasks, that is, check out, edit, and check in directly in the TSO Client. ClearCase operations are immediately and transparently echoed back to your ClearCase view as you invoke them.

Considerations for using the TSO Client

You should consider the TSO Client as another portal from which you can perform typical developer commands, such as adding members to source control, check out, check in, comparing different versions of a controlled element, and so on.

You can also execute a build script, either locally (submitting a build script that resides in the local PDS on the host) or remotely (invoking Remote Build on the ClearCase view server to submit a build script from the ClearCase repository as though you were calling Remote Build from ClearCase). It is more likely that you will be running builds from the ClearCase environment, or with a combination of ClearQuest hooks, Build Forge steps, or Rational Developer for system z Builder steps.

The ability to perform administrative functions is also very limited; your release engineering team should plan to use one of the distributed clients for administrative work.

Preparing to use the TSO Client

Installation and configuration of the host to use the TSO Client is explained in *Rational ClearCase Guide to Installing and Implementing z/OS Extensions*, GI11-6714, found at:

`http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=GI11-6714-00`

Once the product is installed and configured, you need to create the user workspaces on the host and the corresponding ClearCase views. The usage model for the TSO Client is similar to that for using ClearCase in general. In the same way that we recommend creating separate views for each activity, you should maintain a separate view for an exclusive user of the TSO Client. You should not make any modifications to elements in this view except through the TSO Client.

Once you have all of the components installed, you need to make a few decisions about how you will work with TSO Client. Each user should create, or have created for them, an environment definitions file like the example shown in Figure 3-14 on page 89.

Archived

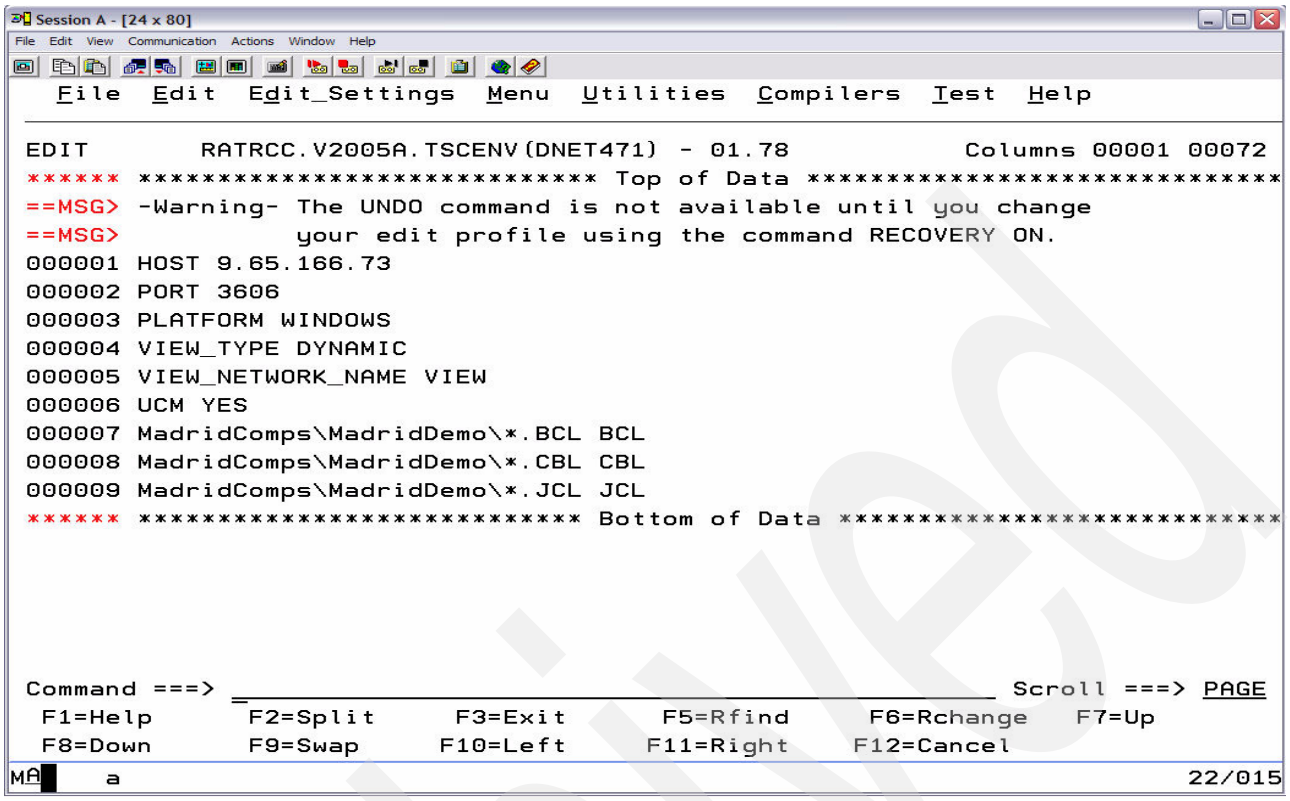


Figure 3-14 TSO Client environment definitions

The fields are explained in *Rational ClearCase Guide to Installing and Implementing z/OS Extensions*, GI11-6714 and are pretty straightforward, with the probable exception of the mapping statements (lines 7-9), so here is a quick explanation.

The line contains two fields. The first field is the vob-extended path name of the element(s) you are mapping. You use the file extension to determine which data set the element version is destined for. Note that any elements that you intend to work with using the TSO Client must have a file extension. Usually these are three letters and commonly understood, that is, CBL for COBOL or PLI for PL/I. You can use any extension you want, but you do need to have one.

The second field is the low-level qualifier of the data set where the element will be shadowed to. The TSO Client appends this field to the Local TSO Prefix shown in Figure 3-14. In the example above, Line 7 maps all of the elements in the ClearCase directory MadridComps\MadridDemo with an extension of BCL to a data set called DNET471.MADRDZ.BCL.

You can create the data sets in advance, or have the TSO Client create a PDS for you automatically.

Once you have configured your TSO Client environment, you can invoke the TSO Client application by navigating to the installation PDS, which is usually named something like <HLQ>.<V200x>.TSCRX and execute the member called RCCTSOR. Your sysprog may provide an alternate means for invoking the TSO Client.

Archived

When the client starts, assuming your environment is configured correctly, you will see a window that looks like Figure 3-15.

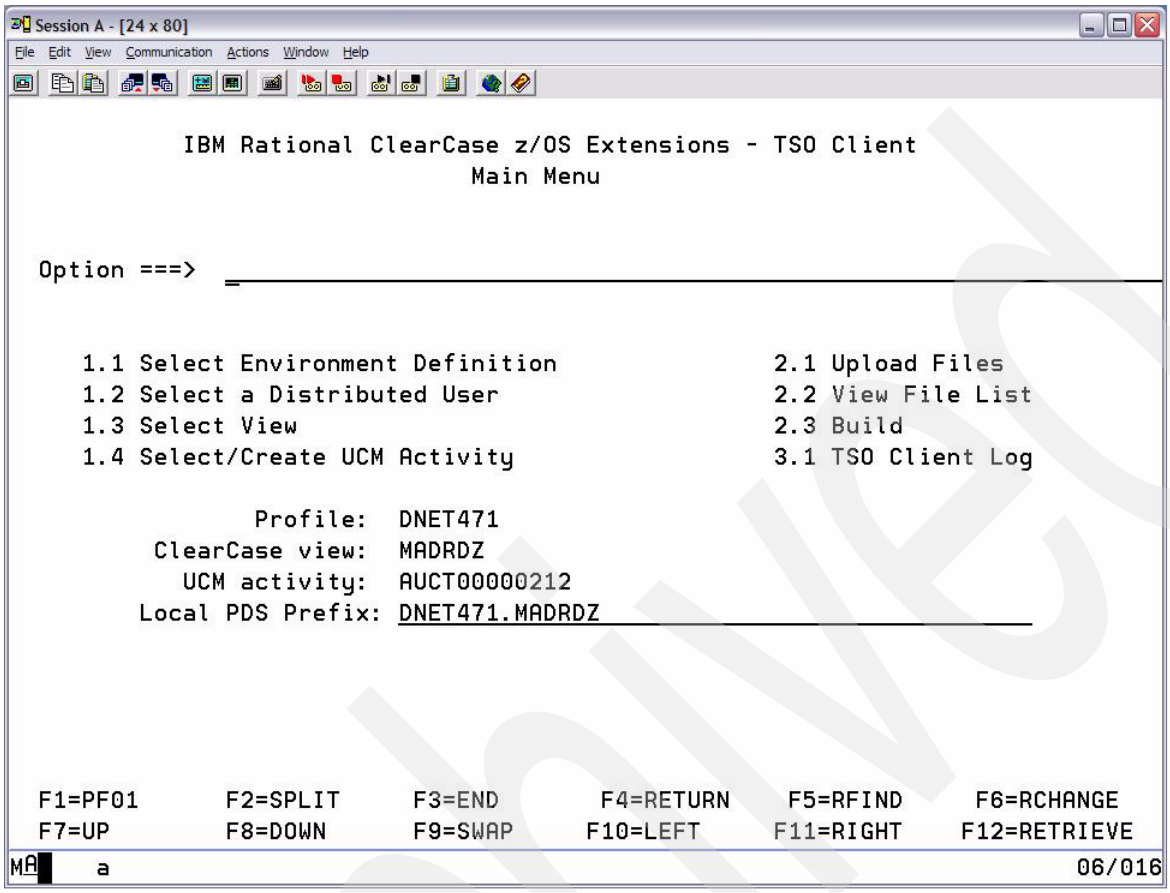


Figure 3-15 TSO Client initial window

Before you can use the TSO Client, the 4 fields above (Profile, ClearCase view, UCM Activity, and Local PDS Prefix) need to be completed. You need to proceed through panels 1.1 through to 1.4 in order before you can begin work. Note that in this example, because we set “UCM” to “YES” in the environment definitions file in Figure 3-14 on page 89, panel 1.4 requires us to select an activity. You can work in base ClearCase views by setting “UCM” to “NO”, and you will not be prompted for activity information.

The TSO Client remembers all of this information, so it is not necessary to go through 1.1 to 1.4 at each invocation.

Most of your work will be off of panels 2.2. To begin work with ClearCase controlled members, from panel 2.2, you are presented with a window to select the data set you wish to work in. Choose the data set, and the window shown in Figure 3-16 on page 91 appears. In the next window, the name of the controlled member appears, with a set of options and commands that will look familiar to an experienced ClearCase user, but may require some explanation to an ISPF user seeing ClearCase for the first time. Of most significance are the CO and CI actions. You also use this window for uploading (for example, refreshing your data set version of the member with the version stored in the repository), and to view the version history and compare differences.

Obviously, ISPF being what it is, you get a text-only versions of the version tree and different output.

Archived

Nonetheless, the TSO Client enables those who are most comfortable in a green screen setting to use the same solution and repository as the rest of the team to perform normal day-to-day SCM functions.

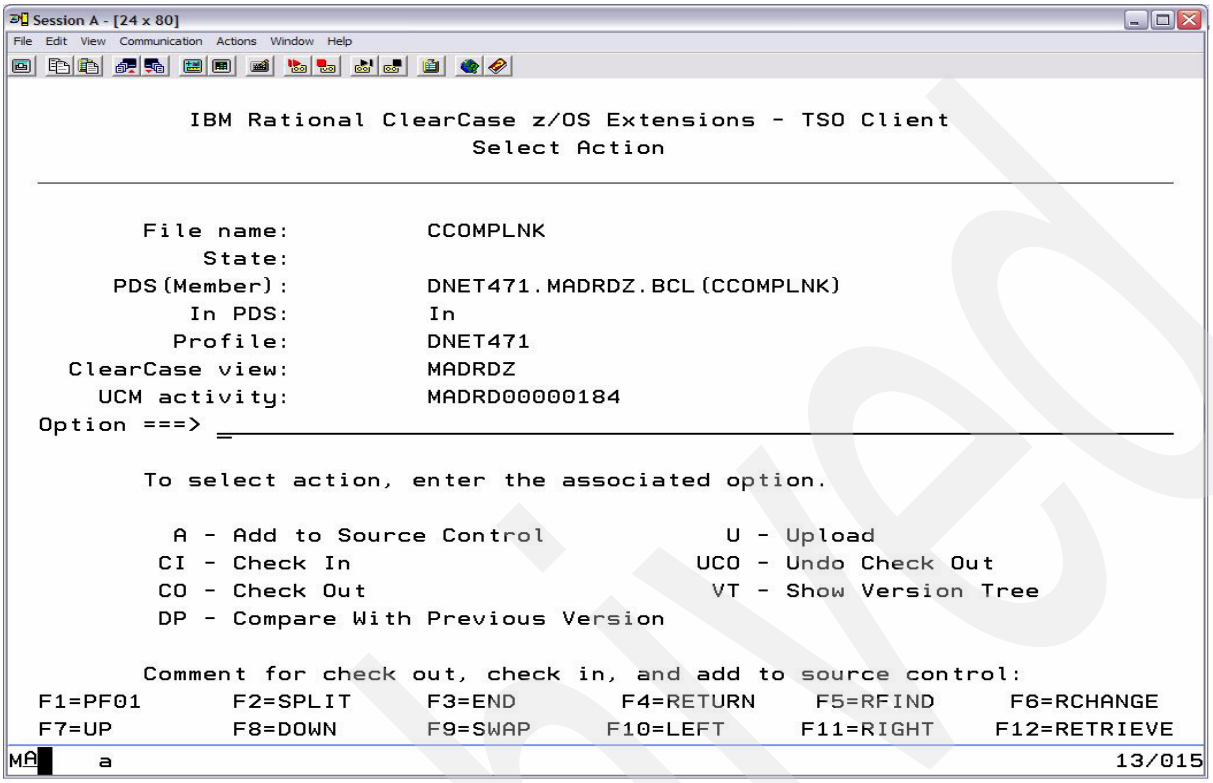


Figure 3-16 Green screen

3.2.3 Programmatic access to the ClearCase repository from z/OS

You do not need to use the TSO Client to work with ClearCase-managed data from the host. There is a published Application Programming Interface (API). The API is described in Appendix D, “The TSO Client API”, in *Rational ClearCase Guide to Installing and Implementing z/OS Extensions*, GI11-6714.

Although it was not in the scope of the project for this book, we have a large financial customer who, for audit reasons, had a requirement for an off-host means of shadowing their z/OS production assets. They implemented this by inserting calls to the ClearCase TSO Client API into the screens of their existing host-based SCM solution. As their developers edit and deliver their z/OS assets, calls to the TSO Client API echo analogous commands to ClearCase, performing checkouts, additions to source control, checkins, and so on. This transparently created and maintained a shadow ClearCase repository, and helped them close out a significant audit issue.

You can perform many of the typical user-level ClearCase commands on the distributed system, such as uploading and downloading members, adding to source control, checkout and checkin, and so on. The complete set is described in detail in Appendix D, “The TSO Client API”, in *Rational ClearCase Guide to Installing and Implementing z/OS Extensions*, GI11-6714.

Like the TSO Client itself, you should consider the TSO Client API as another portal for performing typical developer functions.

If you intend to use the TSO Client API, there are some preparatory steps necessary.

TSO Client: Hints and tips

Here we discuss some useful TSO Client hints and tips.

Connection issues

As mentioned earlier, communications between the TSO Client and the ClearCase view server running on the distributed system is over secure POSIX pipes and sockets. Remote Build uses the same mechanism; in fact, they are essentially two portals at either end of a “tunnel”.

For security reasons, the TSO Client performs a series of tests to ensure that it is communicating with the intended ClearCase environment. The tests are fairly stringent, with the result that at times, it is difficult to establish a connection between the host and the distributed system.

Both the host and the ClearCase view server need to be known to each other both by IP address and by host name. When the TSO Client is started up, it pings the host name or IP address supplied in the HOST parameter in the environment definition file, as shown in Figure 3-14 on page 89. In order for the TSO Client to work, your distributed system must be “known” to the host, by whatever name resolution service is employed, at the very least in the DS member TCPIP.HOSTS.LOCAL, or with whatever name resolution service is running.

It is likely that you may see the following message in the process if you encounter connection problems:

```
Error contacting TSO Client server. See the log for more details.
```

You access the TSO Client log from panel 3.1 off of the main TSO Client menu. The latest entries are appended to the end of the file, so you need to press F8 to get to the end. Most of the traffic between the TSO Client and the distributed system is logged here.

When you open the log, you will see a message like the one shown in Example 3-1.

Example 3-1 Log message

```
The host name, <hostname>, for the RCCBLDS remote host
cannot be resolved.

 Recovery:

   - Verify that the remote host server's name and
     address are included in the current workstation's
     tcp/ip hosts file or the data files of the name server.
     If the host name and address appear in either of these
     files, the network or name server may be experiencing a
     temporary problem.

+++++End execution of testServer+++++
```

This means that the TSO Client could not connect to the distributed server where the ClearCase view server is running, probably because either the IP address or the distributed host name were not known to the host. You can troubleshoot this by issuing **`ping`** commands from both machines. Be sure to use both the IP address and the host name you entered. In order for the TSO Client to function, both the IP address and the host name of the view server and the z/OS host have to be known to each other.

Synchronization issues and recovery from errors

If you exit abnormally out of the TSO Client (such as by cancelling your session instead of exiting by pressing F3), it is possible that the PDS that the TSO Client is shadowing your managed files in can get out of sync with the ClearCase view.

Manually removing the TSO Client cache often fixes this. The cache is located in the installation PDS in the TSCUSER directory. If you do not have authority to do this yourself, ask your sysprog to remove the file TSCUSER.<yourTSOIC>.

You can also safely remove the following members in your private area:

```
<username>.TSC.CMD
<username>.TSC.JCL
<username>.TSC.LOG
```

These steps should clear up any residual issues from an improper exit from the TSO Client.

In the worst case, and as a last step, manually clear and recreate your local copy areas.

3.2.4 Using ClearQuest to manage the application life cycle

We often get requests to assist customers with implementing a Lifecycle Management Solution. The approach we take involves incorporating a build solution as a step in an overall life cycle managed by ClearQuest.

If you plan to implement an activity-based application life cycle, you need to include ClearQuest in the solution. In an activity-based life cycle, the goal is to ensure that all work and modifications take place in the context of a defined activity, such as a change or enhancement request, production issue, or a defect. The ClearQuest Unified Change Management schema (UCM) implements activity-based life cycle management for software development.

You can implement an Enterprise SCM Lifecycle in many ways, depending on your requirements.

We usually found that the most expedient approach is to begin with the UCM schema and augment it with a combination of the BuildTracker package or the CrossPlatform SCM package. To insure adequate security and control over the turnover process, we strongly recommend incorporating the ClearQuest Electronic Signature package.

With Version 7.1 of ClearQuest, the ClearQuest Application Lifecycle Management (CQALM) package became available. The CQALM package is intended for comprehensive enterprise-level automation of an Application Lifecycle. It is a significant transition of the role of ClearQuest from a software development-focused issue tracking solution to an enterprise-level workflow management solution.

There is considerable documentation on CQALM available, much more than can be reproduced here. There is a book on CQALM currently being printed. In the meantime, you can see a full discussion of CQALM on IBM developerWorks at:

http://www.ibm.com/developerworks/rational/library/edge/08/mar08/pampino-pierce/index.html?S_TACT=105AGX15&S_CMP=EDU

We used the CQALM schema in the project we used for creating this book.

Archived

Figure 3-17 shows the schema with a shot of the ALM activity record we created. We used ALM activities as the work items.

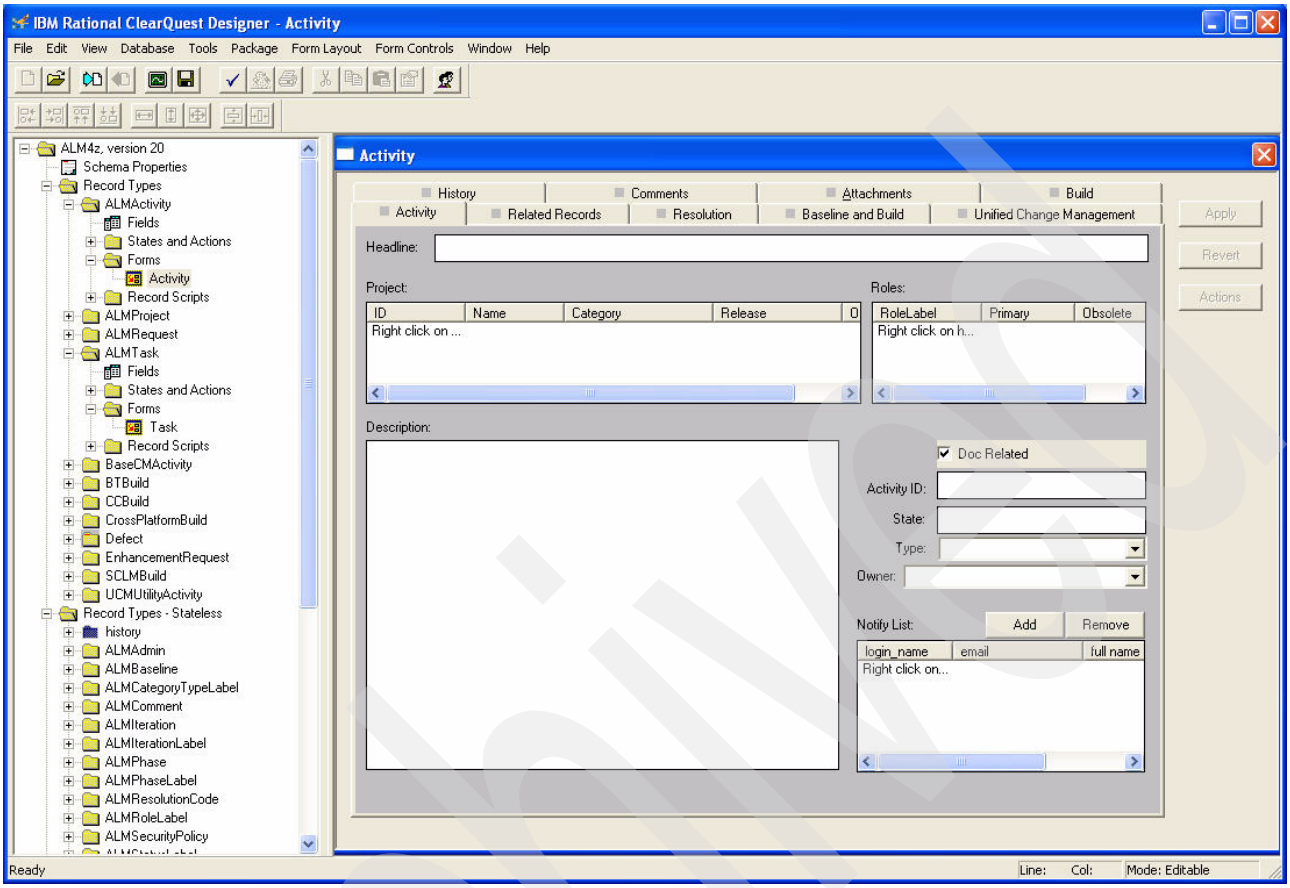


Figure 3-17 The CQALM Schema and Activity Record

We found that it took some effort to successfully adopt and implement a CQALM-based life cycle. You should plan for this, but the resultant benefit in the form of a robust Enterprise SCM environment should not be underestimated. It is well worth the effort. You should review these sources, and decide whether the CQALM schema is appropriate to your organization.

The CQALM Packages for ClearQuest can be downloaded from:

`http://www.ibm.com/services/forms/preLogin.do?lang=en_US&source=swg-ratcq`

There is no charge for the packages, but you need to register in order to download them. The download includes the ALM Packages for ClearQuest, a sample database, and three tutorials for use with the sample.

Regardless of how comprehensive (or how simple) your ClearQuest life cycle is, if you want to incorporate an automated build as part of the life cycle, you can attach a hook to the “complete” action of the BTBuild record, or to the “submit” action of a CrossPlatform Build request, or to whatever action and record type you identify for managing builds.

In one implementation, we created a ClearQuest hook code that uses the ClearQuest Perl API (CQPERL) to retrieve the changeset from ClearCase. It then iterates through the returned list, and passes the version extended path name to Host-Build.pl. The fragment shown in Example 3-2 illustrates this implementation. Note that this hook can also call Remote Build directly.

Example 3-2 ClearQuest hook code

```
#-----------------------------------------------------
# Get a ClearQuest Query Def object
#    We will query on All UCM Activities
#-----------------------------------------------------
my ($QueryDef) = $CQsession->BuildQuery("All_UCM_Activities");
#-------------------------------------------------------------
# Create the ClearQuest result set and execute
#-------------------------------------------------------------
my ($ResultSet) = $CQsession->BuildResultSet($QueryDef);
$ResultSet->Execute();
#-------------------------------------------------------------------
# loop through the CQ Query results.  Use an initial MoveNext to
#    get first record
#-------------------------------------------------------------------
$xlix = 1;
$records = 0;
$status = $ResultSet->MoveNext();
while ( $status == $AD_SUCCESS ) {
$records++;
$xlix++;

#-------------------------------------------------------------------
# Get the field values from the returned query result
#-------------------------------------------------------------------
$id = $ResultSet->GetColumnValue(1);
$state = $ResultSet->GetColumnValue(2);
$headline = $ResultSet->GetColumnValue(3);
$owner = $ResultSet->GetColumnValue(4);
write();
}
#-------------------------------------------------------------------
# Get the CC activity ID from the CQ "ucm_vob_object" field
#-------------------------------------------------------------------
$activity_id = $ResultSet->GetColumnValue(5);
#-------------------------------------------------------------------
# If the activity ID from CQ is null, there is no
#    change set info
#-------------------------------------------------------------------
if (! $activity_id ) {
print "     No ClearCase Activity info yet\n";
}
} else {
#-------------------------------------------------------------------
# If the CC activity ID is not null, then we need to use CAL
#    to extract the changeset info from the CC activity object
#-------------------------------------------------------------------
my ($CCApp) = Win32::OLE->new ("ClearCase.Application") or
```

Archived

```
            die "Can't create ClearCase application object via call to
Win32::OLE->new(): $!";
print "Processing Activity info\n" if $DEBUG;
#-------------------------------------------------------------
# Get an activity object from CAL
#-------------------------------------------------------------
      $myactivity = $CCApp->Activity($activity_id);
if (! $myactivity ) {
print "Can not resolve activity info in ClearCase\n";} else {
      $view = $myactivity->NameResolverView;
#-------------------------------------------------------------
# Get the activity's change set, which is a CCVersions collection.
#   Use the activity's "nameresolver view" for name resolution.
#-------------------------------------------------------------
      $ChangeSet = $myactivity->ChangeSet($view, "False");
      $CS_Entries = $ChangeSet->Count;
#-------------------------------------------------------------
# Loop through the CCVersions collection, collecting the names of
#     the versions for printing.
#-------------------------------------------------------------
$CS_Index = 1;
      while ($CS_Index <= $CS_Entries) {
         $Version = $ChangeSet->Item($CS_Index);
         $VersionPN = $Version->ExtendedPath;
            if ("$CS_Index" eq "1") {
               $cs_list = $VersionPN;
            } else {
               $cs_list = $cs_list . "\n" . $VersionPN;
            }
            $CS_Index++;
          }
      $status = $ResultSet->MoveNext();
```

The variable $VersionPN is what passed to Host-Buld.pl, or to Remote Build. This should be the ClearCase version-extended path name of the element to be built.

There is a developerWorks article that can guide you through the use of the ClearQuest PERL API, found at:

http://www.ibm.com/developerworks/rational/library/4702.html?S_TACT=105AGX15&S_CMP=EDU

3.3 Incorporating Build Forge with Remote Build

Build Forge automates many of the key steps of building an enterprise application. If Build Forge is in your plans, you are probably considering how to use Build Forge to build the mainframe components of your applications. We used Build Forge as part of this project.

As with ClearQuest, your first decision is whether Build Forge fits into your strategy. You can implement a robust cross-platform build process without Build Forge, but like any tool, you need to decide whether a home-grown solution can deliver the benefits less expensively. We found Build Forge was very easy to implement, with a few caveats.

Archived

If you are not already familiar with Build Forge, here is a link to a brief whitepaper that will get you started:

ftp://ftp.software.ibm.com/software/rational/web/whitepapers/RAW14005-USEN-00.pdf

3.3.1 Build Forge

There are two ways to build mainframe assets with Build Forge. You can use the Build Forge z/OS agent or you can use Build Forge to call ClearCase Remote Build.

Using the z/OS Build Forge agent

The advantage of the z/OS agent is that Build Forge manages the creation and submission of the z/OS builds.

A whitepaper describing how to configure and use the Build Forge z/OS agent can be found at:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/rational/pdf/BuildForge-SystemzUseCases.pdf

The z/OS agent runs in UNIX System Services, and you use REXX scripts in the HFS to run your build jobs.

Figure 3-18 is an example of a Build Forge step that runs a REXX script.

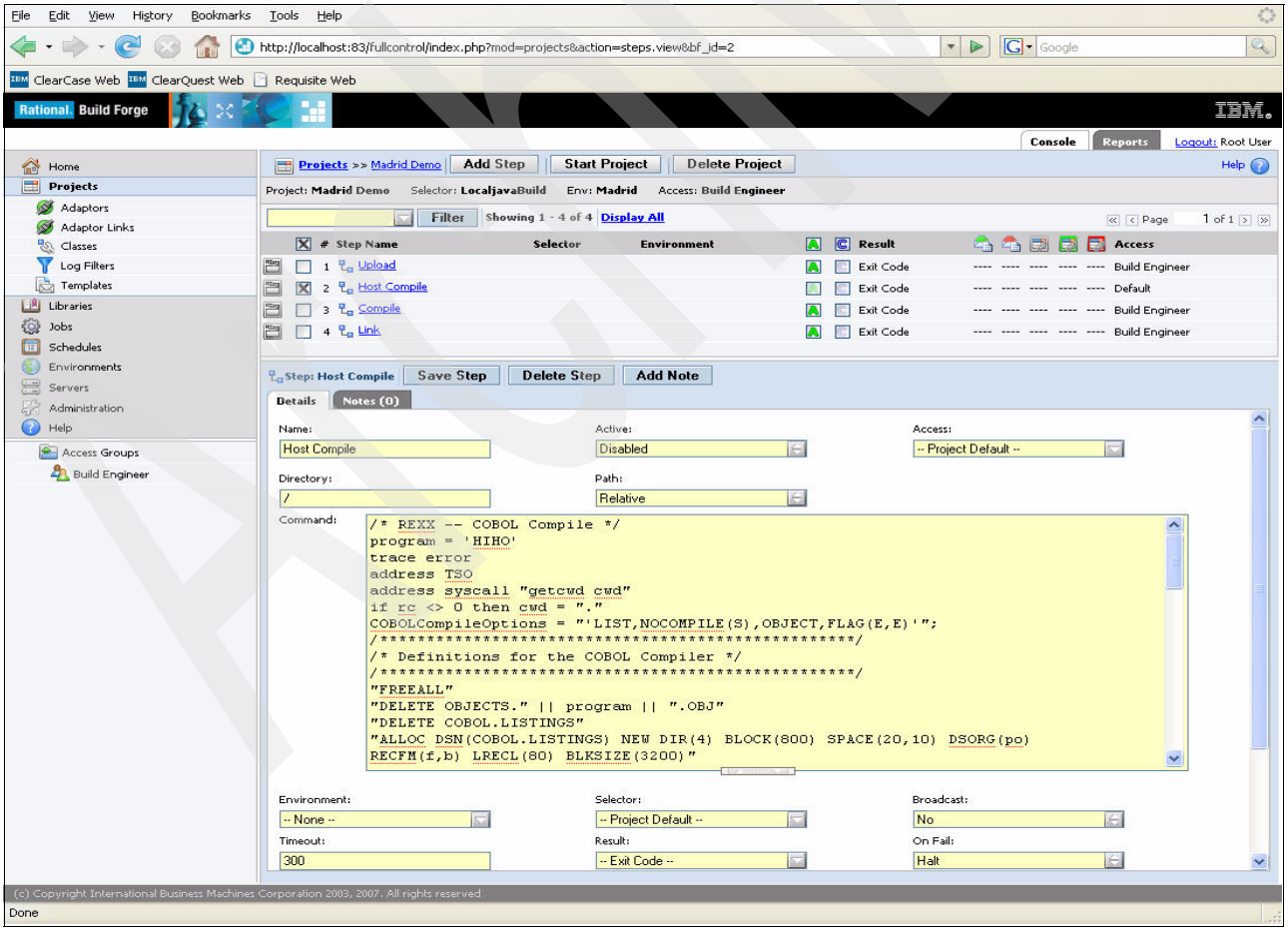


Figure 3-18 Using the Build Forge z/OS agent to run a host build from UNIX System Services

The agent can only work with artifacts that are already in a PDS or in an HFS. There is no means of transporting artifacts between the host and distributed environments. If you want to use the z/OS agent to build ClearCase managed artifacts, you need to transport them there by other means. One way you can do this is to create an “UPLOAD” step in your Build Forge project, as shown in Figure 3-19.

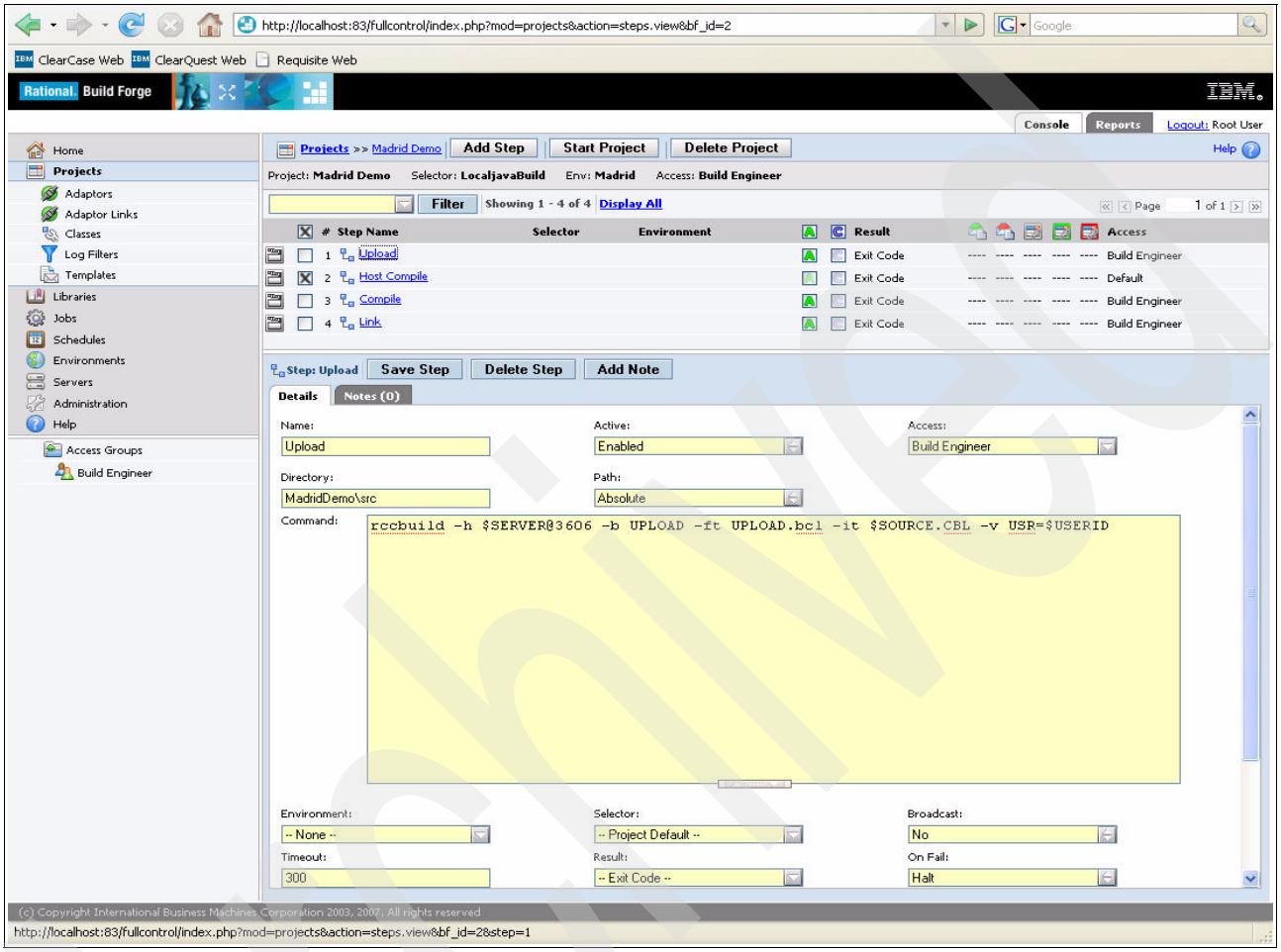


Figure 3-19 A sample Build Forge step uploading elements with ClearCase Remote Build

The step shown in Figure 3-19 calls a simple BCL script that ships the elements to the designated PDS on the host:

```
//UPLOAD EXEC PGM=IEFBR14,REGION=4M
//CBLSRC DD DSN=&USR..MADRDZ.CBL,DISP=SHR,RCCEXT=CBL
```

The next step is run on the host and contains the REXX steps to submit the compile job. Note that in using the z/OS agent, the compile output and listings remain on the host by default. You would have to create an extra step to return them to the ClearCase repository.

ClearCase build auditing features cannot capture builds run with the Build Forge z Agent.

Building host applications with a Build Forge local agent

In our sample implementation, we decided to try an alternative approach. Instead of using the Build Forge z agent to submit host builds, we instead configured a local (desktop) Build Forge agent to use the ClearCase Remote Build feature to run the build job. We did this just by creating a different compile step, which, like the upload step, calls ClearCase Remote Build to perform the compile, as shown in Figure 3-20.

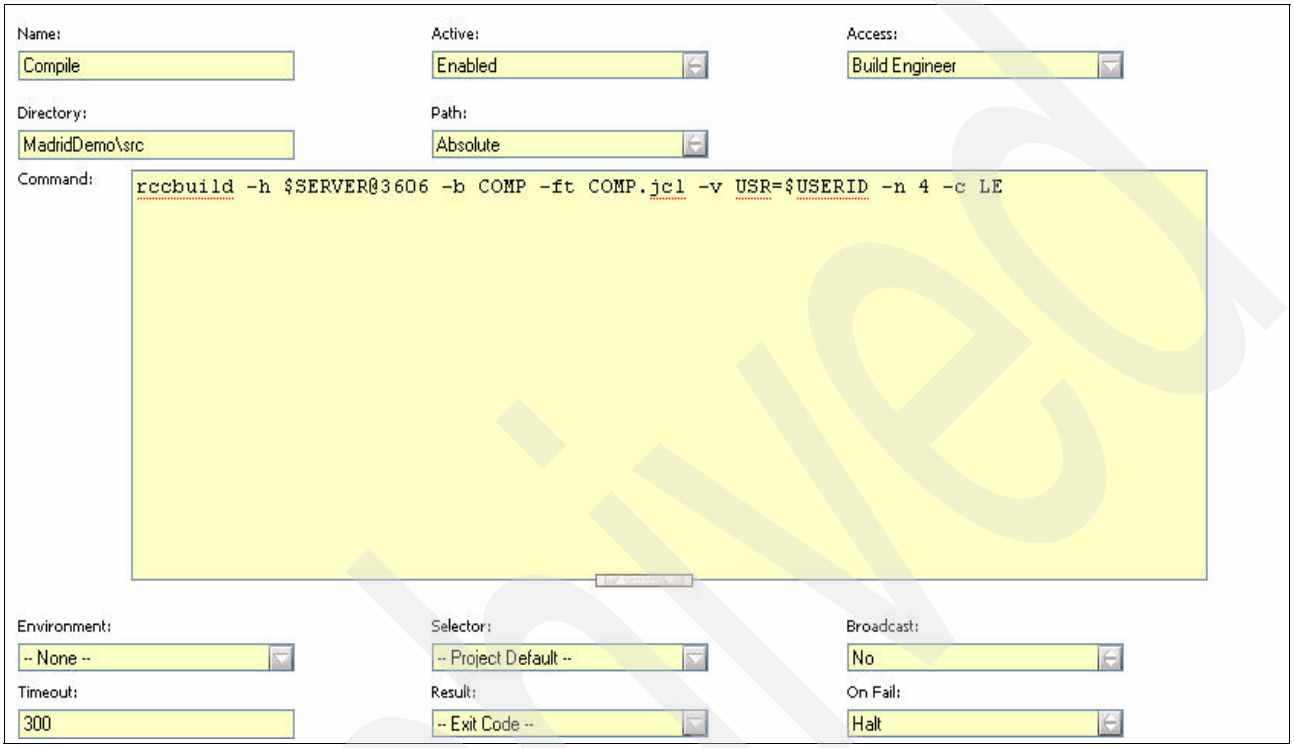


Figure 3-20 Build Forge Step Invoking ClearCase Remote Build

If you choose to work this way, you do not need a separate upload step, since Remote Build manages the transport of the files between the host and the distributed side.

3.4 Enterprise Application Lifecycle Management with Rational Developer for System z

In the preceding sections, we showed you how to use combinations of ClearCase, ClearQuest, and Build Forge to build a mixed workload application. In this section, we show you how we configured Rational Developer for system z to manage these steps as part of an Application Lifecycle Management.

Rational Developer for System z (in its original incarnation of WebSphere Studio Enterprise Developer) was created to equip the mainframe developer with a broader set of tools, a modern GUI, and the ability to offload some development tasks, such as syntax checks to a less expensive environment.

When we started this book, we did so with the intent of defining the solution, or set of solutions necessary to implement an Enterprise SCM. Our original assumption was that Rational ClearQuest would act as the ALMz “cockpit” or platform.

Archived

After some effort, it became clear that although it is entirely possible to construct a robust Application Lifecycle using ClearCase, ClearQuest, and Build Forge, what was needed was a means to more closely couple the host and distributed environments So while we started the expectation of using Rational Developer for System z as the IDE, we found that although not mandatory, using Rational Developer for System z as the “control center” enabled us to build a more flexible and robust Application Lifecycle Management solution with less effort. There is substantial documentation, training, and tutorials to get you up to speed with Rational Developer for system z. This section is not intended to replicate that. Here we just cover those features of Rational Developer for System z that are relevant to Enterprise Software Configuration Management.

You will need some familiarity with Rational Developer for System z to make effective use of this section. If you need to familiarize yourself with Rational Developer for System z, we suggest you start with the information found at this link:

http://www.ibm.com/developerworks/rational/products/rdz/

3.4.1 Relational Developer for System z: a very brief overview

Rational Developer for System z is an Eclipse-based IDE for developing enterprise applications. It is a super-set of Rational Application Developer, and offers all of the same benefits, and additional features to support the development, debugging, and deploying of COBOL, PL/I, C, and C++ Web applications deployed to distributed and mainframe host environments. In addition to being a fully featured Java development environment, Rational Developer for System z includes a broad set of utilities to simplify and automate many of the tasks of an ISPF developer, and enables the offloading of some non-production tasks that are traditionally host-based, such as syntax checks, onto your desktop.

With Rational Developer for System z, you can browse a host PDS in the same way you can browse a local desktop file system. You can drag and drop files between the two environments, and you can configure Rational Developer for System z to automatically synchronize the contents of your local and remote file systems.

For this book, we focus only on those features most directly relevant to Enterprise SCM, although there is obviously much more you can do with it.

We focused on two primary activities: Working with ClearCase-managed artifacts, and configuring and initiating cross-platform builds.

3.4.2 Working with ClearCase managed artifacts

While we mostly used a z/OS project’s perspective in producing this book, you can use any perspective you wish. We found that the default window configuration in the z/OS projects perspective was the best suited to ESCM activities. In particular, with the z/OS projects window, you can work directly with ClearCase-managed artifacts by mapping your project directly to your ClearCase view. More on this shortly.

You can access ClearCase controlled artifacts in two ways: Through traditional ClearCase views (dynamic and snapshot) or through ClearCase Eclipse views.

You can use any kind of ClearCase view with Rational Developer for System z. You determine this by enabling either the ClearCase SCM adaptor or the Eclipse plug-in. Figure 3-21 shows the steps for doing this.

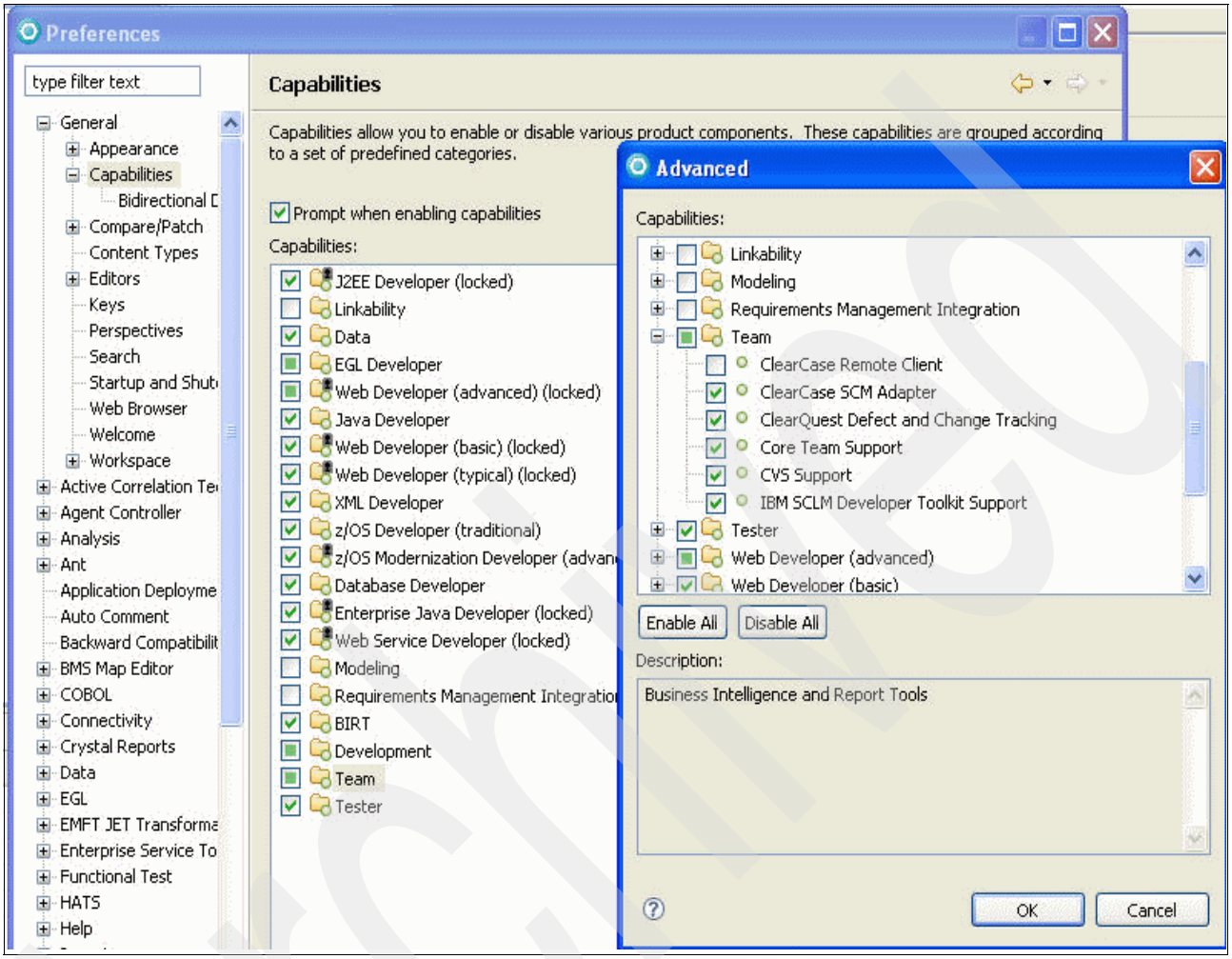


Figure 3-21 Activating the ClearCase SCM adapter

The SCM adapter is included with Rational Developer for System z. It assumes that local ClearCase services are available, so you need to have ClearCase locally installed on your machine to use it. In Figure 3-21, we have enabled the SCM adaptor by selecting **Window** → **General** → **Preferences** → **Capabilities**, and then selecting **Team**. Clicking the **Advanced** button in the lower right corner of the Preferences window will present the Advanced box. Selecting **Team** in the Advanced box will present you with the choice of activating the ClearCase Remote Client (the Eclipse plug-in) or the ClearCase SCM adapter.

If you are already running with the ClearCase fat client, using the SCM adaptor gives you the benefit of being able to run with ClearCase dynamic views inside of Rational Developer for System z. If you do not use the ClearCase fat client, you will use the ClearCase Remote Client (CCRC) instead.

The most significant difference between using the SCM adaptor and the CCRC is whether you load and synchronize your Web views (which is unnecessary with dynamic views). The rest of this section is equally applicable to either approach.

Note: You must be running Rational Developer for System z V7.1.1.2 and ClearCase V7.01 or later to use the SCM adaptor. If you are running earlier versions of either tool, you should use the Eclipse client.

3.4.3 Connecting to your repository

The usage model for Eclipse is based on the idea of a local workspace. When a developer starts work on an activity, they populate the workspace by copying (or checking out) a set of controlled artifacts from an external SCM repository, such as SubVersion or CVS, and then copying them back in (or checking them in).

One of the original strong selling points of ClearCase is that with ClearCase, the tedious task of manually copying artifacts back and forth between repository and work area is no longer necessary. ClearCase instead presents the developer with a “virtual” file system that projects a set of versions of managed artifacts as a native (Windows or Linux/UNIX) file system that is entirely private to the developer. With many other SCM systems, the rationale for a sandbox or work area is to enable developers to work in parallel and in isolation to avoid overwriting one another’s changes. With ClearCase, developers isolate themselves with these private file system (these are referred to as “views”).

As mentioned above, this approach is the same whether you use local views and the ClearCase SCM Adapter, or if you use the CCRC and an Eclipse view.

To work with Web or Eclipse views, use the ClearCase Eclipse plug-in. If you enable the Eclipse plug-in, you can work with your assets with the ClearCase Eclipse perspective, in addition to using the File menu in the z/OS projects view.

When a developer begins to work on an activity, they usually start by creating a new ClearCase view. The view will appear as a private workspaces to the developer, without requiring the developer to physically copy the artifacts to a different location.

What this means with Rational Developer for System z (or any Eclipse-based solution) is that when you create or import a project, you do not need to physically copy the artifacts into your workspace; in fact, you should not. Instead, just map your workspace to your view drive or directory, as shown in Figure 3-22 on page 103. If you do copy the artifacts into the workspace, you circumvent the value of ClearCase, because if you copy the artifacts out of ClearCase, you are creating non-managed versions of them that at some point need to be reconciled with the repository version.

Archive

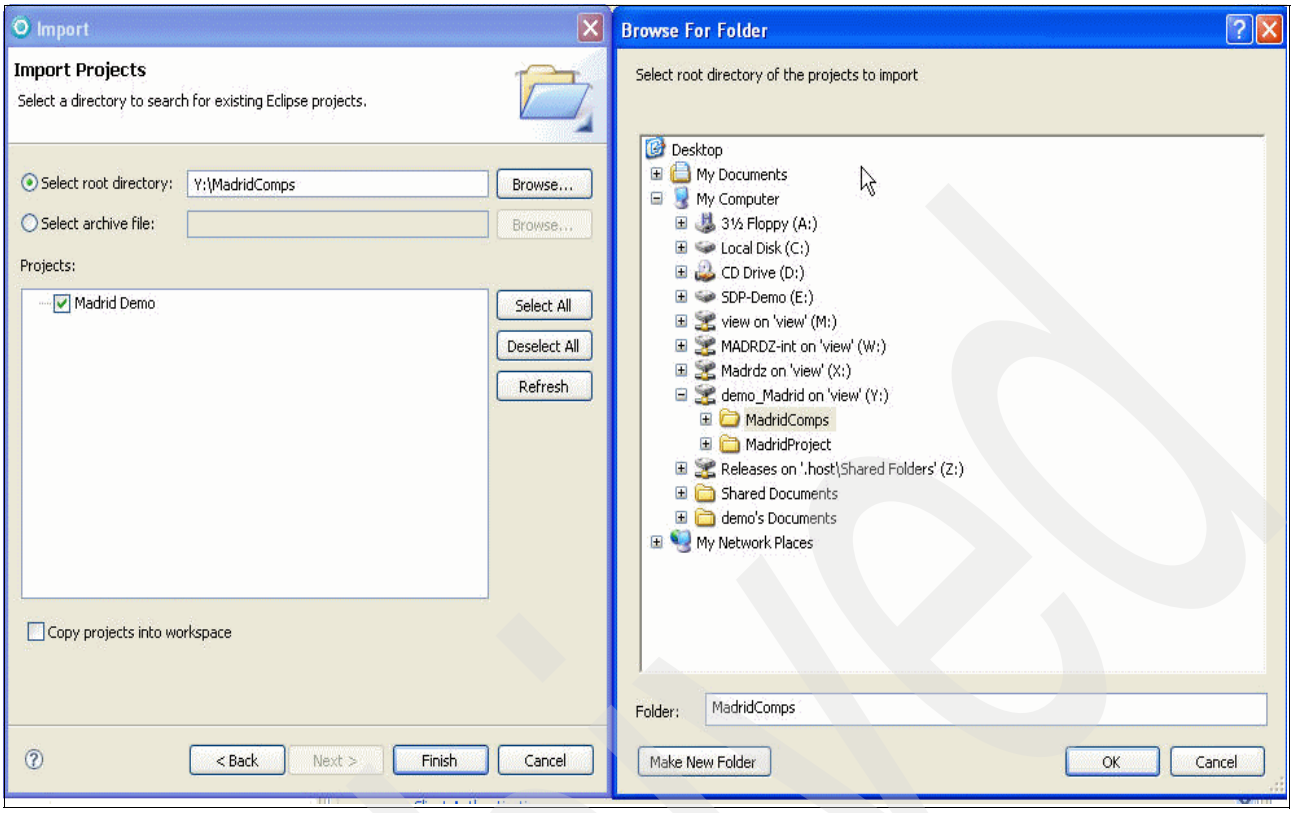


Figure 3-22 Importing a ClearCase-controlled Rational Developer for System z project

3.4.4 Working with controlled elements

Note that there is no ClearCase perspective with the SCM adaptor. ClearCase operations are accessible by right-clicking the element, or by customizing your perspective to add ClearCase buttons to your toolbar, as shown in Figure 3-23.



Figure 3-23 The ClearCase Eclipse toolbar

In Eclipse projects, certain files and directories need to be writable. You will not be able to work with the project until you at the very least make all of the directories writable. The project meta data files (.project and so on) need to be writable as well. You can do this either by making them controlled elements and keeping them checked out, or by creating them as view-private files.

Once you have imported your project as above, the fact that you are working with ClearCase managed files is, or should be, transparent. Aside from having to check out elements to modify them, you should be able to build, test, and deploy in the same manner as you can with a native file system. What will be different is that it will no longer be necessary to copy assets in and out of a source repository to work with them. The only time it should be necessary to physically copy files around is when you deploy them to test or production environments.

File level ClearCase operations can be found by right-clicking the element and selecting **Team**, as shown in Figure 3-24.

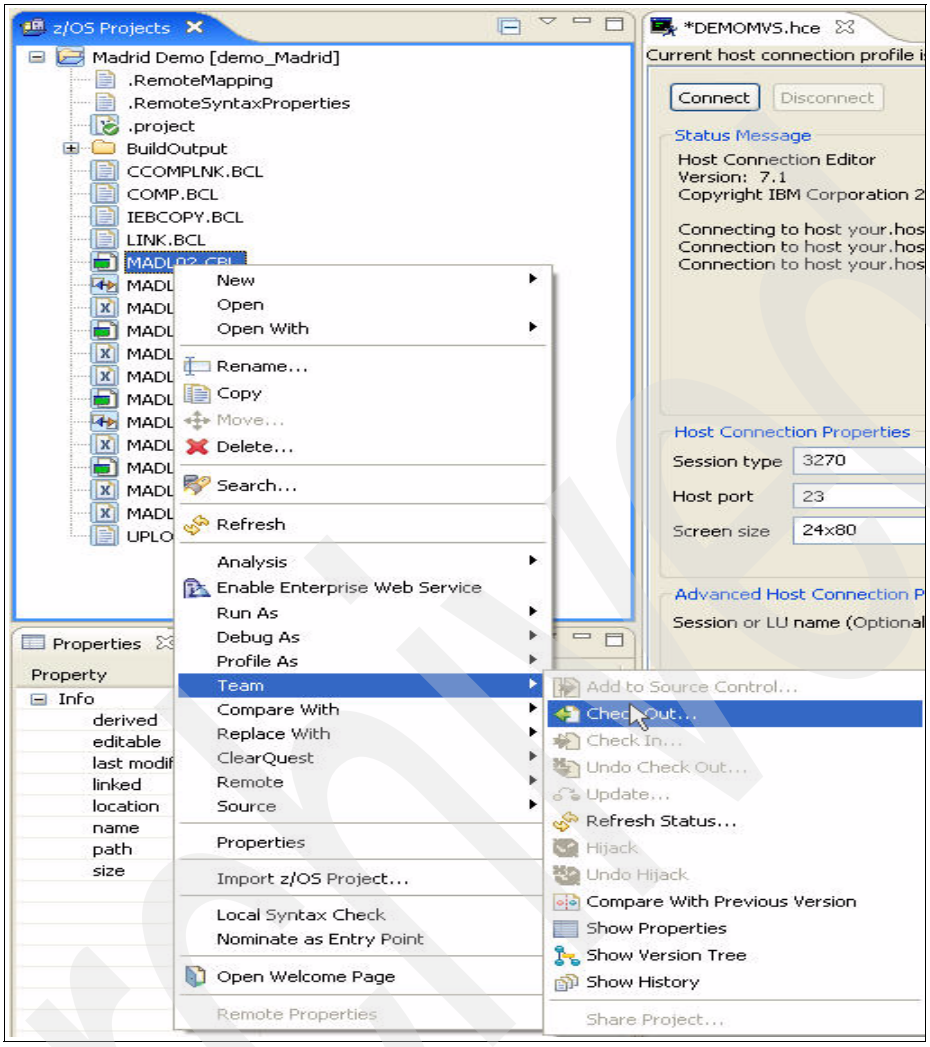


Figure 3-24 Checking out a ClearCase-managed element from the Rational Developer for System z z/OS Projects Explorer

3.4.5 Build, migrations, and promotions

Build Forge and ClearQuest all have fully functional Eclipse perspectives that enable you to use them from within Rational Developer for System z. These are well described in the documentation for the respective products.

Rational Developer for System z also includes its own build manager. As with Build Forge and the other solutions we have discussed in this section, the same calls to Remote Build or a build engine script can be built into an Rational Developer for System z build step, as shown in Figure 3-25 on page 105 and Figure 3-26 on page 105.

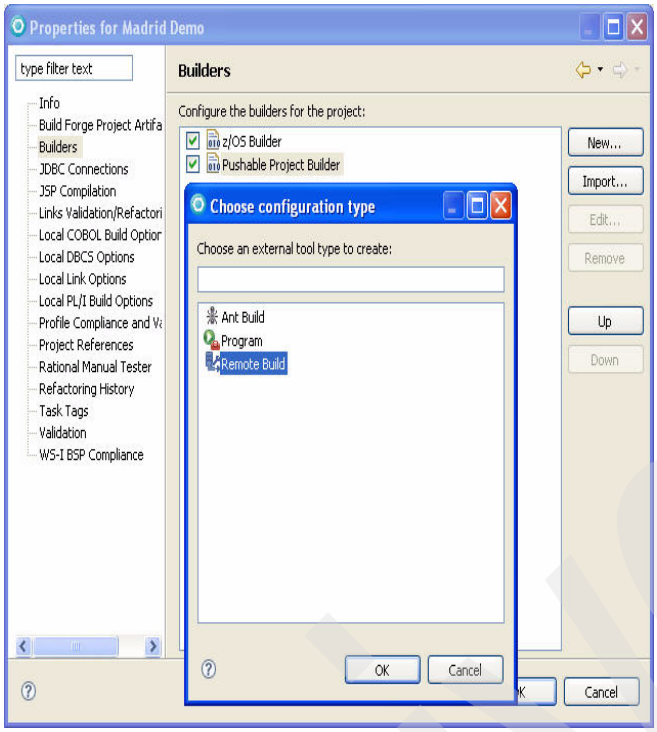


Figure 3-25 Configuring an Rational Developer for System z Builder for ClearCase Remote Build

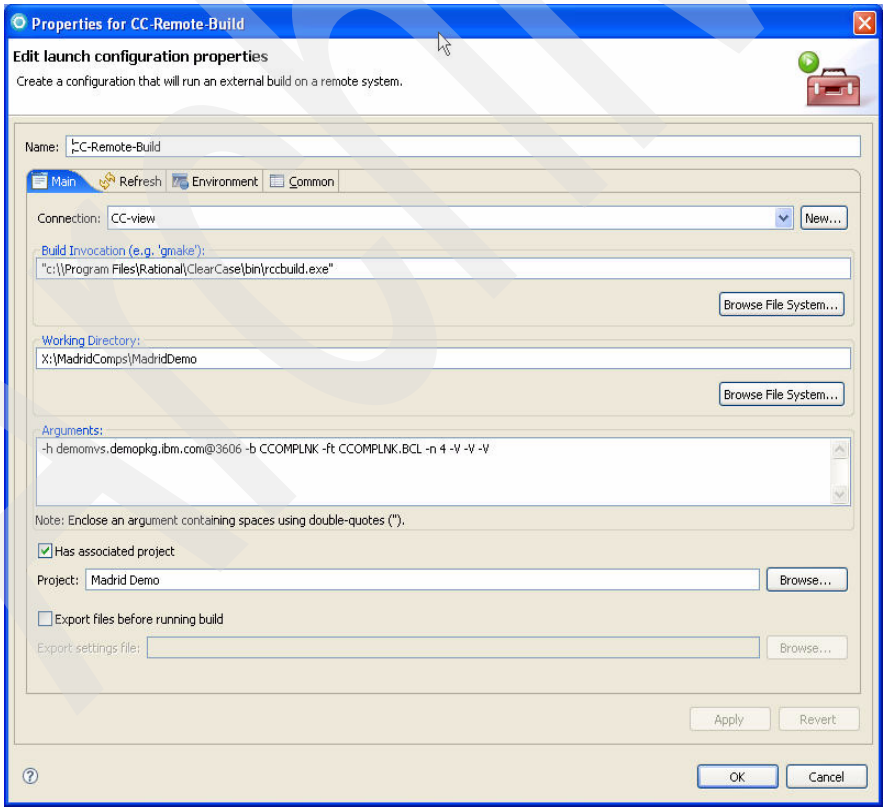


Figure 3-26 Creating an Rational Developer for System z Build Launcher to run ClearCase Remote Build

You can create additional launchers for each of your different build steps and options. Once you have created your launchers, you can build your project by either selecting **Project** → **Rebuild Project** or by right-clicking the project itself.

When you run the build. Rational Developer for System z will display the results in a remote shell window.

As part of the proof of technology for this book, we also configured Rational Developer for System z builders to invoke ClearCase Remote Build directly and to invoke the Host-build.pl script, as shown in Figure 3-27.

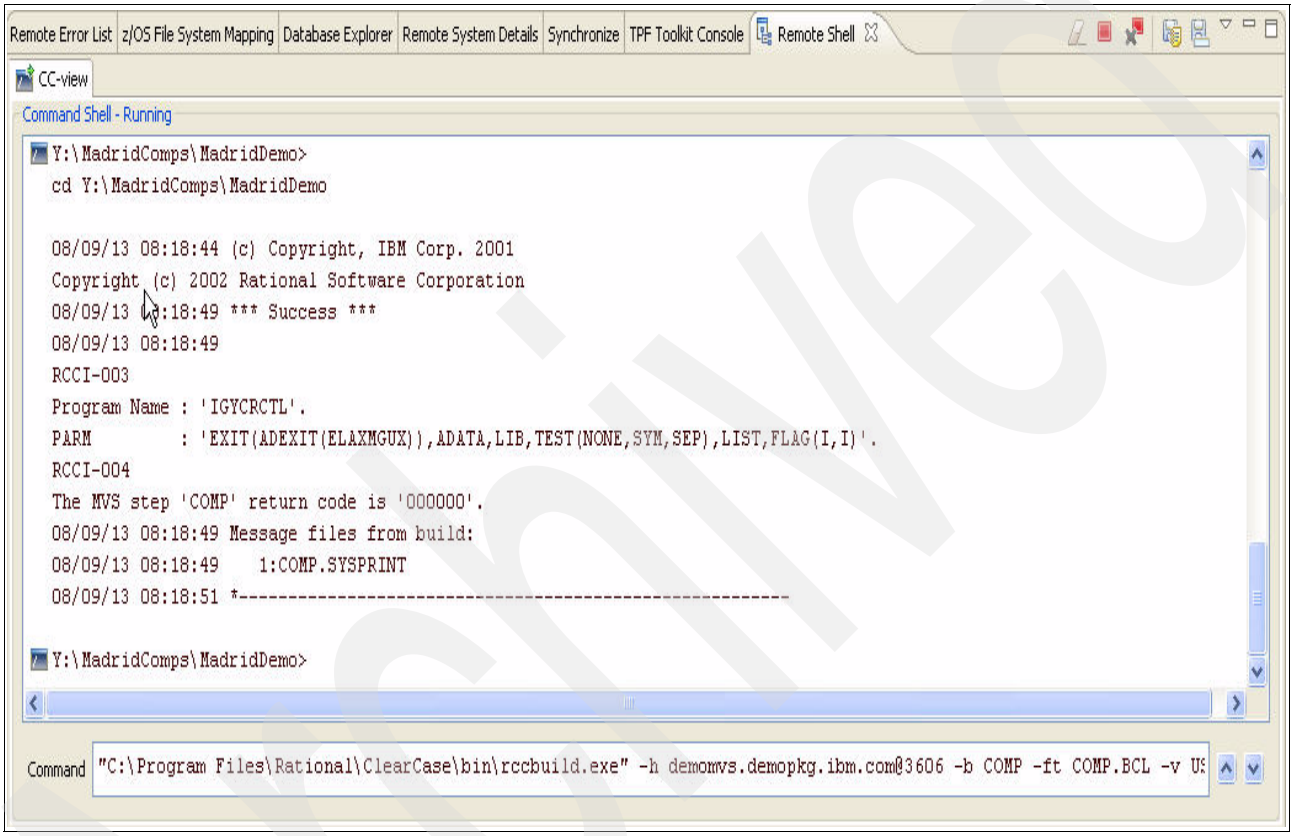


Figure 3-27 Remote Build Output from Rational Developer for System z Build Initiator

Regardless of which of the solutions actually invokes the build process, it was our experience that at some point in the process, there is a dependency on either Build Forge or the ClearCase Remote Build feature. The Remote Build feature had the advantage of being invokable by both GUI and command-driven environments, and as such could more easily be invoked as part of an audited ClearCase build.

We found that you can construct a reasonable synchronized build solution just with ClearCase and Rational Developer for System z. Build Forge adds a layer of robustness and reporting that greatly simplifies this process, especially if you are confronted with audit or compliance issues.

As we developed the project for this book, we found the following deployment model seemed to fit best:

- Rational Developer for System z and ClearCase for developers
- Build Forge for lead developers and production control
- ClearQuest for project leads and project managers

As we have seen in this section, you can use a variety of build paths through these solutions. Figure 3-28 illustrates the alternatives. What you actually deploy will depend on your organizational style, policies, and solutions available.

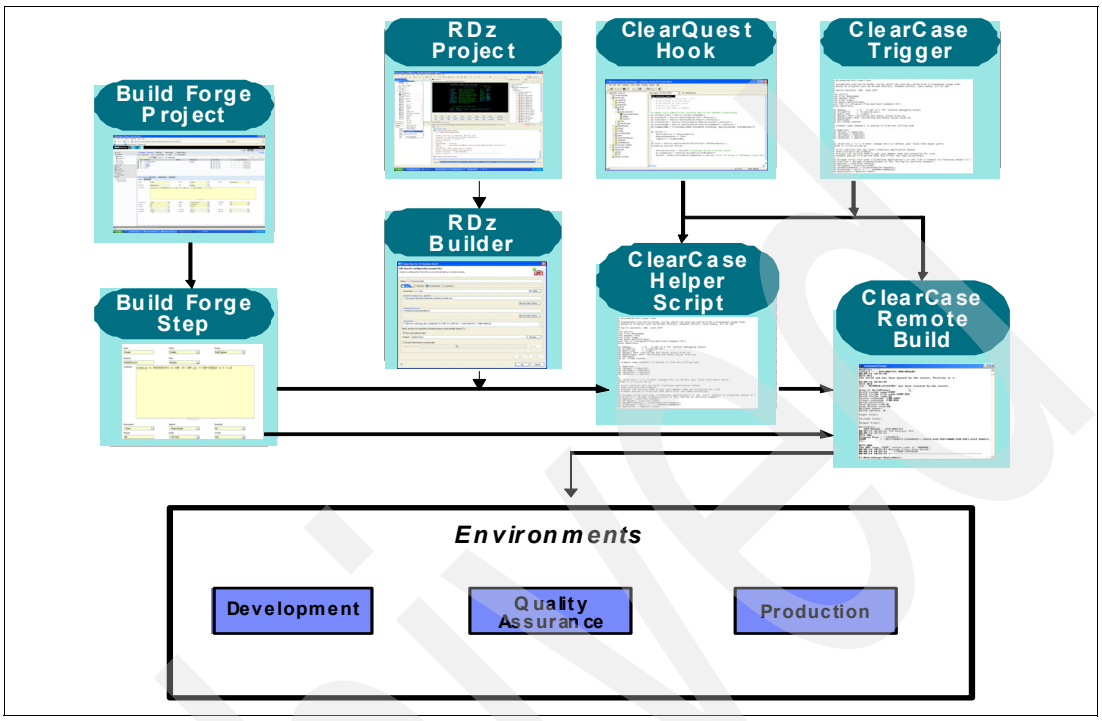


Figure 3-28 ESCM build paths with Rational solutions

3.5 Using WebSphere Studio Asset Analyzer for impact analysis

In this section, we are going to discuss how to use WebSphere Studio Asset Analyzer (WSAA) to help project managers, programmer analysts, application developers, quality assurance testers, and other users to do application dependency analysis and impact analysis as part of the Application Lifecycle Management for System z. It is not the intention of this book to describe how to set up and use WSAA. More information about WSAA can be found at:

http://www.ibm.com/software/awdtools/wsaa

3.5.1 What WebSphere Studio Asset Analyzer (WSAA) is

To start with, WSAA is a DB2 database, which contains just over 100 tables. WSAA scans both mainframe and distributed software assets, then takes the scanned information and loads it into a DB2 repository that resides on the mainframe. These assets include PDS/PDSE data sets on mainframe, and directories on Windows or UNIX. WSAA can also scan assets managed by Software Configuration Management systems (SCMs). For distributed assets, WSAA supports scanning from IBM Rational Clearcase, CVS, and PVCS. For mainframe assets, SCLM is supported.

Archive

Figure 3-29 shows the architecture of WebSphere Studio Asset Analyzer V5.1.

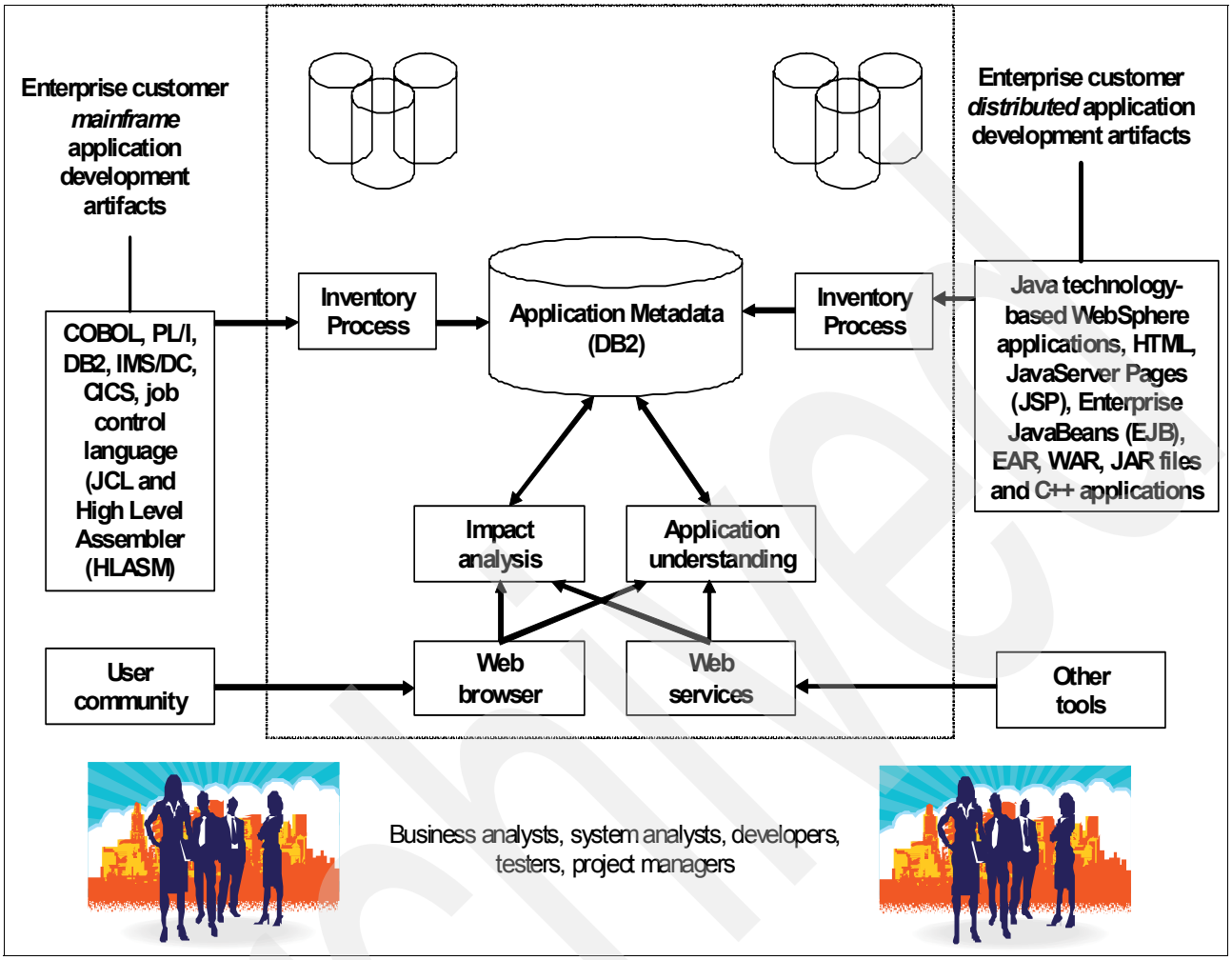


Figure 3-29 WSAA architecture

WSAA is a WebSphere application. Typically, a user accesses the application through a Web browser. WSAA's open architecture also offers user programmatic access through either direct Structured Query Language (SQL) queries or a Web Services application programming interface (API).

WSAA can be used in any phase of the application development process. However, in this chapter, we only discuss how to use WSAA to do application dependency and impact analysis as part of the Application Lifecycle Management for System z.

3.5.2 Identify application dependency

Project managers and program analysts can use WSAA to analyze application dependency and define the scope of the business project before the project is assigned to work on. Using a Web browser, you can see your applications in both a general and detailed way. We use the "Birthday" application as an example. As we described in the earlier sections in this chapter, EPSL is a CICS application. A ClearQuest change request is submitted against this product defect.

Launch the WSAA through a Web browser. Typically, a user needs to log in the WSAA, depending on the setup of the WSAA system.

Figure 3-30 shows the WSAA main interface.

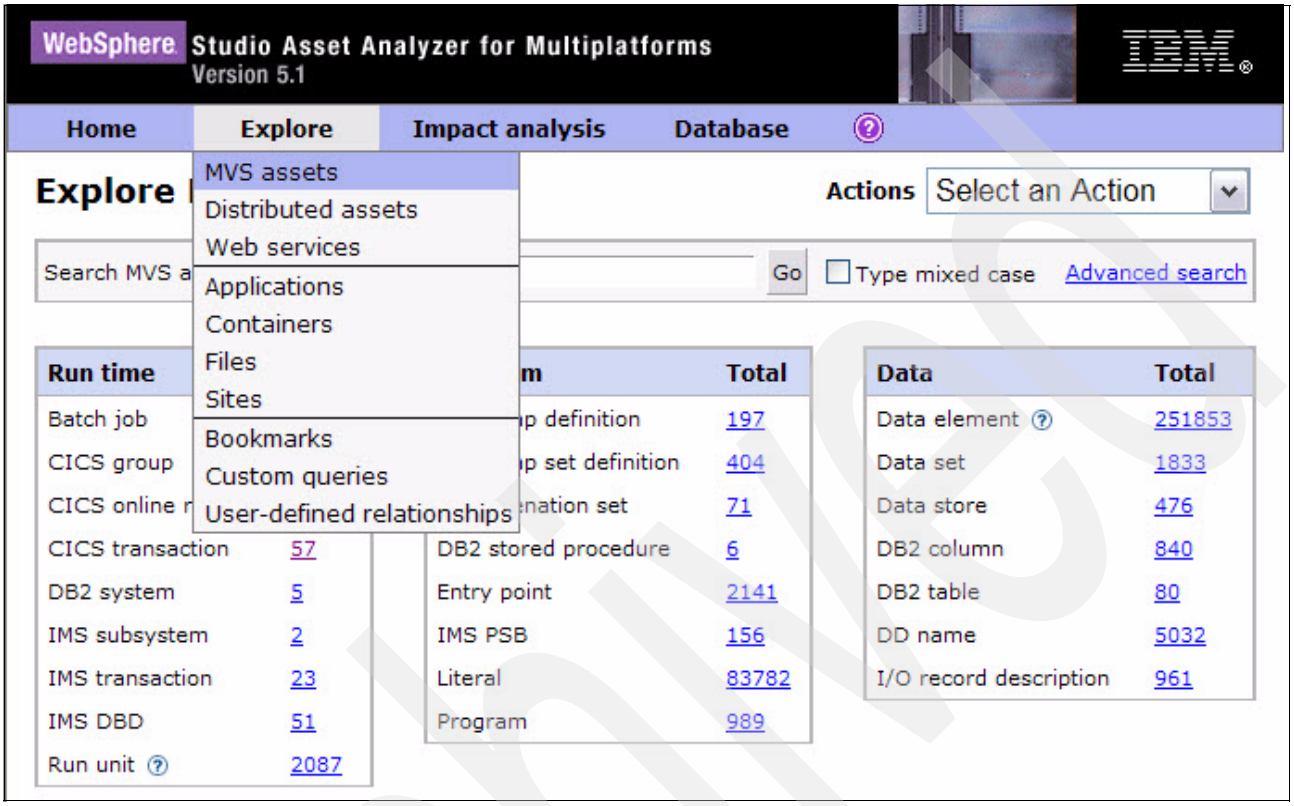


Figure 3-30 WSAA main interface

Archive

Since we know that the sample application “EPSL” is a CICS application, we select **Explore** to list the MVS assets. WSAA allows you to search for a particular application as well. For our example, we select **EPSL** from CICS transaction category. Figure 3-31 shows the CICS transaction details of the EPSL application. Other application information, like online region, CICS group, main entry points, and other resolved entry points, are also displayed. WSAA also provides a variety of utilities for your convenience. You can display all the application dependency information as shown in Figure 3-32 on page 111.

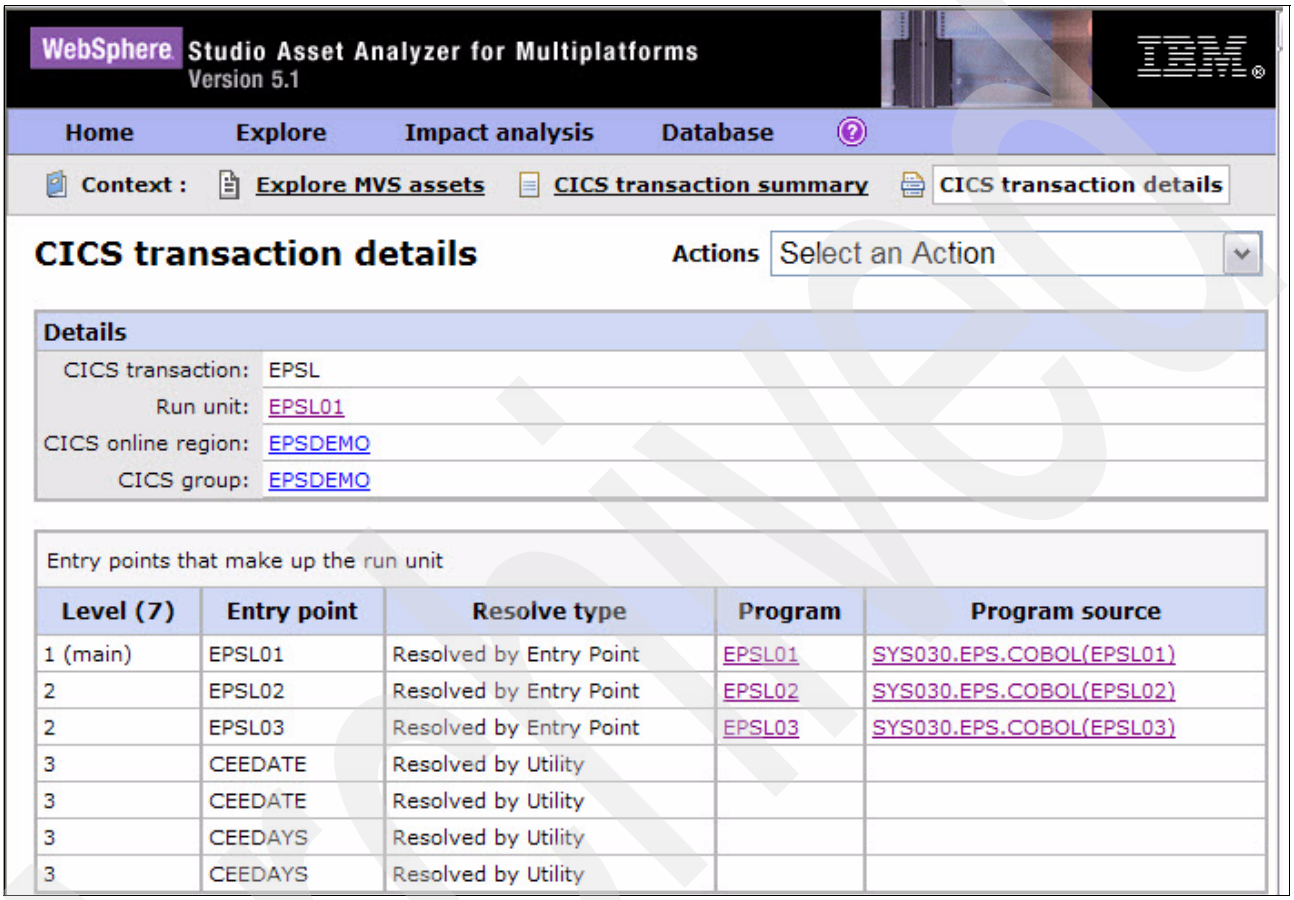


Figure 3-31 WSAA CICS transaction details

Figure 3-32 shows the high level application dependency. WSAA provides easy to use functions for you to explore further with these assets.

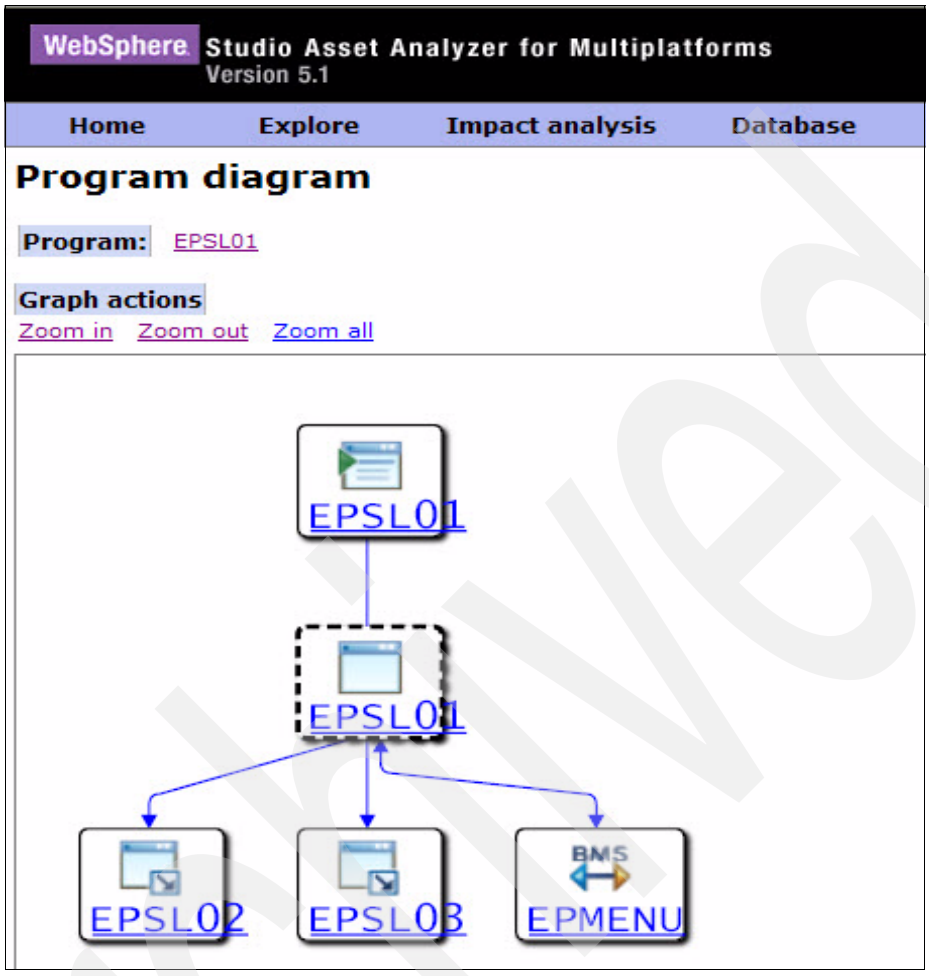


Figure 3-32 WSAA application dependency diagram

Archived

A structure diagram shows you more details information about the particular program. Figure 3-33 is the program structure diagram of EPSL02. The structure diagram lists the procedures in program EPSL02. For our example, we know that the defect is about the calculation of the difference between today and the retirement date. Procedure “CALCULATE-DAY-DIFFERENCE” is the best area a developer should investigate.

All program names and procedure names listed in the diagram are hyperlinks. Clicking a procedure in the program structure diagram will bring up the source code of the selected procedure. Figure 3-34 on page 113 shows the source code of procedure of “A-200-CALCULATE-DAY-DIFFERENCE”.

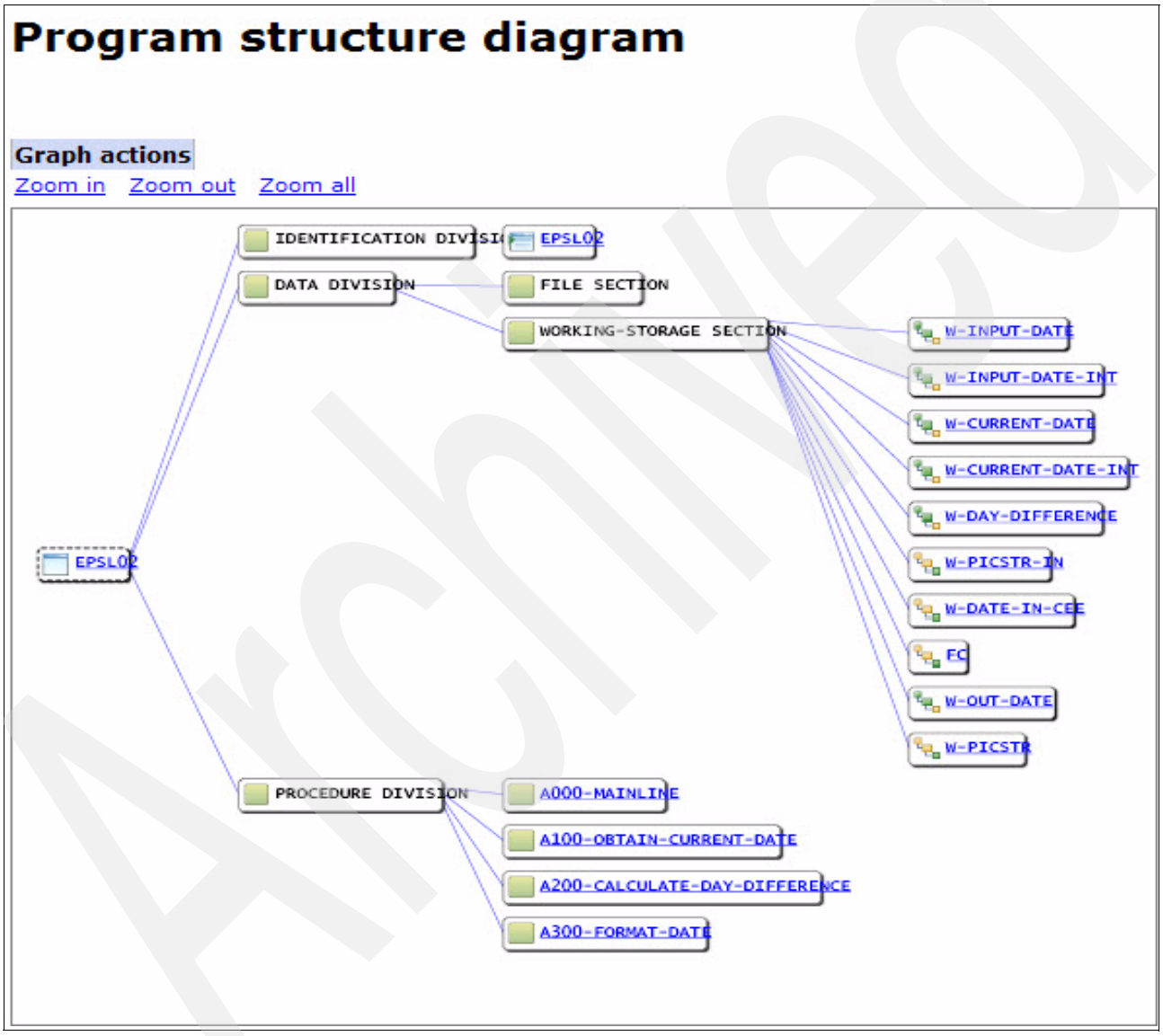


Figure 3-33 WSAA program structure diagram

Archived

```
81.             MOVE FUNCTION CURRENT-DATE(1:8) TO W-CURRENT-DATE
82.             COMPUTE W-CURRENT-DATE-INT = ,
83.                FUNCTION INTEGER-OF-DATE(W-CURRENT-DATE)
84.             .
85.         *
86.          A200-CALCULATE-DAY-DIFFERENCE.
87.             MOVE INPUT-DATE TO W-INPUT-DATE
88.
89.             COMPUTE W-INPUT-DATE-INT = ,
90.                FUNCTION INTEGER-OF-DATE(W-INPUT-DATE)
91.
92.             COMPUTE W-DAY-DIFFERENCE = ,
93.                W-CURRENT-DATE-INT - W-INPUT-DATE-INT
94.
95.             MOVE W-DAY-DIFFERENCE TO DAY-DIFFERENCE
96.             MOVE 0                TO COMM-PROGRAM-RETCODE
```

Figure 3-34 WSAA source code diagram

We just walked through how WSAA can help developers and product managers define the scope of a task with a simple example. The real application is more complicated, and it may take multiple iterations to define the right scope of the task. On the other hand, this is the reason WSAA is useful because WSAA provides an easy to use interface for a user to explore and navigate from a higher level structure diagram to source code.

Archived

3.5.3 Impact analysis

Impact analysis is one of the key features of WSAA. WSAA impact analysis helps developers and users evaluate the effect of a particular code change and reduce the resulting impacts on assets. Figure 3-35 is the interface used to create an impact analysis with WSAA.

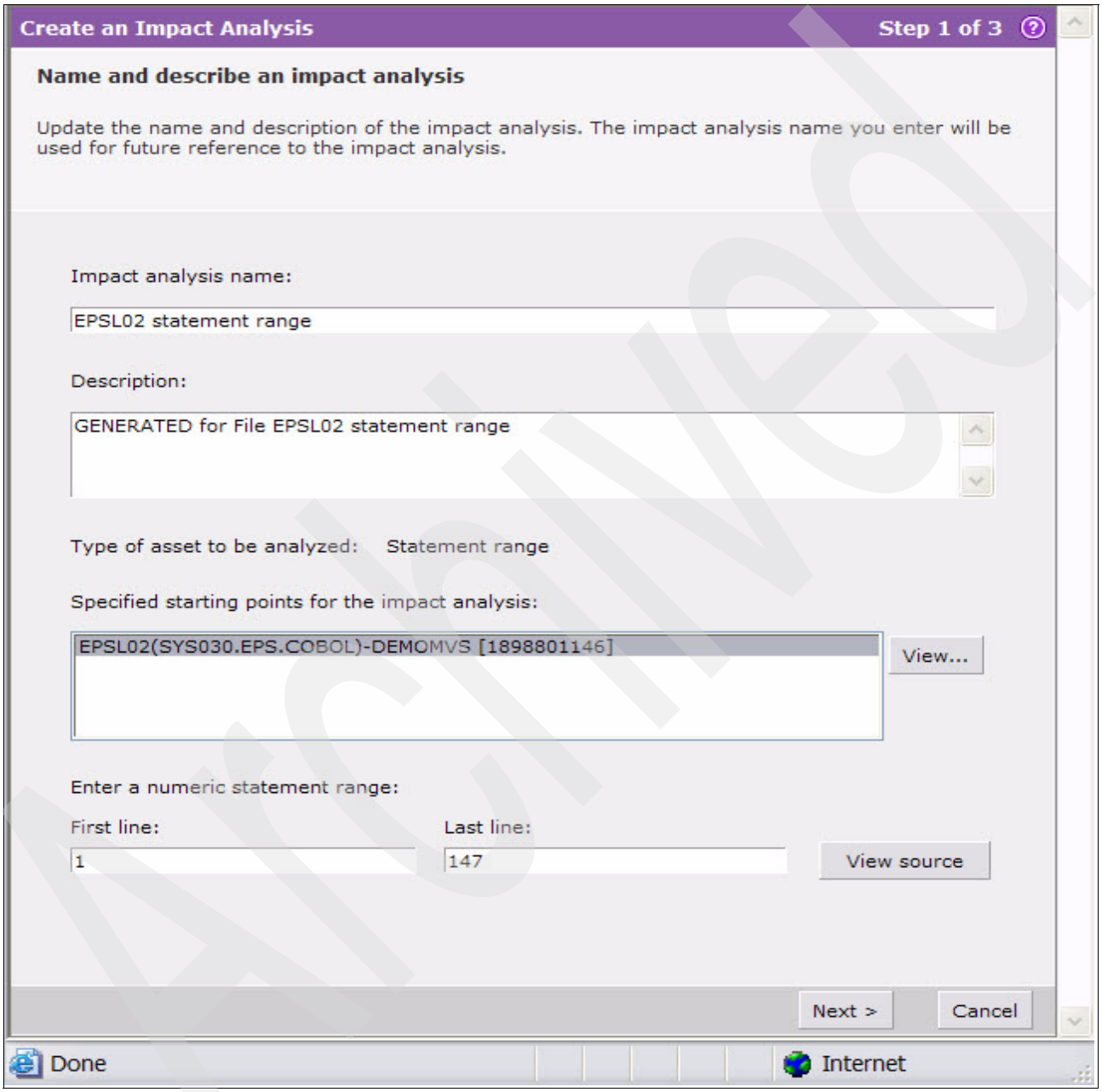


Figure 3-35 WSAA create impact analysis

Figure 3-36 on page 115 is the impact analysis diagram of sample program EPSL02. On the right-hand side of the diagram, WSAA shows the indirect impacts; on the left-hand side of the diagram, WSAA shows the direct impacts. The direct impacts identify the number of data elements that are affected and the number of programs that use them. In the center of the diagram is the data set and data store information. Our simple example does not have any data sets.

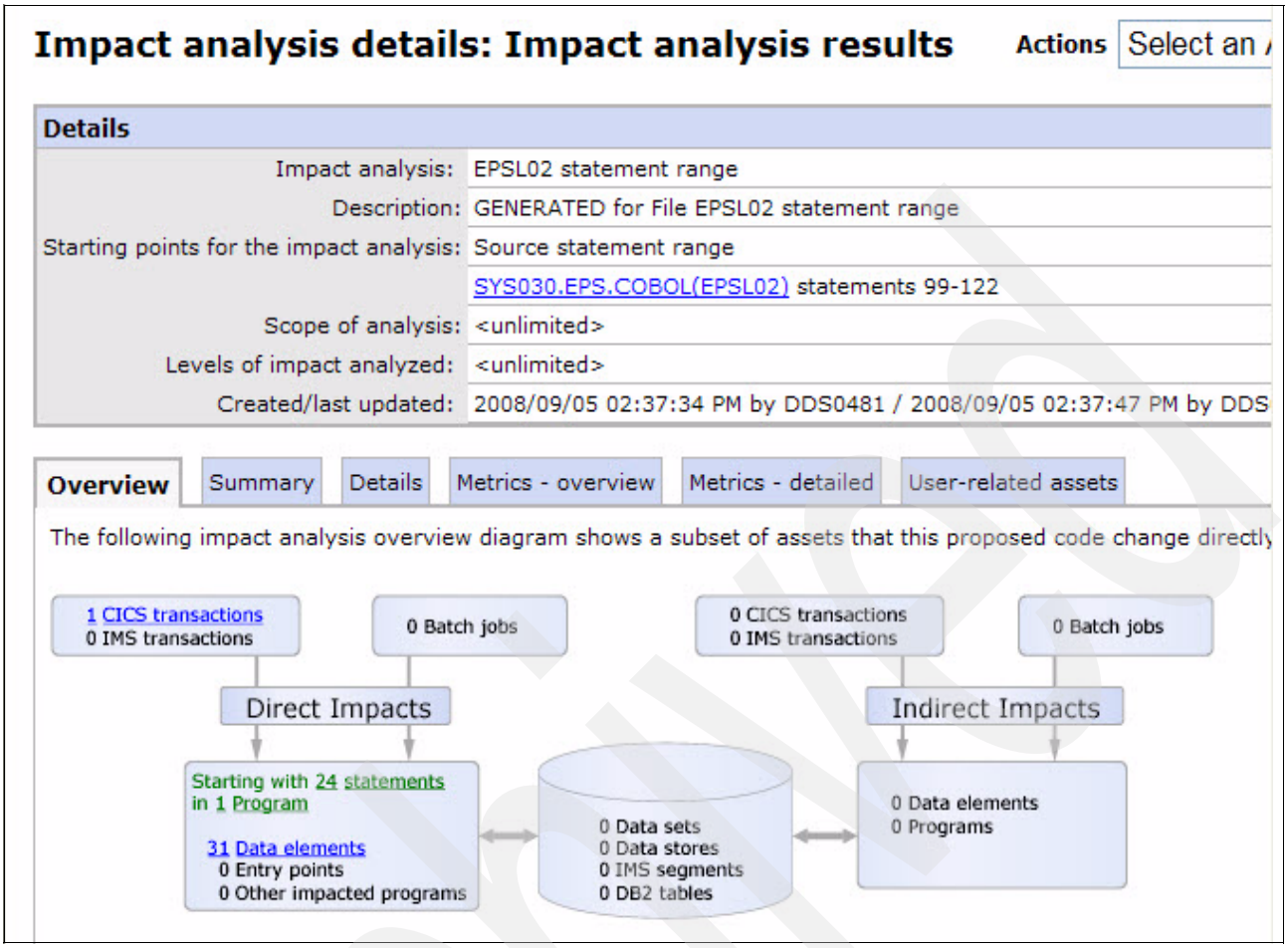


Figure 3-36 WSAA impact analysis diagram

Currently, WSAA impact analysis is not automatically integrated into a managed build process.

There are two ways to integrate WSAA impact analysis into a managed build process:

1. Integrate through WSAA's programmatic access interface

   WSAA's open architecture allows users to access a DB2 database directly through SQL queries so that a user can acquire the impact information and integrate it into the build process. A user can also access the WSAA DB2 database with a Web Services API. Unfortunately, either way requires certain amount of programming.

2. Integrate through Clearcase and ClearQuest

   The goal is to ensure that all work and modifications take place in the context of a defined activity. All the changed programs in the same activity are built together. If a program is affected by the changes in this activity and needs to rebuild, a user can simply add this program into the change set of the activity by checking out and checking in the same program with ClearCase. In our example, to fix the defect, we need to modify EPSL02 only, as the change set of this activity has only that one program. Although there is no code change required for EPSL01, WSAA impact analysis tells us that we may need to rebuild and deploy EPSL01. Checking out the EPSL01 and checking in the same file (without any code change) creates a new version with same content in ClearCase, so EPSL01 is added into the change set of this activity. When the management build process

builds and deploys the activity, both EPSL01 and EPSL02 will be built and deployed automatically.

Archived

Part 2

IBM WebSphere Developer for System z

Part 2 includes topics related to WebSphere Developer for System z Version 7.0

Archived

© Copyright IBM Corp. 2009. All rights reserved.

Archived

4

Traditional and modern application development tools

IBM WebSphere Developer for System z bridges the gap between today's visual programming environment and yesterday's mainframe culture.

WebSphere Developer for System z is a unique Integrated Development Environment. It integrates in a single tool the technologies to develop and maintain traditional mainframe applications and to enable those mainframe applications in Web Services.

IBM WebSphere Developer for System z provides the tools to accelerate the development of a service-oriented architecture (SOA), including services deployable to CICS, IMS, WebSphere Application Server, and DB2 Stored Procedure environments.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

4.1 TSO/ISPF

While it may seem that this new situation will phase out TSO/ISPF, this will not happen in at least the short term. There are many operations that only TSO can perform, especially operations executed by z/OS systems programmers, DB2 database administrators, and others. There are also some installations that depend on ISPF interactive CLISTs to perform z/OS development. However, young developers do not need to learn TSO and ISPF to develop or maintain mainframe programs. Today, we have many efficient ways to interact with the mainframe.

So why not use TSO for only those activities that are a “must” and WebSphere Developer for System z for all other activities?

Many readers probably still remember when the way to interact with a computer was to use a punch card of 80 columns (96 for the IBM System/3). When the first interactive terminals were born, we got a similar question about the phasing out of punch cards. Some developers would not rely on the TSO/ISPF and keep decks of cards handy as a backup just in case.

4.2 The dawn of programming: a look back

Remember when a COBOL program was measured by the number of cards it needed? A programmer would say “My program is big, it fills two boxes.”

Yes, boxes. Each box contained about 2,000 cards, so this example program had 4,000 cards or 4,000 statements; each card usually contained one COBOL statement line. The best practice was to number each card in columns 1 to 6, and add an identification mark on columns 73 to 80. There were even IBM machines (like the IBM 129) that had a way to create special programs to perform automatic numbering without human intervention.

At this time, a developer could take few days or even a week to clean up the compilation of a large program, depending on of the quality of the card puncher, which usually made mistakes with COBOL verbs and names. The process to create a program was:

1. The programmer filled paper templates with the COBOL statements.
2. The program templates were sent to the key puncher (sometimes for small programs or small modifications, the programmer punched his own cards).
3. The deck of cards was sent for compilation.
4. The programmer received the error compile listing report. Many of those errors were due to bad punch quality and misspelled COBOL words.
5. The errors were either fixed by the programmer or sent again to the key puncher until the compile listing was clean.
6. After a clean compile and link, the first tests were run. (At that time. most of the programs were batch and printed reports.)
7. For debugging, the programmer usually used simple COBOL statements like DISPLAY. IF statements were also heavily used, for example, IF BRANCH = 074 DISPLAY FIELD-VALUE.
8. When the program was okay, those statements were removed and the program was sent to be installed into production.

Yes, it was very painful. It was rare to create a program and get it running in the same day.

It was also difficult to keep program versions. Usually the cards were duplicated and special notations written on columns 73 to 60 to denote another version. Later, special tools were available on the market, like LIBRARIAN, that provided the ability to keep program versions on PDS instead of cards, but most installations still kept the card decks and computer printer listings as a backup.

When TSO/ISPF and MVS were born, these helped developers to compile, test, and even debug much more easily. That was in the 1970s. How have we evolved since then?

4.3 Emulating a mainframe on PCs

When PCs were born, we had some solutions and non-IBM products that were designed to help developers build COBOL programs to run on PCs and emulate the mainframe. But was that the best solution? Can we really emulate a mainframe operating system and all its components in a PC?

This infrastructure has problems with application development involving maintenance and significant integration to existing processing and data environments. It can be quite costly to set up emulated environments while actually increasing overall testing time, which includes testing on both the workstation and mainframe.

Today, mainframe products are also changing very fast. For example:

- COBOL has a new verb to parse XML. Does this emulated environment have this new statement in its local COBOL?
- CICS, IMS, and DB2 have many new features and capabilities. CICS Version 3.1 has the capability of implementing services to help with SOA. Is it possible to emulate all those capabilities in the workstation?

Emulating a mainframe could in fact cost more than the savings you get with the CPU cycle. We need a way to simplify the mainframe interaction and use the power of the workstation to perform some trivial activities like editing, syntax checking, and local tests. It would also be useful to have local CICS and DB2 for syntax checking and simple local tests.

4.4 WebSphere Developer for System z

IBM WebSphere Developer for System z V7 includes capabilities that make traditional mainframe development, Web development, Web services development, and integrated composite development faster and more efficient. People that need to code Java, HTML, JSP™, and so on, can use also WebSphere Developer for System z, because it expands the capability of IBM Rational Application Development and Rational Software Architect. WebSphere Developer for System z is not only for a mainframe; it is a tool that can be used for distributed development and mainframe development as well.

WebSphere Developer for System z has many new wizards and capabilities to implement new industry standards, such as service-oriented architecture (SOA) on z/OS, and many installations could justify the usage of WebSphere Developer for System z, just considering basic and traditional mainframe development.

For the rest of this chapter, though, we will focus just on the traditional development that would replace TSO.

4.5 Improving developer productivity with WebSphere Developer for System z

WebSphere Developer for System z is powered by the Eclipse open source platform so developers can adapt and extend their development environment to match their needs and increase their productivity. Here are a few examples of where productivity will improve. (This assumes a scenario where the programmer is performing COBOL program maintenance.)

4.5.1 During the coding phase (code assist)

Suppose the programmer needs to use the CALL CICS statement in COBOL but forgot the syntax. He just types the part of the statement that he does remembers, then presses Ctrl+Space to complete his action.

Suppose the programmer needs to code a MOVE statement to a field defined in working storage. He does not remember the field name, but he knows that it starts with the letter C. He types C and presses Ctrl+Space. All possible COBOL user-defined data names for program usage that starts with C will be listed.

Suppose the programmer accidentally deleted a few lines of code and saved the modifications. He can replace the damaged edition with the modifications that are kept on the workstation, even though the program is located in the z/OS PDS.

How do you do that with ISPF? You cannot, because the file was saved; there is no undo here, so the programmer would need to either recover the backup copy or retype it.

Suppose the programmer needs to use a statement he is not very familiar with, like the new XPARSE COBOL statement. He types XPARSE, then presses F1. The statement from the COBOL documentation that explains the command displays in the Help feature. The full COBOL (or PL/I) manual is available online along with other documentation.

4.5.2 During syntax checking (Compilation)

This is one area where not only will you increase productivity, but you will also likely save some CPU cycles.

Since WebSphere Developer for System z has either COBOL or PL/I compilers installed on the workstation, it is not necessary to submit a job for compilation on z/OS. The programmer just has to perform local syntax checking for the program before submitting a mainframe compilation, and the errors will be listed in the WebSphere Developer for System z Problems view. Also, if you double-click the error message, the COBOL program that is located in the mainframe is edited and an indication of the line in error will be shown. Even someone not familiar with COBOL syntax will be able to locate the error. This is exactly the same behavior that Eclipse uses for all syntax errors in other file types, like HTML, JSP, Java, XML, and others.

4.5.3 JCL generation

WebSphere Developer for System z can generate JCL based on models that we define. When this is done, as an option, the local TCP/IP address of your workstation is captured, because it is required for the interactive debug.

Although optional, this feature can also benefit those who are already familiar with JCL. The smart editor can help find syntax errors, and possibly even save some JCL submissions and mainframe CPU time.

4.5.4 z/OS interactive remote debug

Although you may already have tools to help you in this area, this is probably the most interesting area, and the one where productivity could increase substantially.

With the interactive remote debugging feature, you will be able to run a program on z/OS and have the ability to view and change data contents, establish breakpoints, jump backward and forward in the execution, recover data exceptions, and more. This can be done for batch, CICS, COBOL or PL/I, DB2 stored procedures, debug-called subroutines, and so on. Plus, it is extremely easy to use.

4.5.5 Get z/OS listings with JES monitor

When programmers need to access program listings, most use the SDSF command to do so. Would it not be easier if you could access the job listings as though they were located on your own workstation, so you could use local commands like Print? The JES monitor in WebSphere Developer for System z lets you do just that.

Given the five areas described above, the collective increase in productivity with WebSphere Developer for System z could be upwards of 15% on the conservative side, although some people have reported 30% to 50%. Of course, this is not an easy area to measure, and will depend on many individual and environmental factors.

4.6 “End to end” debug

Remember that WebSphere Developer for System z is an extension of Eclipse and Rational Application Developer, and so a developer would be able to use this same tool to develop and debug composite applications. For example, if you have an application that is written with HTML/JSP/Java that connects to COBOL/CICS through Java connectors, you could perform what we called an “end to end” debug: the same debug perspective will show all the different language types, and you could make breakpoints in any of those languages.

Another example would be a Web page calling a z/OS COBOL DB2 stored procedure in z/OS, in which case, there is a need to debug using the application server components as well the COBOL stored procedure.

There are currently no other tools on the market today that would let the user perform this type of extensive “end to end” debugging within the same development product (IDE).

4.7 How easy is it to replace TSO/ISPF with WebSphere Developer for System z

There is no simple answer here, since a similar question would be, “How easy is it to adopt a new technology?” For some users, this could be very easy, and for others, it could be very difficult, or almost impossible. Remember how hard it was for some programmers to move from the punch card to interactive terminals?

However, one interesting feature of WebSphere Developer for System z is its ability to emulate mainframe screens, which could make adoption easier; if you ever need to jump to TSO/ISPF, or even to CICS terminals, you can connect to either without ever leaving WebSphere Developer for System z. You could also record and play macros that would make this connection very easy (for typing a user ID and password, for example), which would avoid extra typing each time that the mainframe green screen was required.

In situations where the developer really needs TSO/ISPF for interactive CLISTS execution, for example, this could be the best solution.

4.8 WebSphere Developer for System z as the standard interface for the mainframe

We discussed some of the beneficial productivity aspects that could result from using WebSphere Developer for System z. This product has many more features that are detailed in this book. WebSphere Developer for System z will soon be a standard interface for the mainframe, much the way TSO/ISPF is today, unless the developer has an IBM 3270 type of terminal.

The main problem that can be seen at customer locations around the world is that most mainframe developers today have a PC and use it just for e-mail and z/OS screen emulation. Why not make better use of those workstations? Will they need more memory? Probably yes, but how much does 1 GB of memory cost today? It is relatively cheap, and adopting WebSphere Developer for System z is easy to justify.

Archived

5

Enterprise Service Tools

Enterprise Service Tools combines capabilities that were formerly available in the XML Services for the Enterprise plug-in and in the Service Flow Modeler plug-in. Enterprise Service Tools features an enhanced user interface that increases usability and simplifies the overall service and flow development process.

The Enterprise Service Tools perspective provides views and editors that allow you to develop:

- Enterprise Service Tool single-service projects
- Service Flow Projects

Before describing the new Enterprise Service Tool Perspective, we first look at the terminology used.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

5.1 Web Services terminology

Here is an overview of the terminology that will be used in this chapter and that is specific to the Web Service environment.

XML eXtensible Markup Language (XML) is a markup language much like HTML. XML is designed to describe data and to focus on what data is. XML uses a Document Type Definition (DTD) or an XML Schema to describe the data. XML with a DTD or XML Schema is designed to be self descriptive. In the Web environment, XML is now the most common tool for all data manipulation and data transmission.

SOAP Simple Object Protocol (SOAP) is the standard for transporting XML based messages. SOAP is an XML lightweight messaging protocol. SOAP defines an “envelope”. The “envelope” wraps the message itself and identifies the package as a “SOAP Message”. The SOAP Envelope is a structure that describes what is in the XML base message. The SOAP Envelope can be sent through HTTP, Java Message Service, MQ, and so on. The most common transport mechanism for SOAP is HTTP.

WSDL A Web Service Definition Language (WSDL) file is always associated to a Web Service. It is an XML based language file for describing a Web Service and how to access it.

Web service A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols. Web Services Description Language (WSDL) allows enterprise application services to be described in one standard form regardless of the hosting platform (WebSphere, CICS, IMS, .NET, and so on). CICS hosted services can be published alongside Web services hosted on any other platform.

5.2 Enterprise Service Tool perspective

The Enterprise Service Tools perspective contains views that allow you to develop projects.

The first step in using the Enterprise Service Tools is to open the Enterprise Service Tools perspective.

To open the Enterprise Service Tool perspective, perform the following steps:

1. Start WebSphere Developer for System z with a new workspace.
2. From the menu bar, select **Window** → **Open Perspective** → **Other**.

Archive

3. Scroll down and select **Enterprise Service Tools**. Click **OK** to confirm the selection. See Figure 5-1.

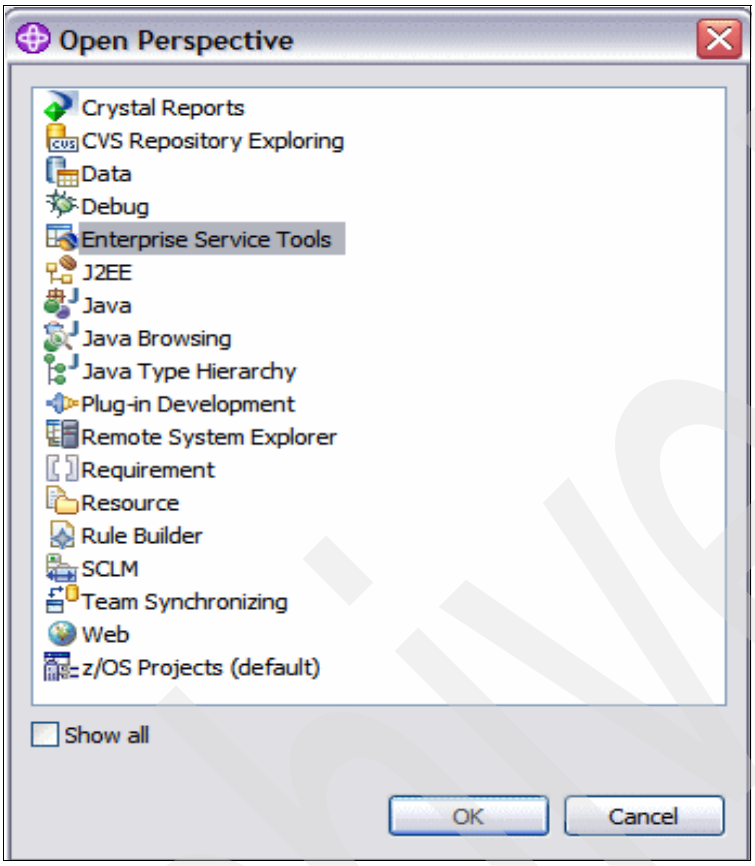


Figure 5-1 Select perspective window

Archived

4. WebSphere Developer for System z now has a changed the title bar, called Enterprise Service Tools. The available views are shown in Figure 5-2:
   - The EST Project Explorer view (in the upper-left pane)

     The EST Project Explorer displays a hierarchical and conceptual view of the projects, subprojects, folders, subfolders, and artifacts in your workspace.
   - The Navigator view (in the upper-left pane)
   - The Editor view (in the upper-right pane)
   - The Outline view (in the lower left pane)
   - The Properties view (in the lower right pane)
   - The Tasks view (in the lower right pane)
   - The Problems view (in the lower right pane)

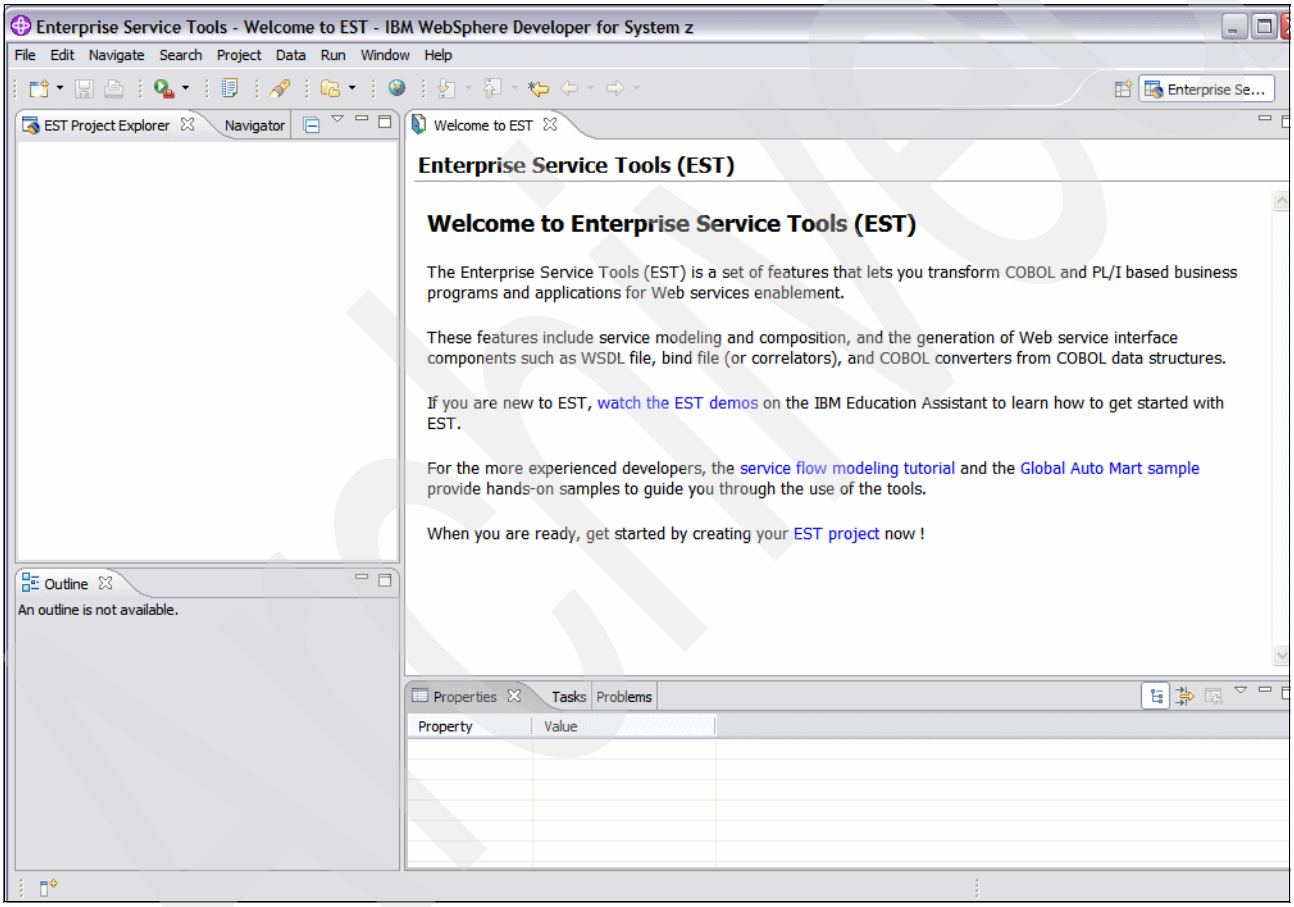


Figure 5-2 The Enterprise Service Tool perspective

5. Close the Welcome EST view.

Note: If the Enterprise Service Tools entry is not listed in the Open Perspective window, then you have to enable the appropriate user role in the workbench.

The first time that you open the Enterprise Service Tools perspective, a pop-up window asks you if you want to enable the corresponding capabilities. If you select **Yes**, then the appropriate user role is enabled in the workbench automatically.

But if you select **No**, and in certain other circumstances, then the next time you attempt to use the Enterprise Service Tools, selections for it will not be available in the workbench menus and windows, including the Open Perspective window, the Show Views window, and the Preferences window.

To make the Enterprise Service Tools available, you must manually enable the appropriate user role in the workbench. To enable the user role, do these steps:

1. On the workbench's menu bar, select **Help** → **Welcome**. The Welcome page opens.
2. In the lower right corner of the Welcome page, click the Enable roles icon. The Enable Roles window opens.
3. In the Enable Roles window, select the icon for z/OS Modernization Developer (advanced). This selection causes certain other roles to be selected as well.
4. Close the Welcome page.

Selections for Enterprise Service Tools should now be available in the workbench menus and windows.

5.3 The Enterprise Service Tools projects

The Enterprise Service Tools project is a concept included in V7 of WebSphere Developer for System z. There are three categories of Enterprise Service Tools projects:

1. The Enterprise Service Tool single-service project. The EST single-service project helps you generate the files associated with a single Web service and helps in the development process of four types of single-service projects:
   - Web Services for CICS project
   - SOAP for CICS project
   - IMS SOAP Gateway project
   - Batch, TSO, and UNIX System Services project
2. The Service Flow project. The Service Flow project is a structural container for the various types of artifacts that you can create with the service flow project tools. It provides an organizational framework for the source files and for the generated files associated with a service flow.
3. The Database application generator project. The Database application generator project provides you with a set of tools to generate COBOL programs for accessing database tables using standard Create Replace Update and Delete (CRUD) and List functions.

Figure 5-3 shows the types of projects available when you want to create a new Enterprise Service Tools project.

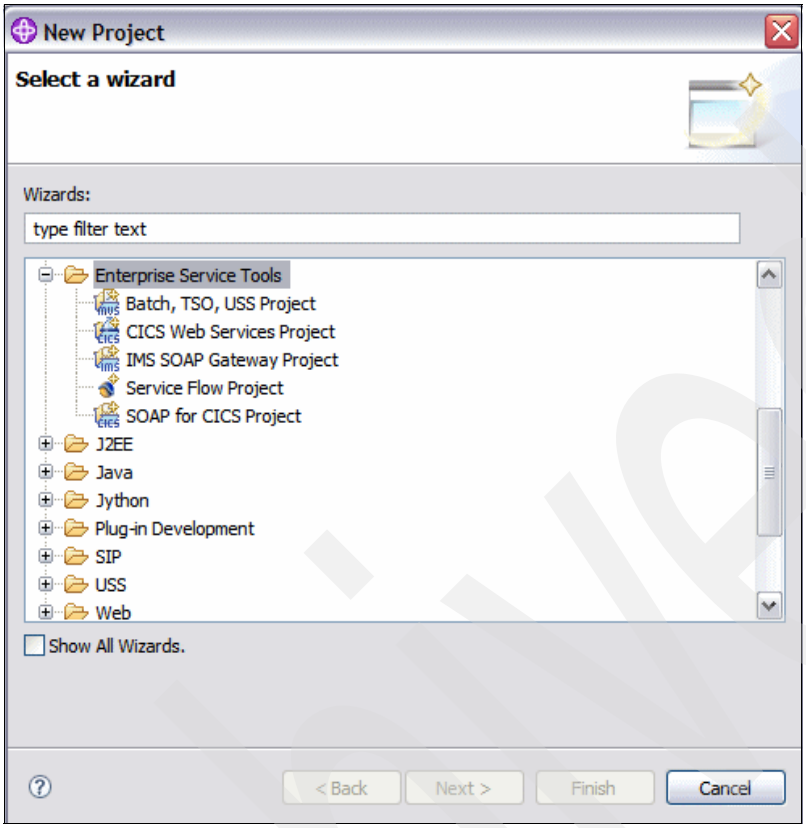


Figure 5-3 The Enterprise Service projects

In this chapter, we will describe those three types of projects.

5.3.1 The Enterprise Service Tool single-service project

An Enterprise Service Tool single-service project is a generic term used to refer any of four specific types of projects in the Enterprise Service Tools: a Web Services for CICS project, a SOAP for CICS project, an IMS SOAP Gateway project, or a Batch, TSO, or UNIX System Services project.

WD/z has wizards that can create COBOL adapter code for parsing the payload of SOAP messages (or other XML documents conforming to the interface of the existing COBOL business programs). The XML parsing code requires Enterprise COBOL Version 3.1 or later. The existing COBOL business programs can be written in older versions of the COBOL language.

Single Web Services enablement styles

Three different enablement styles are available from the Enterprise Service Tool single-service project. Depending on the starting point for the enablement, Figure 5-4 on page 131 shows the artifacts that will be generated.

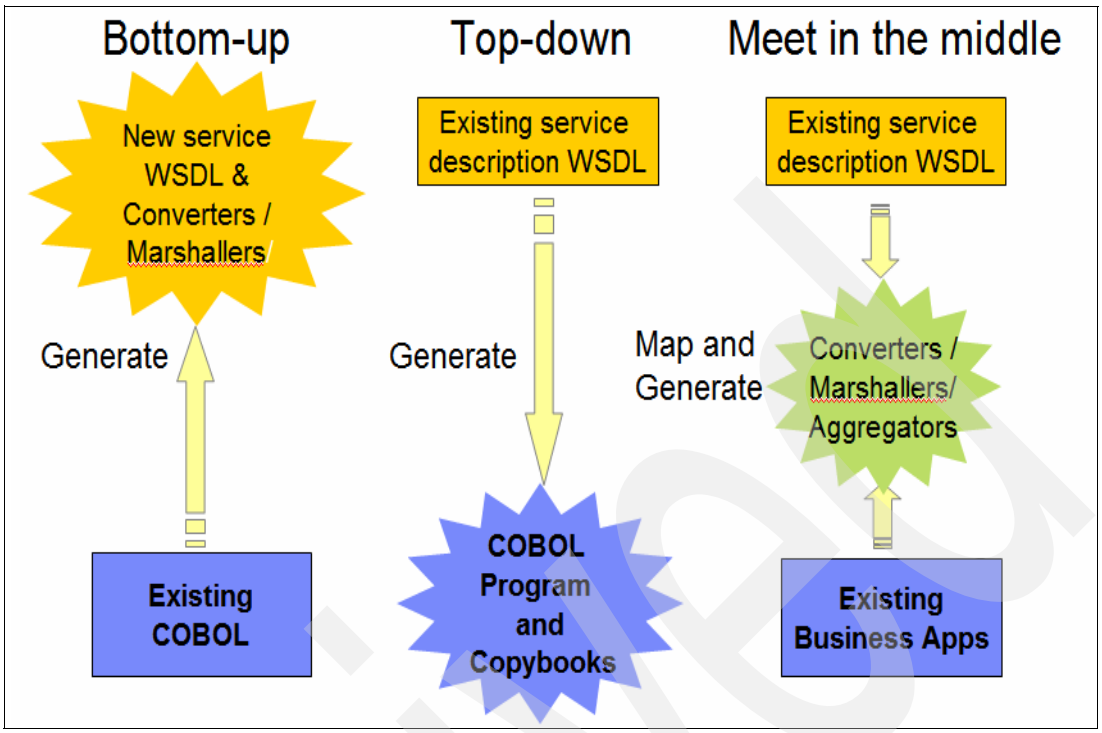


Figure 5-4 Single Web service enablement styles

- Create New Service Interface (bottom-up)

  In this scenario, you generate a Web service description and run time specific XML message processing artifacts from a high level language data structure. This allows you to expose an application program as a service provider.

  You start with an existing COBOL application, select the interface language structure from it, and generate the needed artifacts to deploy it as a new Web service.

- Map to an Existing Service Interface (meet-in-the-middle)

  Note: The Map to an Existing Service Interface (meet-in-middle) wizard is currently not available when you run the Enterprise Service Tools Wizard Launchpad inside the Enterprise Service Tools perspective. However, the wizard is available when you run the wizard outside the Enterprise Tools Perspective.

  In this scenario, you define mappings between high level language structures and WSDL, XML, or XSD files. You also can generate run time specific XML message processing based on the mappings.

  You start with a WSDL file (or XML or XSD file) that defines the interface to an existing Web service and an existing COBOL interface definition and map related fields together. After mapping is completed, you generate compiled converters that follow the mappings.

- Create New Service Implementation (top-down, CICS only)

  In this scenario, you generate a high level language structure and run time specific XML message processing from a Web service description.

  You start with a WSDL document that defines an existing Web service and create a new application from it that implements some or all of the functionality described in the Web service definition. The tool will generate a template COBOL program code for either a new

Web service provider or a new Web service consumer. This new program uses language structures that correspond to the message elements for each operation (operation > input > message > part > element, where element is an XML Schema type) to process requests and issue replies.

To assist you in writing the new application, include files containing the language structure definitions are generated as well as a COBOL template program that includes them.

Web Services for CICS project

A Web Services for CICS project is used when you want to enable an existing CICS application as a Web service that runs under a CICS Transaction Server for z/OS V3 (CICS TS V3.1 or V3.2).

SOAP for CICS project

A Soap for CICS project is used when you want to enable an existing CICS application as a Web service that runs under a CICS Transaction Server for z/OS V2.2 or V2.3.

IMS SOAP Gateway project

An IMS SOAP Gateway is used when you want to enable an existing IMS Cobol application as a Web service. The Web Services artifacts generated by the Enterprise Service Tools have to be deployed to the IMS SOAP Gateway to make the IMS application available as a Web service.

Batch, TSO, or UNIX System Services project

A Batch, TSO, or UNIX System Services project is used when you want to generate Web Services components that are not run time specific. For example, you can have your own Web service processing environment (such as SOAP servers) or if you want to use XML message-based Web service interface for program-to-program communication.

5.3.2 Service flow project

A service flow project lets you develop a Web service that aggregates and processes data from multiple sources. These sources can include CICS non-terminal applications, CICS terminal applications, and other Web services. A service flow project enables you to model a business service from existing CICS applications. This business service will be deployed to a CICS Transaction Server for z/OS V3.1 (CICS TS V3.1) system that has the CICS Service Flow Runtime (CSFR) installed.

Service flow projects support either bottom-up or meet-in-the-middle development approaches. A bottom-up approach is used when the objective is to create a service from an existing application. A meet-in-the-middle approach is used when both the WSDL file describing the service and the implementation component representing the application exist and additional support code that maps the two together needs to be created.

5.3.3 Database Application Generator project

A database application project allows you create a Web-enabled application that accesses a z/OS DB2 database. You can then import the new database application into a service flow project as an outbound Web service subproject. This allows you to invoke the new database application from the service flow project.

6

Web service enablement for an existing CICS application

This chapter provides a reference to assist practitioners in enabling a CICS application as a Web service under CICS.

We will use the CICS catalog manager application to provide an implementation example of WebSphere Developer for System z Web service enablement. This application is a working COBOL application that is shipped with CICS TS V3.1 and is designed to illustrate best practices for exposing CICS applications as Web Services.

In this chapter, we discuss the different steps to enable an existing CICS application as a Web service. We will use the programmatic converters generated by WebSphere Developer for System z as the XML transformation engine.

This chapter shows:

- How to enable your existing CICS application as a Web service
- When to use interpretive versus programmatic generated adapters
- The use of the batch processor

Note: For more information, refer to Appendix B, “Additional material” on page 645 to download the SG247482-Chapter 6.zip file.

© Copyright IBM Corp. 2009. All rights reserved.

6.1 How to transform your existing CICS application in a Web service

Let us discuss how WebSphere Developer for System z (WD/z) tools can enable a CICS program as a CICS Web service.

Figure 6-1 shows what you will generate using WD/z and what will be deployed under CICS. It then shows the execution process when the deployed CICS Web service is invoked.

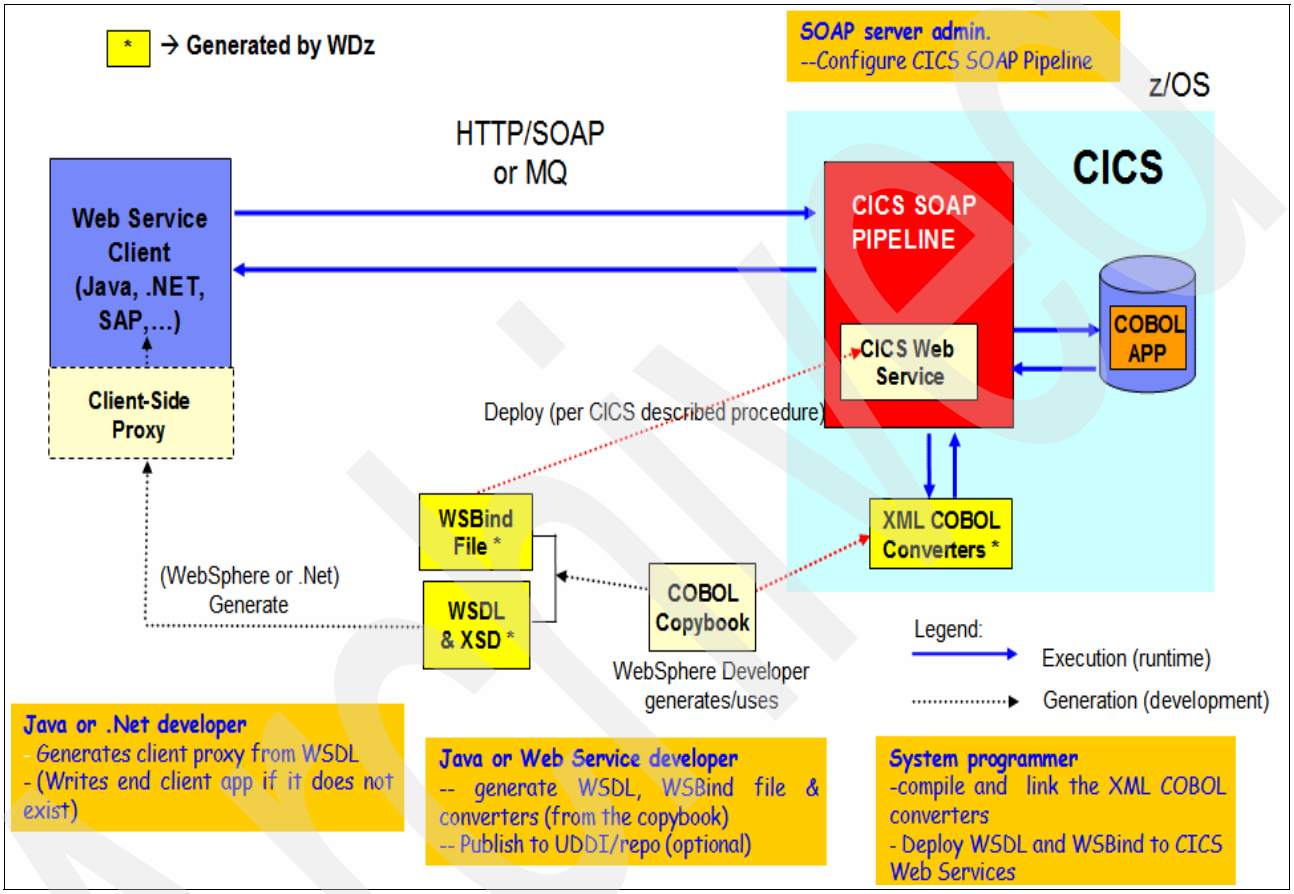


Figure 6-1 CICS TS V3.1 - Web Services support overview

Let us start with the generation process.

6.1.1 Generation of the Web service enablement components

Do these steps:

1. Create an Enterprise Service Tools project to handle the Web service enablement components by selecting **File** → **New** → **Project.** Expand the **Enterprise Service Tools** project and select the **CICS Web Services Project**, as shown in Figure 6-2.

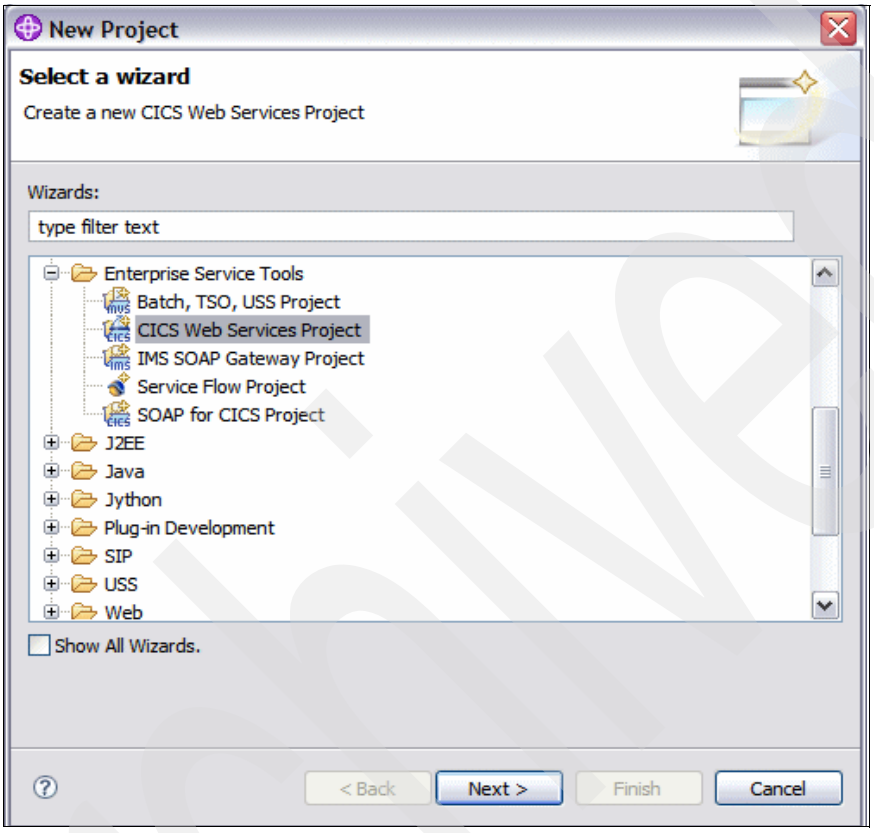


Figure 6-2 EST Project selection

Archive

2. Click **Next** and name your project, such as CICSWS, as shown in Figure 6-3.

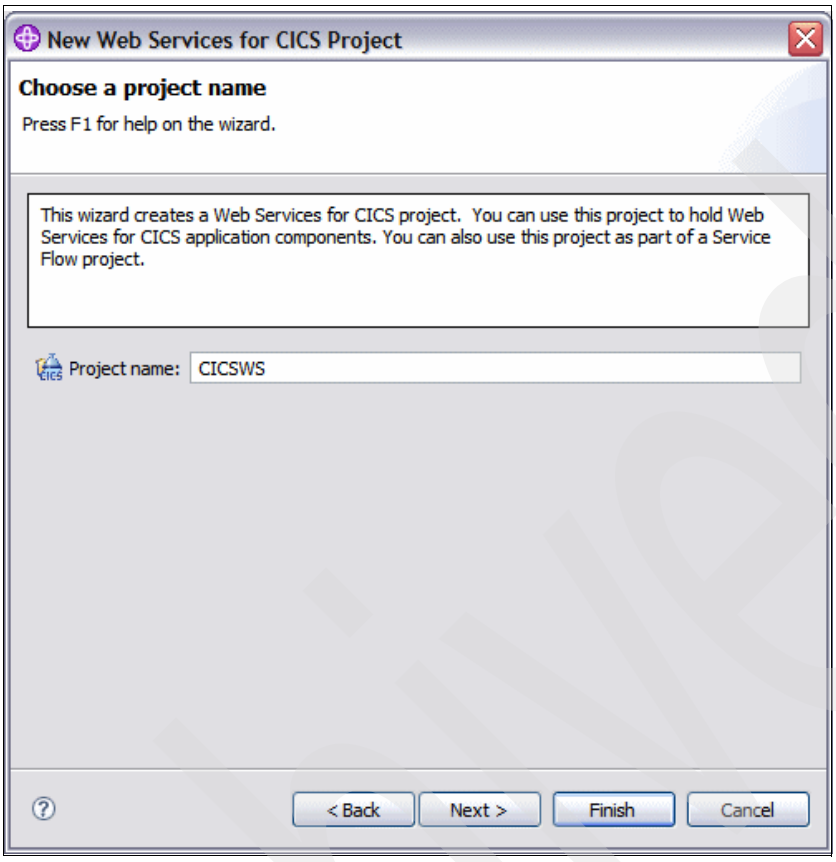


Figure 6-3 Give a name to your CICS single service enablement project

3. Click **Next**. In the new project created, we have to import the program and copybooks used in the Catalog Application that we want to enable as a Web service.

   The Catalog Manager program (DFH0XCMN) and the copybooks it is using (DFH0XCP1, DHF0XCP2, and DFH0XCP4) can be imported using the Import Source Files window.

   From the Import Source Files window, click **File system** and locate the program and copybooks files to import. See Figure 6-4 on page 137.

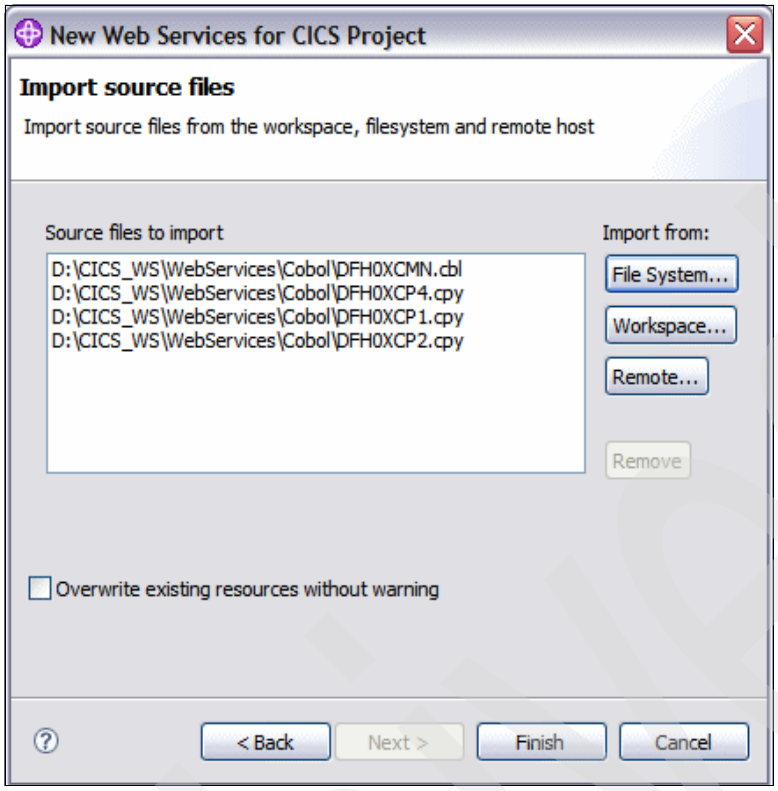


Figure 6-4 Import the Catalog Manager Program and Copybooks in your project

4. Click **Finish.**
5. From the EST Project Explorer view, in the source folder, double click **DFH0XCMN** to open it in the editor view.
6. Locate the Copybook DFH0CP1 that is the commarea used and double click it. An edit window opens. As shown in Example 6-1, you can see that the copy is using redefines.

Example 6-1 Redefines

```
*****************************************************************  01000000
      *                                                               * 02000000
      * CONTROL BLOCK NAME = DFH0XCP1                                 * 03000000
      *                                                               * 04000000
      * DESCRIPTIVE NAME = CICS     (Samples) Example Application -   * 05000000
      *                    Main copybook for example application      * 06000000
      *                                                               * 07000000
      *  @BANNER_START                          01                    * 07166600
      *  Licensed Materials - Property of IBM                         * 07333200
      *                                                               * 07499800
      *  5655-M15             DFH0XCP1                                * 07666400
      *                                                               * 07833000
      *  (C) Copyright IBM Corp. 2004                                 * 07999600
      *                                                               * 08166200
      *  CICS                                                         * 08332800
      *  (Element of CICS Transaction Server                          * 08499400
      *  for z/OS, Version 3 Release 1)                               * 08666000
      *  @BANNER_END                                                  * 08832600
      *                                                               * 09000000
      * STATUS = 6.4.0                                                * 10000000
```

```
*                                                               * 11000000
* FUNCTION =                                                    * 12000000
*      This copy book is part of the example application and     * 13000000
*      defines the commarea interface to the catalog manager     * 14000000
*      module and the datastore modules                          * 15000000
*                                                               * 16000000
*      The fields are as follows                                 * 17000000
*                                                               * 18000000
*        CA-REQUEST-ID            Identifies function           * 19000000
*        CA-RETURN-CODE           Return Code                   * 20000000
*        CA-RESPONSE-MESSAGE      Response message              * 21000000
*        CA-REQUEST-SPECIFIC      Redefined for inquire/order   * 22000000
*                                                               * 23000000
*        CA-INQUIRE-REQUEST       Group for inquire of 15 items * 24000000
*          CA-LIST-START-REF      Reference to start list from  * 25000000
*          CA-LAST-ITEM-REF       Last item returned            * 26000000
*          CA-ITEM-COUNT          Number of items returned      * 27000000
*          CA-CAT-ITEM            Catalog item                  * 28000000
*              CA-ITEM-REF        Item reference number         * 29000000
*              CA-DESCRIPTION     Short description             * 30000000
*              CA-DEPARTMENT      Department item belongs to    * 31000000
*              CA-COST            Cost of item                  * 32000000
*              IN-STOCK           Number of items in stock      * 33000000
*              ON-ORDER           Number of items on order      * 34000000
*                                                               * 35000000
*        CA-INQUIRE-SINGLE        Structure for inquire single  * 36000000
*          CA-ITEM-REF-REQ        Reference number of item      * 37000000
*          CA-SINGLE-ITEM.                                      * 38000000
*            CA-SNGL-ITEM-REF     Item reference number returned* 39000000
*            CA-SNGL-DESCRIPTION  Short description             * 40000000
*            CA-SNGL-DEPARTMENT   Department item belongs to    * 41000000
*            CA-SNGL-COST         Cost of item                  * 42000000
*            IN-SNGL-STOCK        Number of items in stock      * 43000000
*            ON-SNGL-ORDER        Number of items on order      * 44000000
*                                                               * 45000000
*        CA-ORDER-REQUEST         Structure for placing an order* 46000000
*          CA-USERID              User name placing the order   * 47000000
*          CA-CHARGE-DEPT         Department user belongs to    * 48000000
*          CA-ITEM-REF-NUMBER     Item reference to be ordered  * 49000000
*          CA-QUANTITY-REQ        Quantity of item required     * 50000000
*                                                               * 51000000
*---------------------------------------------------------------* 52000000
*                                                               * 53000000
* CHANGE ACTIVITY :                                             * 54000000
*      $SEG(DFHOXCP1),COMP(SAMPLES),PROD(CICS    ):             * 55000000
*                                                               * 56000000
*   PN= REASON REL YYMMDD HDXXIII : REMARKS                     * 57000000
*   $DO= I07544 640 040910 HDIPCB  : EXAMPLE - BASE APPLICATION * 58000000
*                                                               * 59000000
***************************************************************** 60000000
*    Catalogue COMMAREA structure                                 61000000
     03 CA-REQUEST-ID            PIC X(6).                       62000000
     03 CA-RETURN-CODE           PIC 9(2).                       63000000
     03 CA-RESPONSE-MESSAGE      PIC X(79).                      64000000
     03 CA-REQUEST-SPECIFIC      PIC X(911).                     65000000
```

```
*    Fields used in Inquire Catalog                                  66000000
     03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.             67000000
         05 CA-LIST-START-REF        PIC 9(4).                       68000000
         05 CA-LAST-ITEM-REF         PIC 9(4).                       69000000
         05 CA-ITEM-COUNT            PIC 9(3).                       70000000
         05 CA-INQUIRY-RESPONSE-DATA PIC X(900).                     71000000
         05 CA-CAT-ITEM  REDEFINES CA-INQUIRY-RESPONSE-DATA          72000000
                        OCCURS 15 TIMES.                            73000000
             07 CA-ITEM-REF          PIC 9(4).                       74000000
             07 CA-DESCRIPTION       PIC X(40).                      75000000
             07 CA-DEPARTMENT        PIC 9(3).                       76000000
             07 CA-COST              PIC X(6).                       77000000
             07 IN-STOCK             PIC 9(4).                       78000000
             07 ON-ORDER             PIC 9(3).                       79000000
*    Fields used in Inquire Single                                   80000000
     03 CA-INQUIRE-SINGLE REDEFINES CA-REQUEST-SPECIFIC.             81000000
         05 CA-ITEM-REF-REQ          PIC 9(4).                       82000000
         05 FILLER                   PIC 9(4).                       83000000
         05 FILLER                   PIC 9(3).                       84000000
         05 CA-SINGLE-ITEM.                                          85000000
             07 CA-SNGL-ITEM-REF     PIC 9(4).                       86000000
             07 CA-SNGL-DESCRIPTION  PIC X(40).                      87000000
             07 CA-SNGL-DEPARTMENT   PIC 9(3).                       88000000
             07 CA-SNGL-COST         PIC X(6).                       89000000
             07 IN-SNGL-STOCK        PIC 9(4).                       90000000
             07 ON-SNGL-ORDER        PIC 9(3).                       91000000
         05 FILLER                   PIC X(840).                     92000000
*    Fields used in Place Order                                      93000000
     03 CA-ORDER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.              94000000
         05 CA-USERID                PIC X(8).                       95000000
         05 CA-CHARGE-DEPT           PIC X(8).                       96000000
         05 CA-ITEM-REF-NUMBER       PIC 9(4).                       97000000
         05 CA-QUANTITY-REQ          PIC 9(3).                       98000000
         05 FILLER                   PIC X(888).                     99000000
```

7. In order to be able to handle a complex copybook like the one that uses redefines, we will have to select the **Compiled XML Conversion** as conversion type. The native CICS interpreter cannot handle such copy.

   From the EST Project Explorer view, select the program name **DFH0XCMN.cbl** and from the context menu, select **Enable Enterprise Web Service**.

   From the Enterprise Service Tools Wizard Launchpad:

   – As we are in a Web Services for CICS project creation, the run time **Web Services for CICS** is already specified.

   – As we will start from the COBOL program to create the Web service, select the scenario **New Web Service Interface (Bottom-up)**.

   – As the Commarea used by the CICS program uses redefines, select the conversion type **Compiled XML Conversion**.

Figure 6-5 shows the options specified.

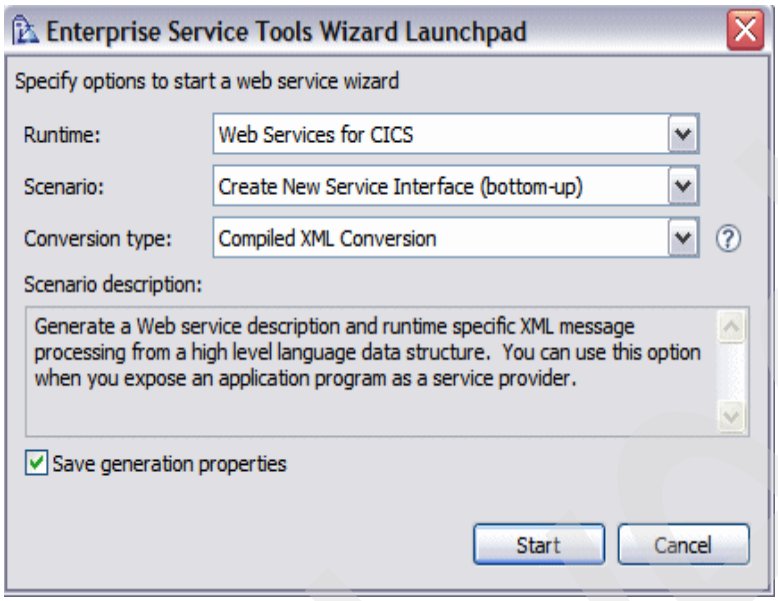


Figure 6-5 EST Launchpad Wizard options to start the Web service wizard

Note: The **Save generation properties** button is already checked, which will keep a record of the generation options and language structure selections that you will specify for regeneration, audit, or debugging purposes.

8. Click **Start** to start the Web service for CICS dialog.

Archive

9. In the Language structure window, specify the commarea structure for inbound, as shown in Figure 6-6.

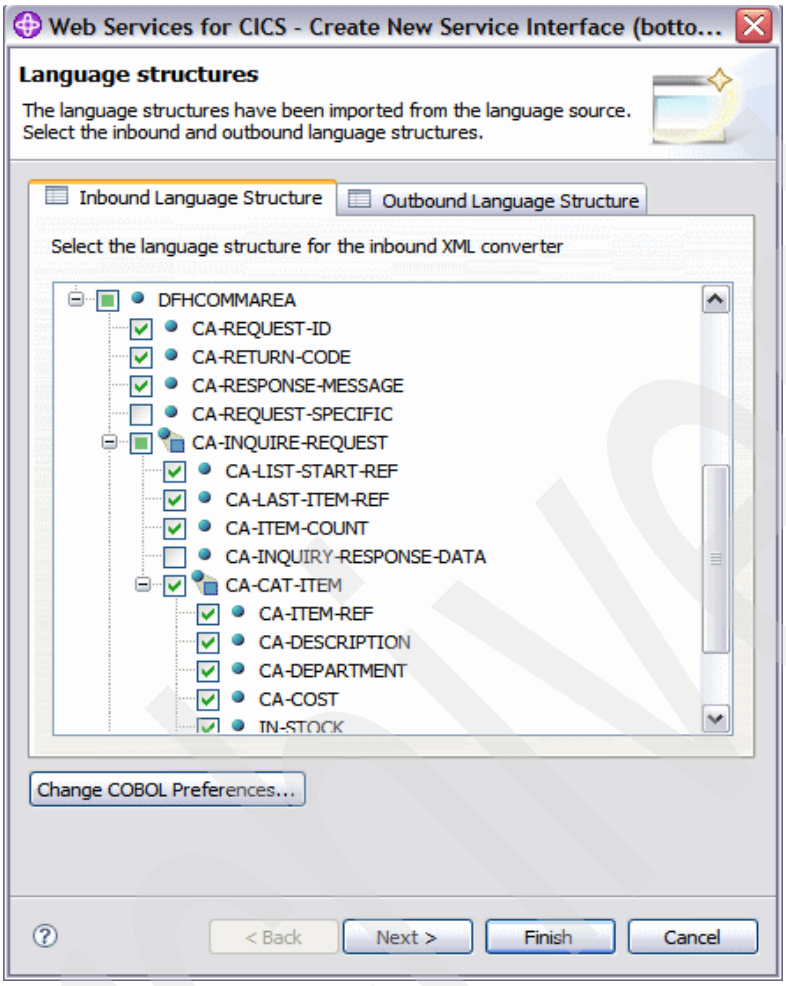


Figure 6-6 Inbound Language Structure tab

Archived

10. On the Outbound Language Structure tab, specify the language structure for outbound, as shown in Figure 6-7.

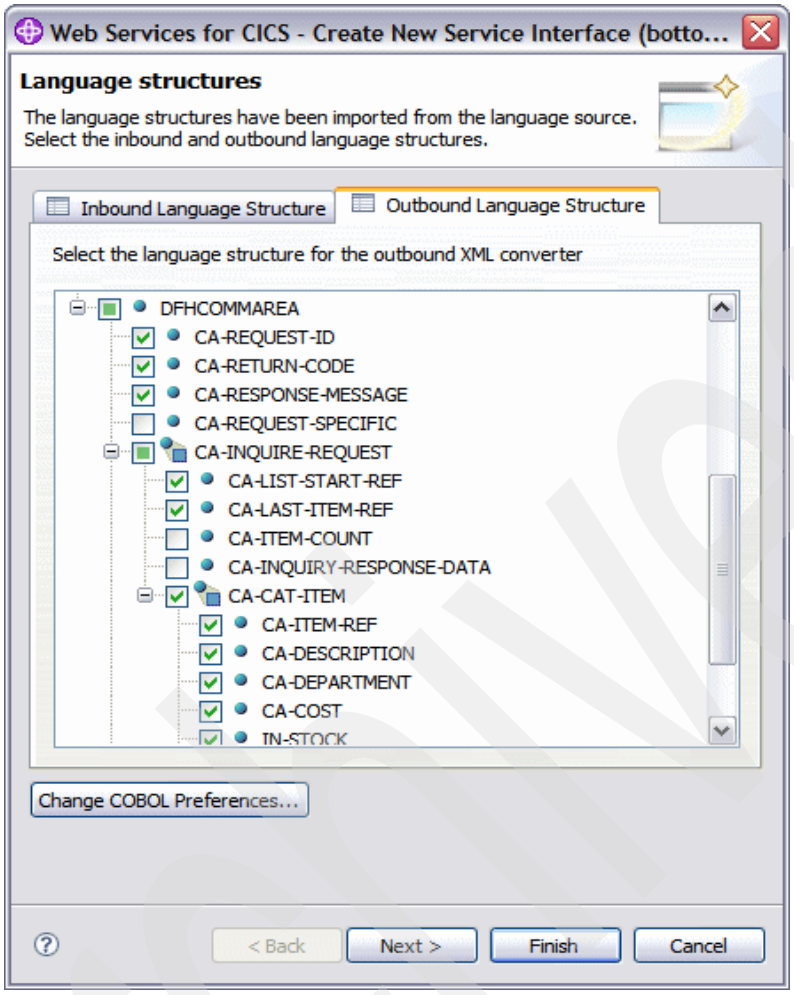


Figure 6-7 Outbound Language Structure tab

11. Click **Next**. You may keep the Attributes Identification proposed in the Generation Options window.

12. As WSDL properties, specify the server location value, in the format http://<hostname>:<soap_port>/<web_service_uri>, as shown in Figure 6-8 on page 143.

Archived

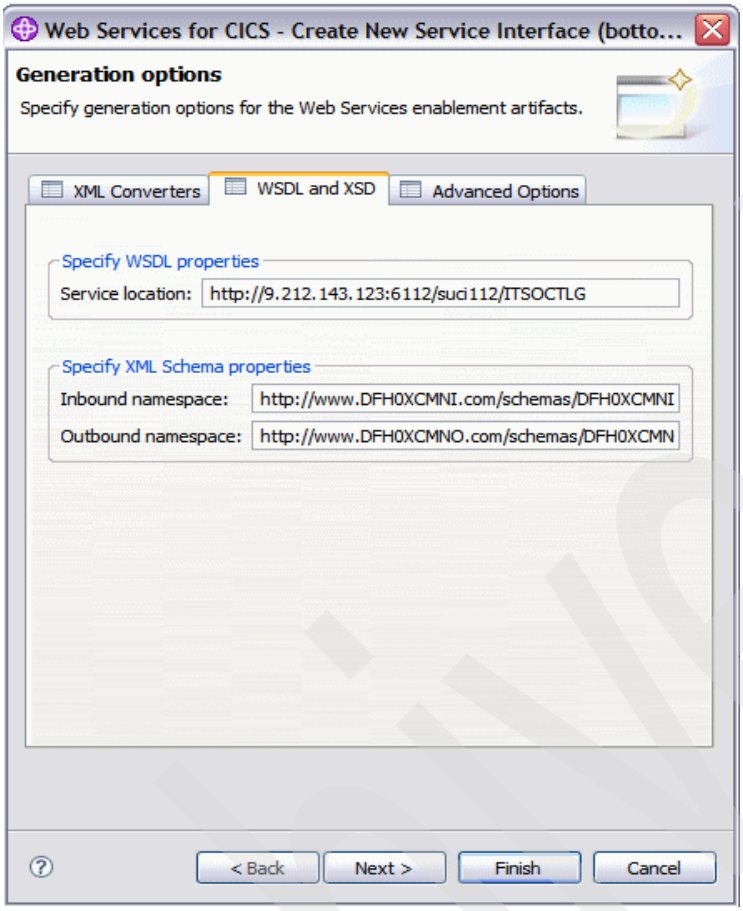


Figure 6-8 Specify the address of the Web service provider

13. Click **Next**.

Archived

14. In the File, data set and member location window, check the **Generate All to Driver** check box. This will result in a single program being deployed under CICS, as shown in Figure 6-9.

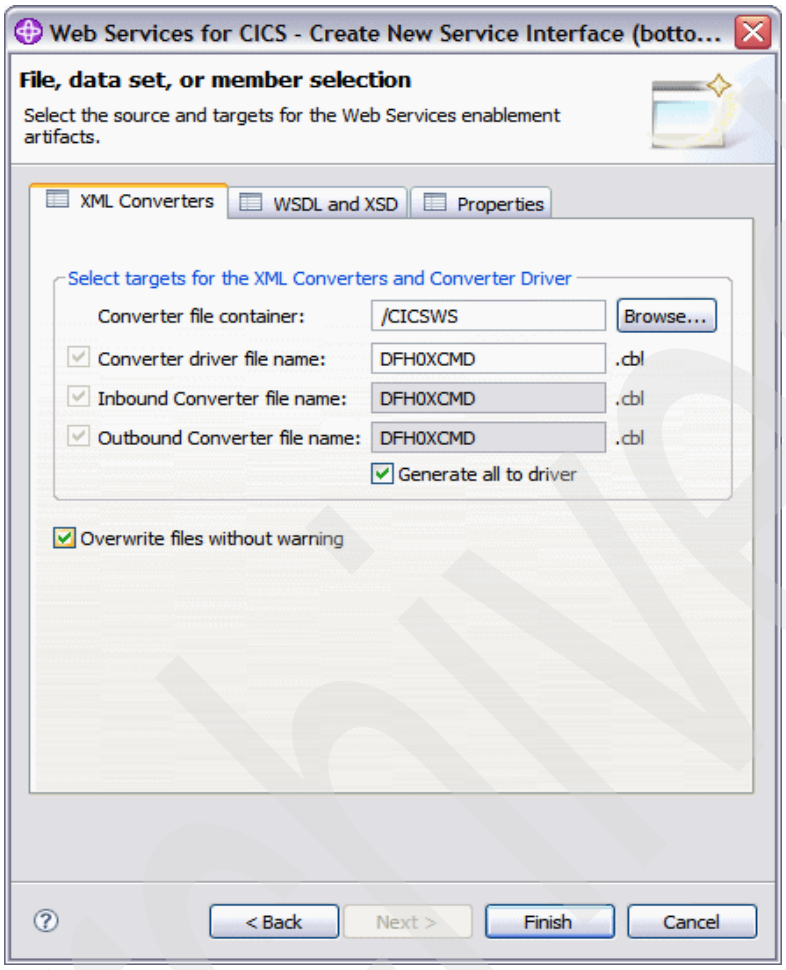


Figure 6-9 Specify to generate all into the driver program

15. Click the **WSDL and XSD** tab and ensure that the WSDL file name is selected.
16. Click **Finish**.

After the Web Services Enablement wizard completes its generation, you should see XML artifacts in the CICSWS Project under the Generation folder. The .xml files are properties that will be used in the next wizard generation run for this project.

Under the Target subfolder, you should see the generated artifacts:

- DFH0XCM.wsdl: The WSDL file that describes this CICS based Web service.
- DFH0XCMD.cbl: The conversion program that is made up of the converter driver and input and output converters.
- DFH0XCMI.xsd and DFH0XCMO.xsd: The XML Schema Definition files that describe the input and output XML documents respectively.
- The *.log file that is the generation log.
- DFH0XCMN.wsbind: The WSBind file that tells CICS how conversion is to be performed and what the URI map looks like.

Note: As XML parsing has been supported since IBM Enterprise Cobol for z/OS V3.1, we need to V3.1 or later to compile our converter program.

Support includes a high-speed parser that enables your COBOL programs to:

- Process XML documents in the principal runtime environments, such as CICS, IMS, and MQSeries®.
- Validate XML documents to ensure that they conform to established XML standards.
- Populate COBOL data structures with the content of XML documents.

You can double-click the generated conversion program and do a Find on XML Parse. You can see that the XML parsing is done by the converter program, as shown in Figure 6-10.

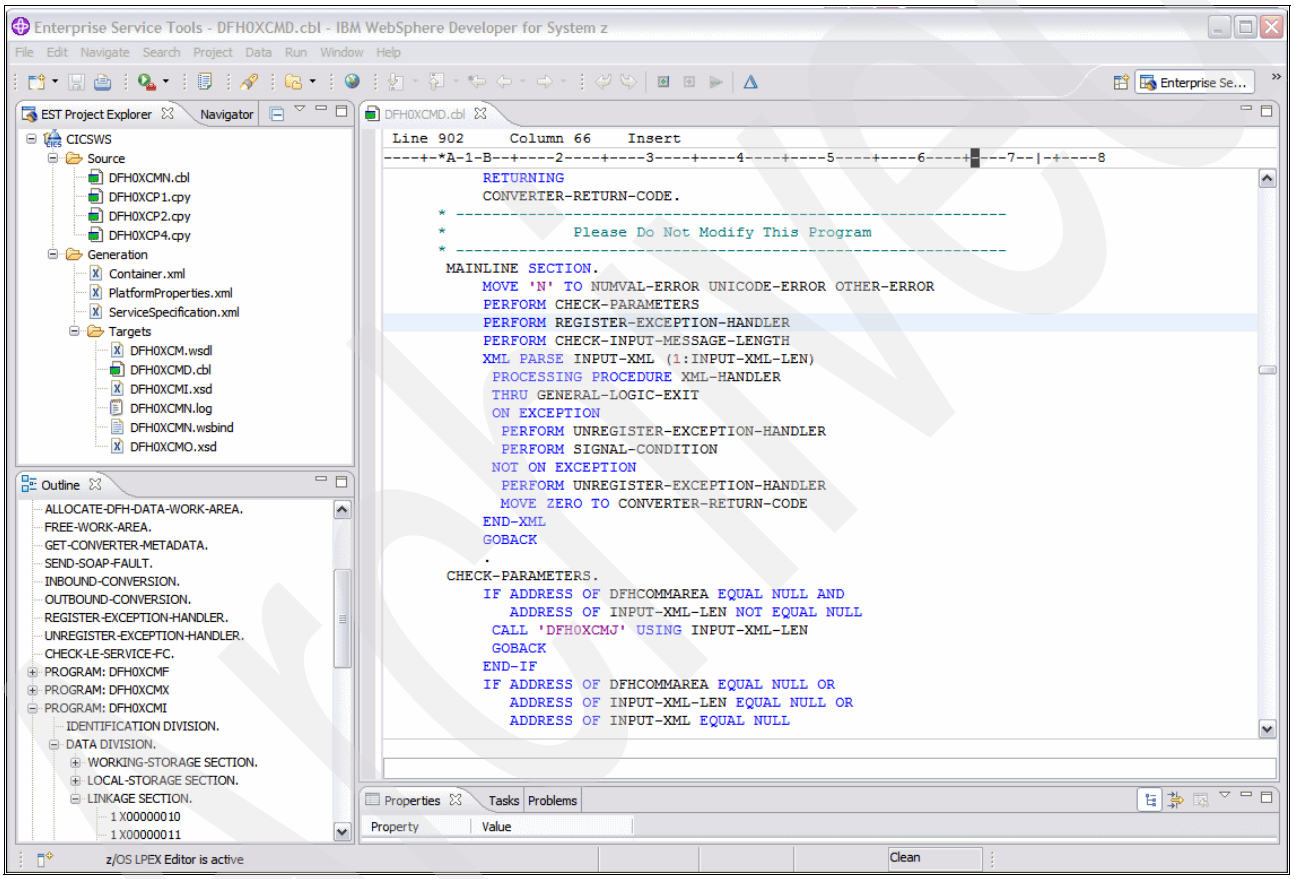


Figure 6-10 The XML converter program

The generation is done. Let us continue with the deployment process.

6.1.2 Configuring CICS Web Services on z/OS

In order to use an existing CICS program as a Web service, you have to set up the CICS infrastructure. There are three main tasks for this setup:

- Create the HFS Directories.
- Create a pipeline resource definition.
- Create a TCPIPSERVICE resource definition.

Set up the CICS infrastructure

To use a CICS program as a Web service over HTTP, TCPIPSERVICE and PIPELINE resources are required.

You can use the WD/z Remote System View to create these HFS directories.

Create the HFS directories

The PIPELINE resource definition points to some Hierarchical File System (HFS) directories and also points to a pipeline configuration file.

The setup requires configuring files and directories in the Hierarchical File System (HFS).

Two directories have to be defined under HFS:

1. The shelf directory

   /etc/cicscfg/sharedir/shelf is used in our implementation.

   The shelf directory is used by CICS to store the Web service binding files that are associated with Web service resources. Each Web service resource is, in turn, associated with a PIPELINE. The shelf directory is managed by CICS together with the PIPELINE resource and you should not modify its contents directly. Several PIPELINEs can use the same shelf directory, as CICS ensures a unique directory structure beneath the shelf directory for each PIPELINE.

2. The pickup directory

   /etc/cicscfg/sharedir/wsdir is used in our implementation.

   The pickup directory contains the Web service binding files that are to be associated with a PIPELINE. When a PIPELINE is installed, or in response to a PERFORM PIPELINE SCAN command, CICS searches this directory for files ending in ".wsbind". For each one, CICS dynamically creates a Web service and URIMAP resource, associates them with the PIPELINE and installs them ready for use.

Archived

Let us now create these directories:

1. To create the HFS directories, open a z/OS project perspective and, if it is not already done, define the z/OS system connection. See Figure 6-11.

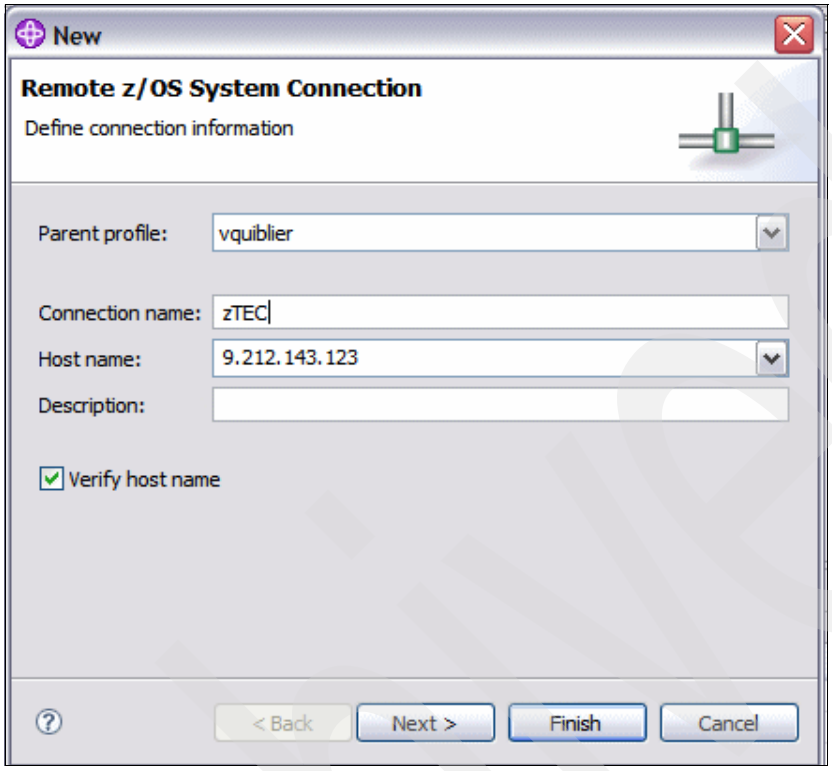


Figure 6-11 Create the connection to your z/OS system

Archive

2. Connect to the defined z/OS system using your TSO user ID and password.

   You have two alternatives to create these directories:

   – Use the UNIX System Services Command shell provided by WD/z, which you can launch by using the contextual menu of the UNIX System Services Shell in the Remote system view. See Figure 6-12.

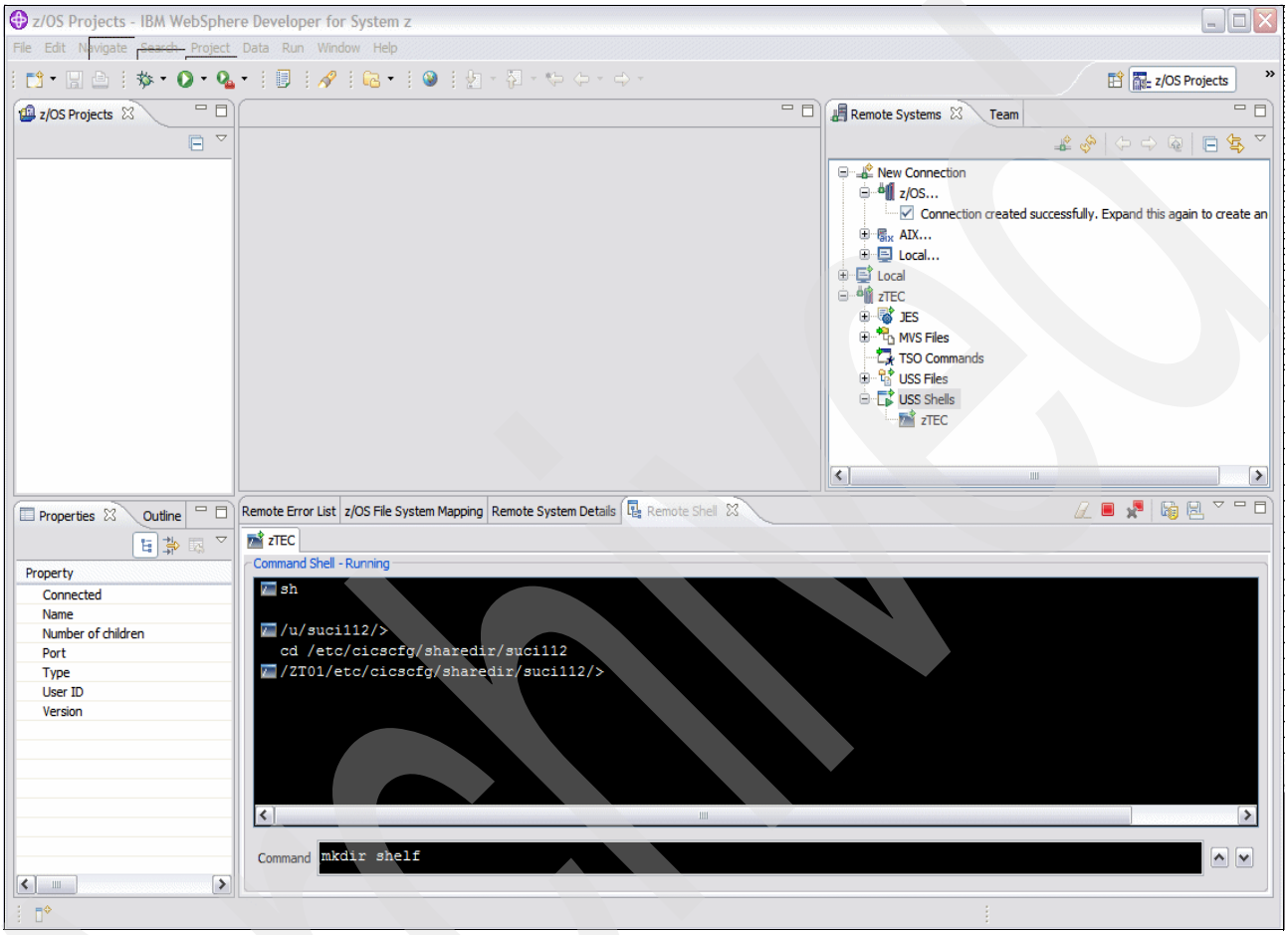


Figure 6-12 The UNIX System Services command shell

   – Use the UNIX System Services file tree view from the Remote System view. This alternative provides you with all the facilities you get for the MVS files, including copy/paste, which will be used to upload the local generated WSBIND file to the appropriate UNIX System Services directory.

You may specify a filter for the UNIX System Services files in order to more quickly access your directories. See Figure 6-13.

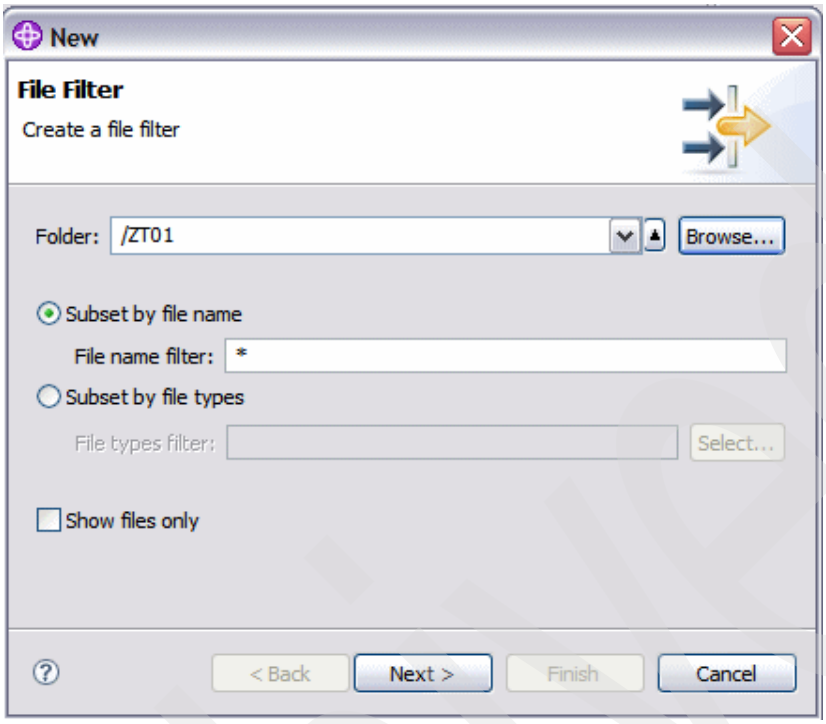


Figure 6-13 Specify the appropriate UNIX System Services filter

Archive

3. Use the contextual menu from the UNIX System Services Files folder in the Remote System View to create the new folders shelf and wsdir:
   - /etc/cicscfg/sharedir/shelf
   - /etc/cicscfg/sharedir//wsdir

   See Figure 6-14.

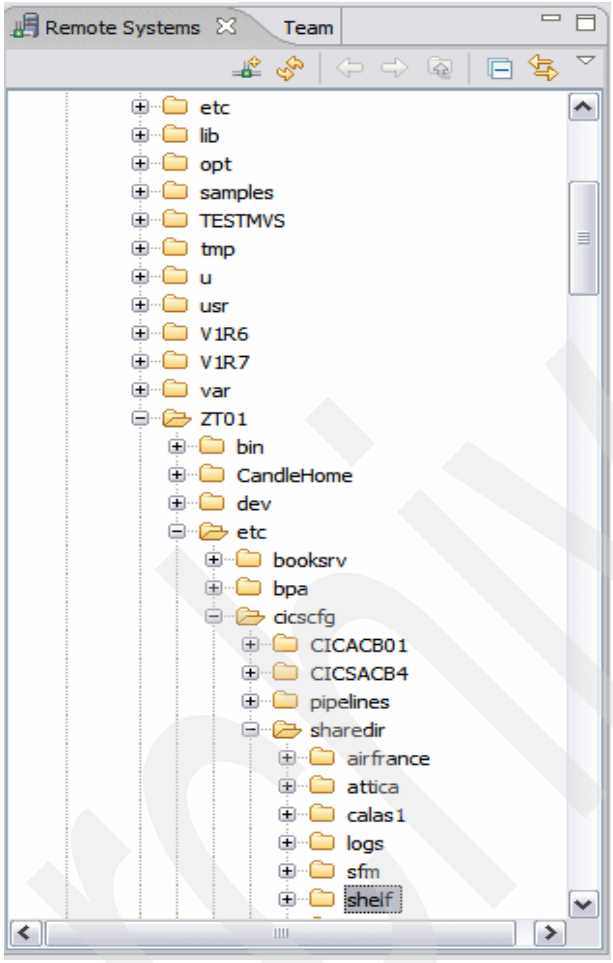


Figure 6-14 Create the UNIX System Services folder required using the Remote System view

Create the PIPELINE resource definition

The complete definition of a pipeline consists of a PIPELINE resource and a pipeline configuration file. The configuration file is in XML format and details the message handlers that will act on Web service requests and responses as they pass through the pipeline.

We are going to use the CICS supplied PIPELINE configuration file for a SOAP 1.1 handler.

More than one WEBSERVICE can share a single PIPELINE, therefore you need define only one pipeline for the inbound requests of our example application. You must, however, define a second PIPELINE for the outbound requests, as a PIPELINE resource cannot be configured to be both a provider and requester pipeline at the same time.

Do these steps:

1. From WebSphere Developer for system z, use the Host Emulation Support to connect to your CICS.

Archived

2. Select your system name.
3. From the contextual menu, select **Host Connection Emulator Support**.

   This opens the mainframe login window associated with your z/OS system IP address in the Editor view. See Figure 6-15.

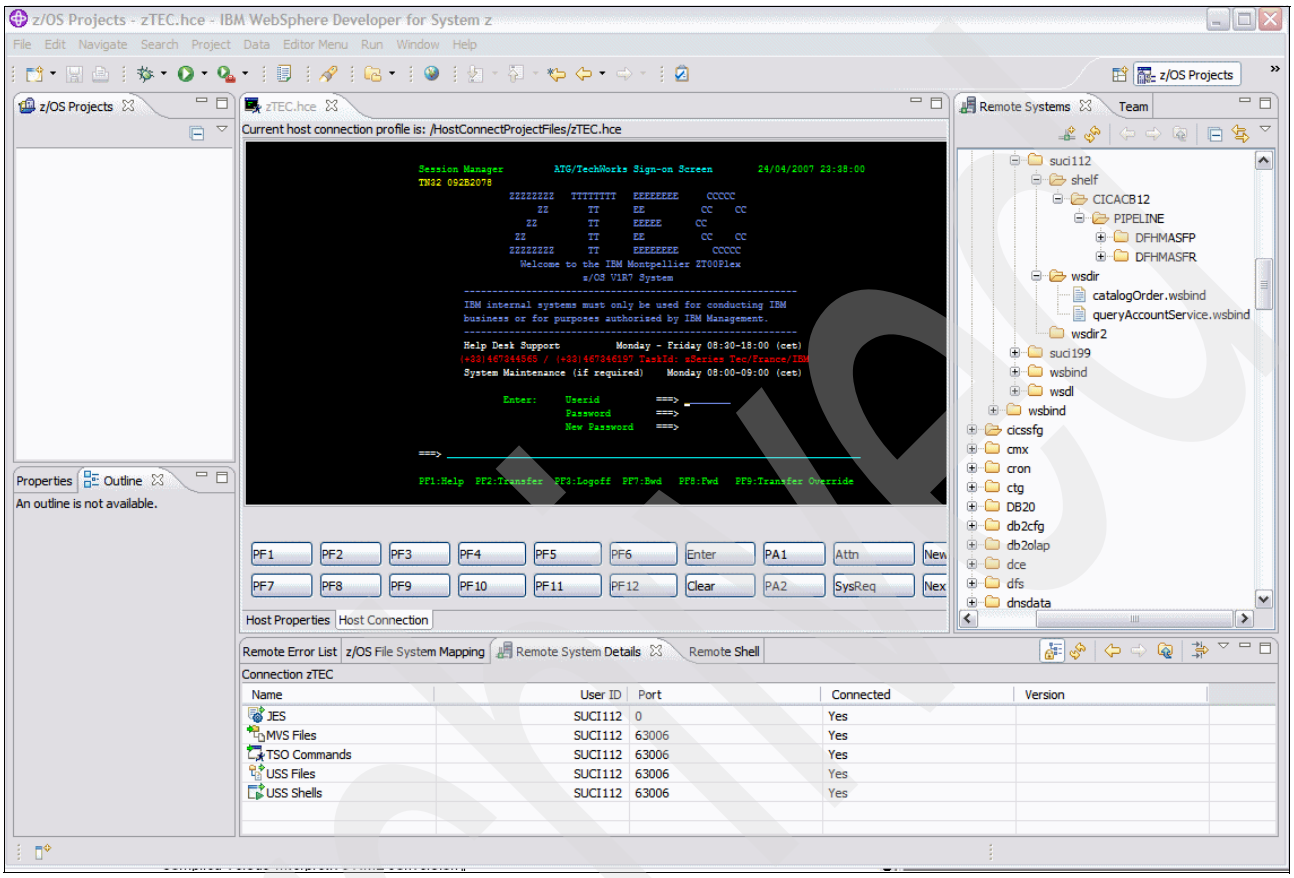


Figure 6-15 Open the Host Connection Emulator support

It is easier to work on the mainframe screen by making the Editor view expand to a full screen. Just double click the **zTEC.hce** tab. You can then connect to your CICS and define the PIPELINE resource within WebSphere Developer for system z.

4. Type `CEDA DEFINE GROUP (groupname) PIPELINE (pipename)` and press Enter at a clear window on the CICS session.
5. Type the additional attributes according to your implementation. See Table 6-1.

Table 6-1 Attributes for the Pipeline definition

Attributes	Value
DESCRIPTION	Basic SOAP V1.1 provider pipeline
STATUS	Enabled
CONFIGFILE	/usr/lpp/cicsts/cicsts.v3r1/samples/pipelines/basicsoap11provider.xml
SHELF	/etc/cicscfg/sharedir/shelf/
WSDIR	/etc/cicscfg/sharedir/wsdir/

6. Press Enter. You should see a window similar to Figure 6-16.

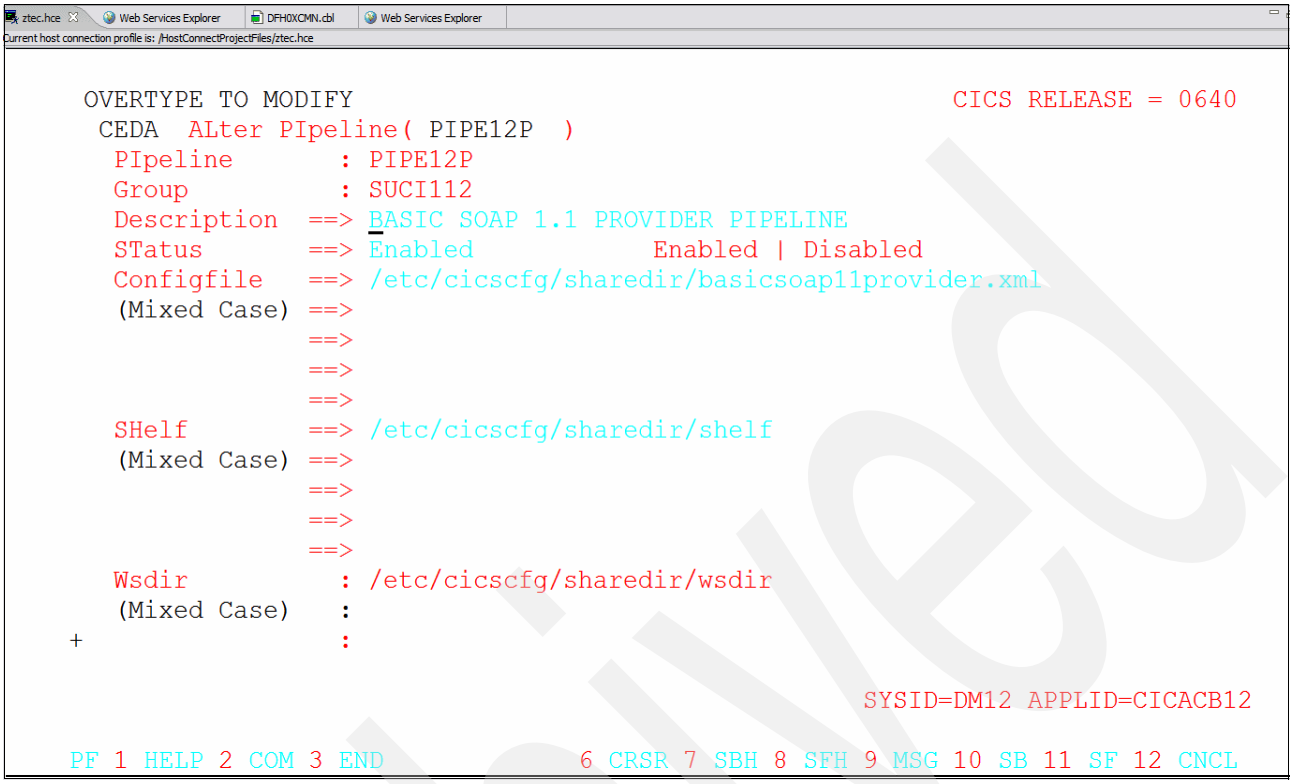


Figure 6-16 PIPELINE resource definition

You can check the directory containing the CICS supplied pipeline configuration files using the UNIX System Services files tree view from the Remote System view. Look at the CICS supplied requester pipeline configuration file. See Figure 6-17.

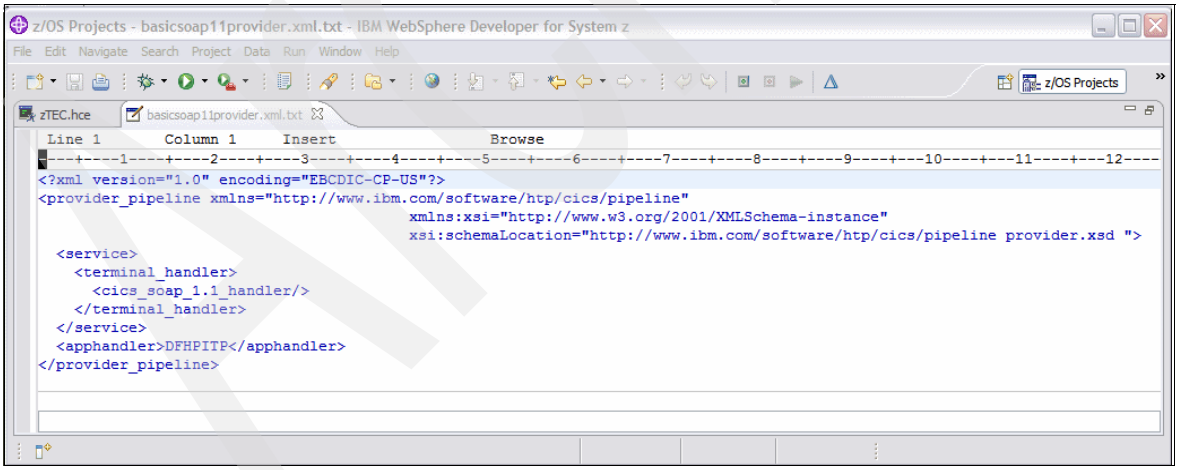


Figure 6-17 The basicsoap11provider.xml file

Note the application handler for Web service requesters DFHPITP.

DFHPITP is a CICS supplied program that:

- Locates the WSBind file associated with the Web service using this pipeline.
- Performs the XML to COMMAREA (or Container) conversion specified in the WSBind file.

- Invokes the user application program specified in the WSBind file.
- Performs COMMAREA (or Container) to XML conversion.

Create the TCPIPSERVICE resource definition

In our sample, the Web service client will connect to CICS using HTTP. Therefore, you need to define a TCPIPSERVICE resource to receive the inbound HTTP request.

Do these steps:

1. From your CICS window in WebSphere Developer for System z, type `CEDA DEFINE GROUP(`*groupname*`) TCPIPSERVICE(`*portname*`)` and press Enter at a clear window on the CICS session.
2. Type the CICS Web service Port Number value in the Port Number field, as shown in Figure 6-18.

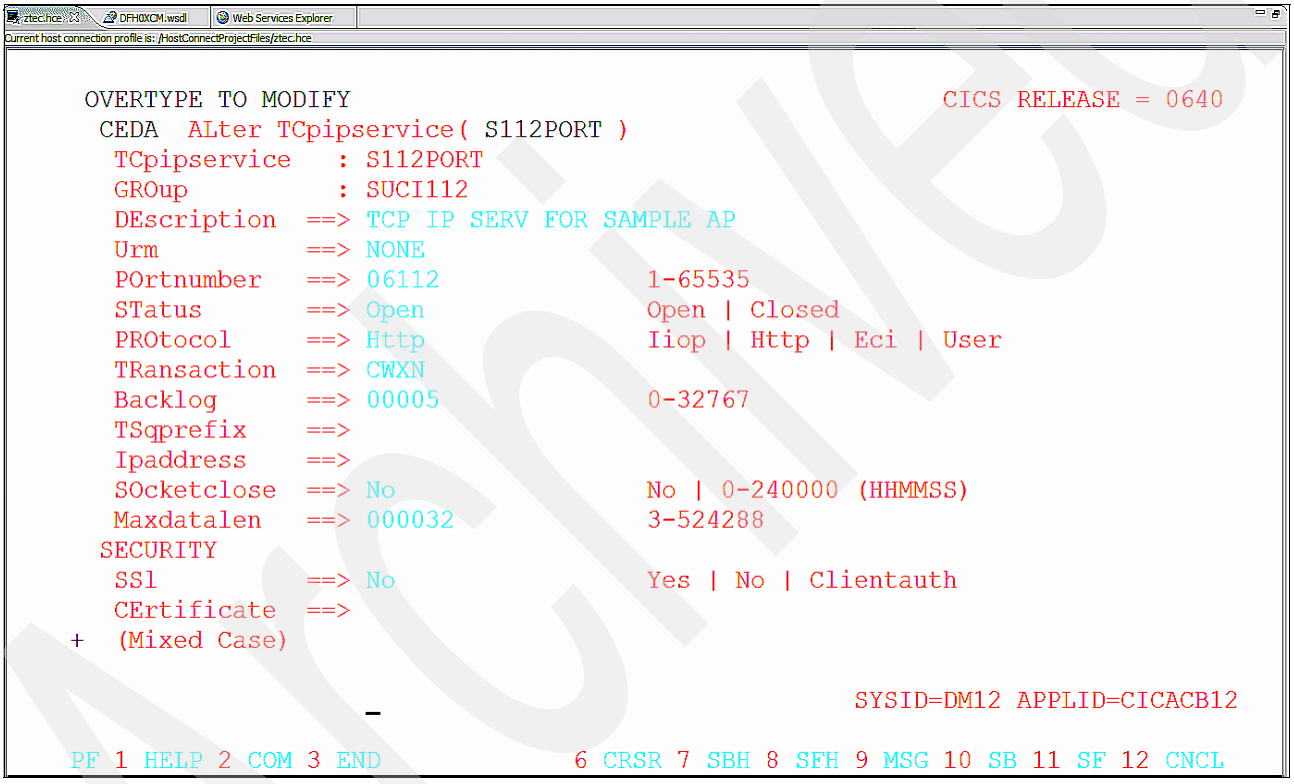


Figure 6-18 The TCPIPSERVICE definition

3. To install the pipename and the portname resources just created, type CEDA INSTALL GROUP(groupname) and press Enter.
4. Look for the `INSTALL SUCCESSFUL` message.

6.1.3 Deploy the generated assets to CICS Transaction Server for z/OS

Now that the CICS environment is ready, you have to:

- Move the generated assets to your z/OS system.
- Generate the JCL to compile the created COBOL converter.
- Create the CICS resource required to execute the CICS Web Services.

Archived

Move the generated assets to your z/OS system

Let us assume that you generated the assets located in the folder CICSWS-generation-targets. See Figure 6-19.

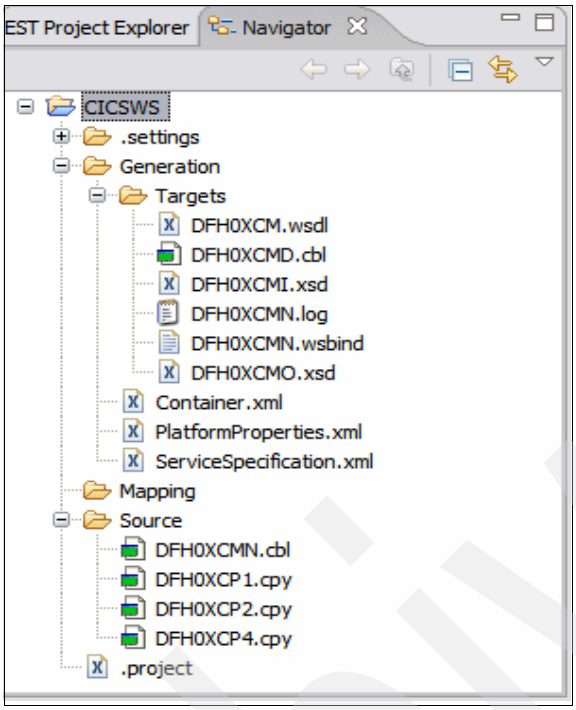


Figure 6-19 The Web service enablement components generated

The assets DFH0XCMN.wsbind and DFH0XCMD.cbl have to be deployed to the z/OS system.

Deploy the Web service binding file

When a PIPELINE scan command is issued, either explicitly through a CEMT PERFORM PIPELINE () SCAN or automatically during a PIPELINE installation, CICS scans the pickup directory to search for Web service binding files, that is, for files with the .wsbind extension. For each binding file found, CICS determines whether to install WEBSERVICE and URIMAP resources.

Your generated .wsbind file must be moved to the HFS directory that the PIPELINE scan will look for, in our implementation, the /etc/cicscfg/sharedir/wsdir directory.

Note: Be careful when you transfer WSBind to z/OS! The WSBind file is an EBCDIC binary file, and needs to be transmitted as a binary to the target z/OS system. You can use the Remote System Explorer facilities of WD/z to copy to an HFS directory on the target z/OS system.

In order to move the WSBind file to z/OS, we need to make sure that the file extensions of WSBind are handled as binary files. You can check that this is true by using the Preferences dialog (select **Windows** → **Preferences** → **Remote Systems** → **Files**) and check that the file type of *.wsbind has the default binary file transfer mode. You do not need to make any changes.

Archived

Do these steps:

1. Using the Enterprise Service Perspective and the EST Project Explorer, right-click **DFH0XCMD.wsbind** and select **Copy**.
2. To paste the .wsbind file program into the HFS pickup directory that contains the Web service binding files, switch to the z/OS perspective. Locate the pickup directory from the UNIX System Services files folder, /etc/cicscfg/sharedir/wsdir in our implementation, right-click it, and select **Paste**.

Deploy the Web service converter source file

Do these steps:

1. Using the Enterprise Service Perspective and the EST Project Explorer, right-click **DFH0XCMD.cbl** and select **Copy**. See Figure 6-20.

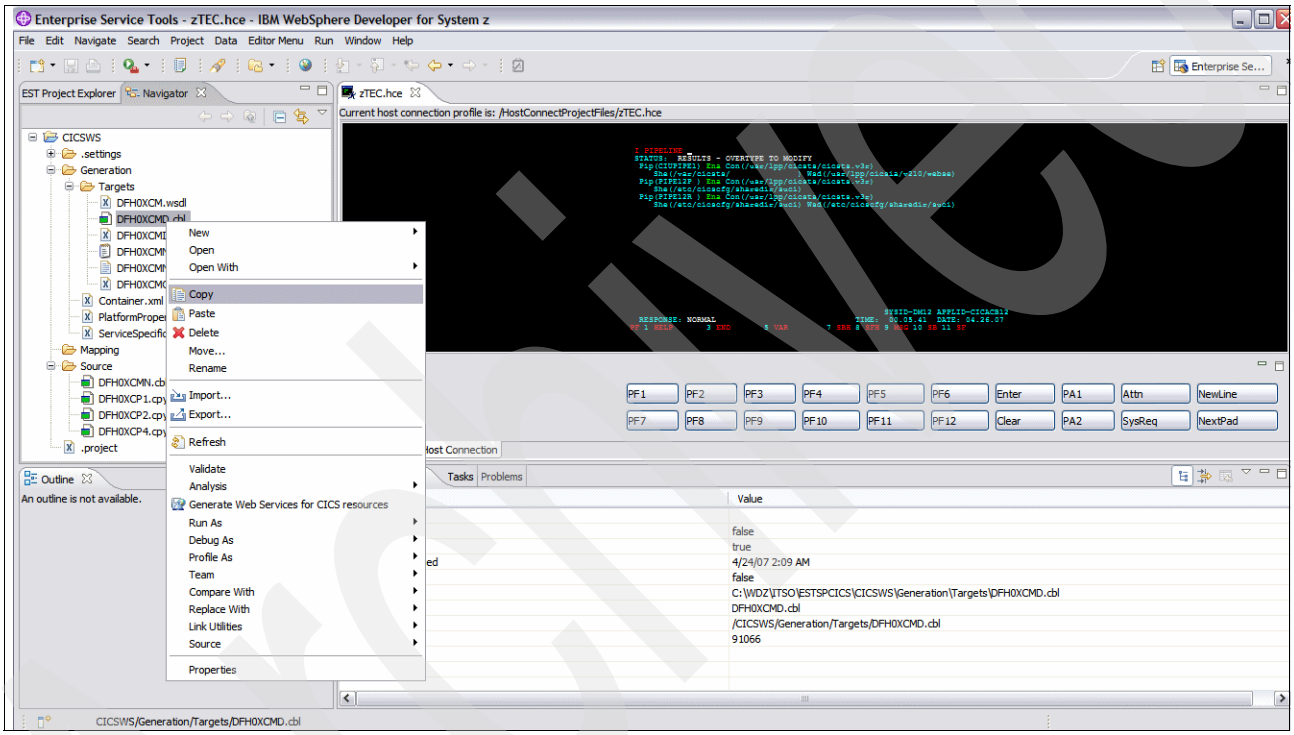


Figure 6-20 Copy the COBOL source file to z/OS system

2. To paste the program to your z/OS COBOL directory, switch to the z/OS perspective, expand the MVS files folder, right-click your source directory (<Userid>.WDZ.COBOL, in our example), and select **Paste**.

The file DFH0XCMD is copied to your z/OS source directory. See Figure 6-21.

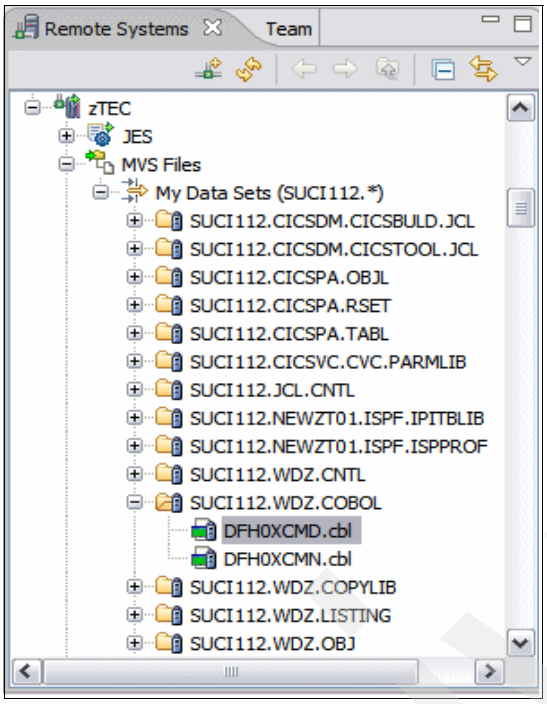


Figure 6-21 Converter source file pasted to your z/OS source directory

Compile your converter file

It is convenient to create a z/OS project and move the assets that you are working on to a subproject. This makes the task simpler because you have all the assets being used grouped in the same location.

Do these steps:

1. From the z/OS Projects view, right-click the blank area and select **New z/OS Project**.
2. Type EST_CONVERTER as the Project name, select **Create an MVS subproject**, and click **Finish**.
3. Type EST_CONVERTER_CICS as Subproject Name and click **Next**. Note that the High Level Qualifier will be your user ID. In our implementation, it is suci112. See Figure 6-22 on page 157.

Archived

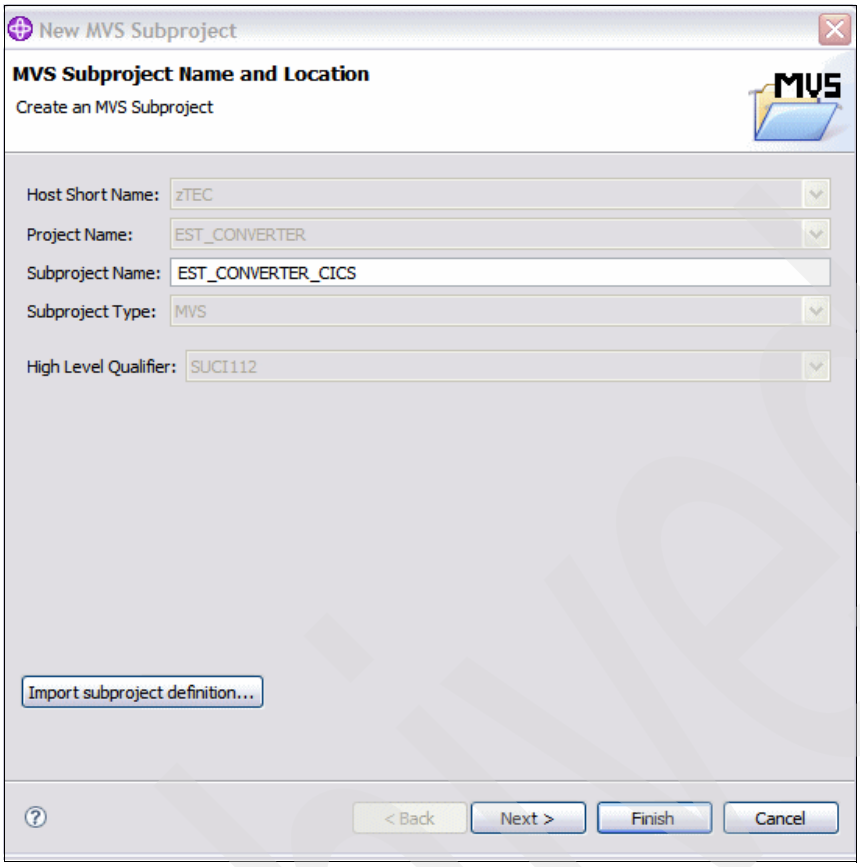


Figure 6-22 Create the Subproject for the XML converter program

4. Click **Next**.

Archive

5. Change the JCL Job Card to your own standards and specify the JCL data set loadlib where the generated JCL will be stored, as shown in Figure 6-23.

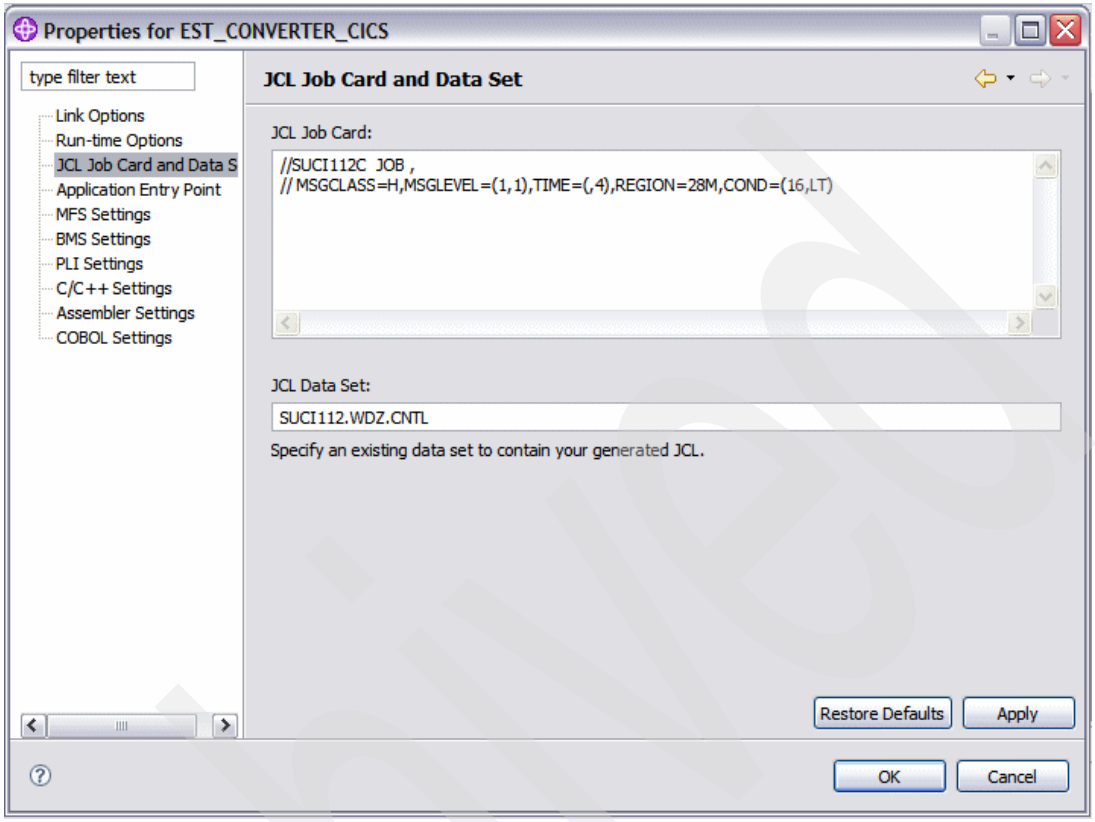


Figure 6-23 The JCL Job card and JCL data set

6. Click **Finish** to create the project and the subproject.

Now you have to add the program to the subproject:

1. From the Remote System view, expand your source files directory.
2. Select your converter program **DFH0XCMD.cbl**, right-click, and select **Add to Subproject**.
3. Select your project and subproject names that you just created and click **Finish**.
4. Switch to you z/OS project view. Your converter program has been added to your subproject. See Figure 6-24.

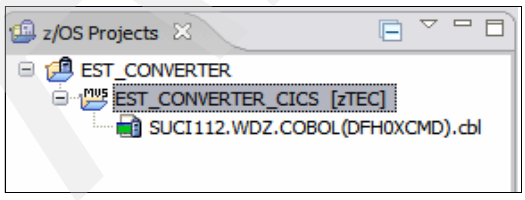


Figure 6-24 The converter subproject

You now have to generate the JCL to compile and linked it the converter program:

1. Right-click the program name and select **Generate JCL → For Compile and Link**.

   The generated JCL member is added to your subproject.

Your JCL should look like the one shown in Example 6-2.

Example 6-2 JCL generated to compile and link it to the Converter Program

```
//SUCI112C JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=28M,COND=(16,LT)
//*
//STP0000 EXEC PROC=ELAXFCOC,
// CICS=',''CICS LIB''',
// DB2=,
// COMP=
//COBOL.SYSPRINT DD DSN=SUCI112.WDZ.LISTING(DFHOXCMD),
//          DISP=SHR
//COBOL.SYSLIN DD DSN=SUCI112.WDZ.OBJ(DFHOXCMD),
//          DISP=SHR
//COBOL.SYSLIB DD DSN=SUCI112.WDZ.COPYLIB,DISP=SHR
//             DD DSN=CICSTS.V3R1.CICS.SDFHCOB,DISP=SHR
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN DD DSN=SUCI112.WDZ.COBOL(DFHOXCMD),DISP=SHR
//*
//******* ADDITIONAL JCL FOR COMPILE HERE ******
//LKED EXEC PROC=ELAXFLNK
//LINK.SYSLIB DD DSN=SUCI112.WDZ.OBJ,DISP=SHR
//             DD DSN=CEE.SCEELKED,DISP=SHR
//             DD DSN=DB2.V7R1M0.SDSNLOAD,DISP=SHR
//             DD DSN=CICSTS.V3R1.CICS.SDFHLOAD,DISP=SHR
//             DD DSN=USER.CICS.LOADLIB,DISP=SHR
//LINK.OBJ0000 DD DSN=SUCI112.WDZ.OBJ(DFHOXCMD),DISP=SHR
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DSN=CICSCFG.ZPARIS.LOADLIB(DFHOXCMD),DISP=SHR
//*
```

2. When the JCL is generated, right-click it and select **Submit for z/OS execution**.

Archived

3. To check the Job execution, from the remote system view in the JES folder, locate the job you just submitted. Double-click it. The output listing opens in the editor view, as shown in Figure 6-25.

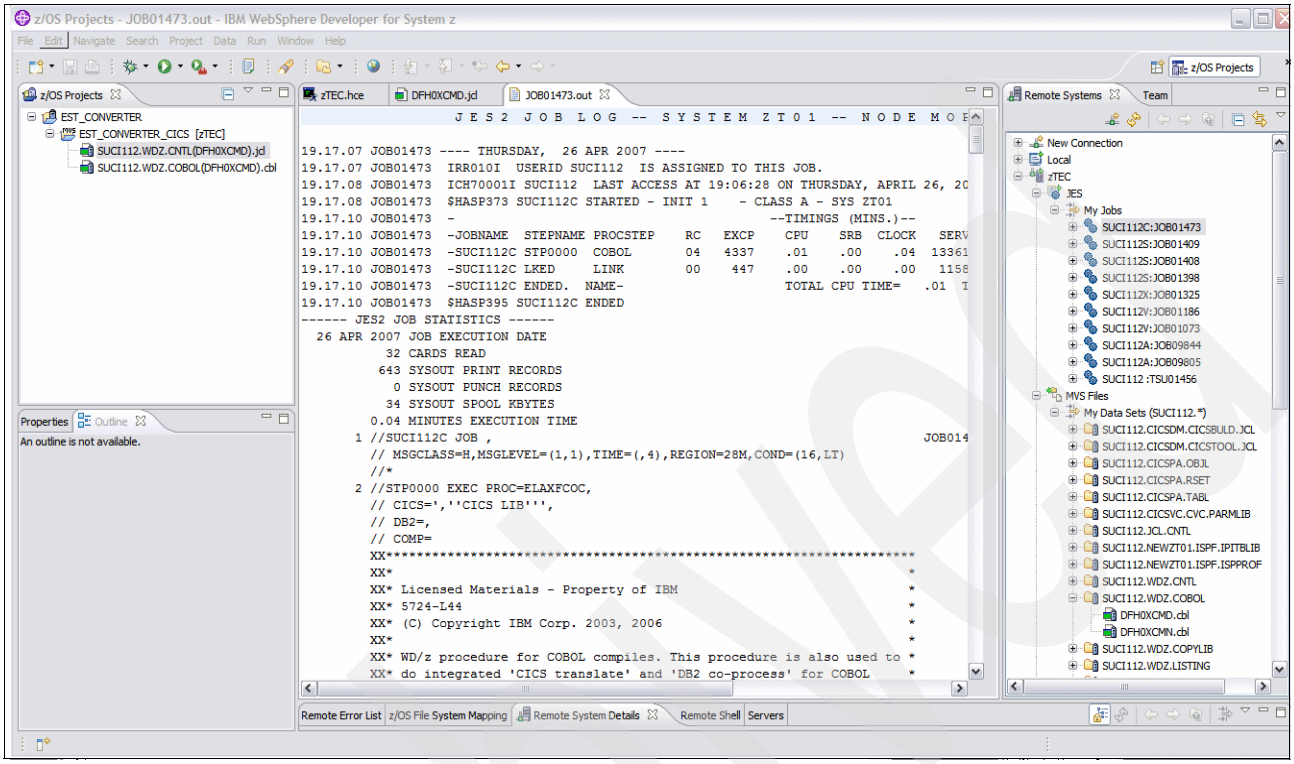


Figure 6-25 The output listing from the Compile and Link JCL submitted

Deploy the Web service provider

You have already copied DFH0XCMN.wsbind to the pickup directory (/etc/cicscfg/sharedir/wsdir in our implementation). When you install the pipeline or perform a scan, CICS will read the WSBind files in that pickup directory and automatically define the WEBSERVICE and URIMAP resources associated with those WSBind files.

As you have copied a new WSBind file to the pickup directory, a PIPELINE SCAN is required.

The PERFORM PIPELINE SCAN command initiates the scan of the PIPELINE's pickup directory.

Do these steps:

4. From the Remote System view, select your defined system connection. Right-click it and select **Host Connection Emulator support**. This opens the emulator in the Editor view. You can make the Editor view larger by clicking the **zTEC** tab. See Figure 6-26 on page 161.

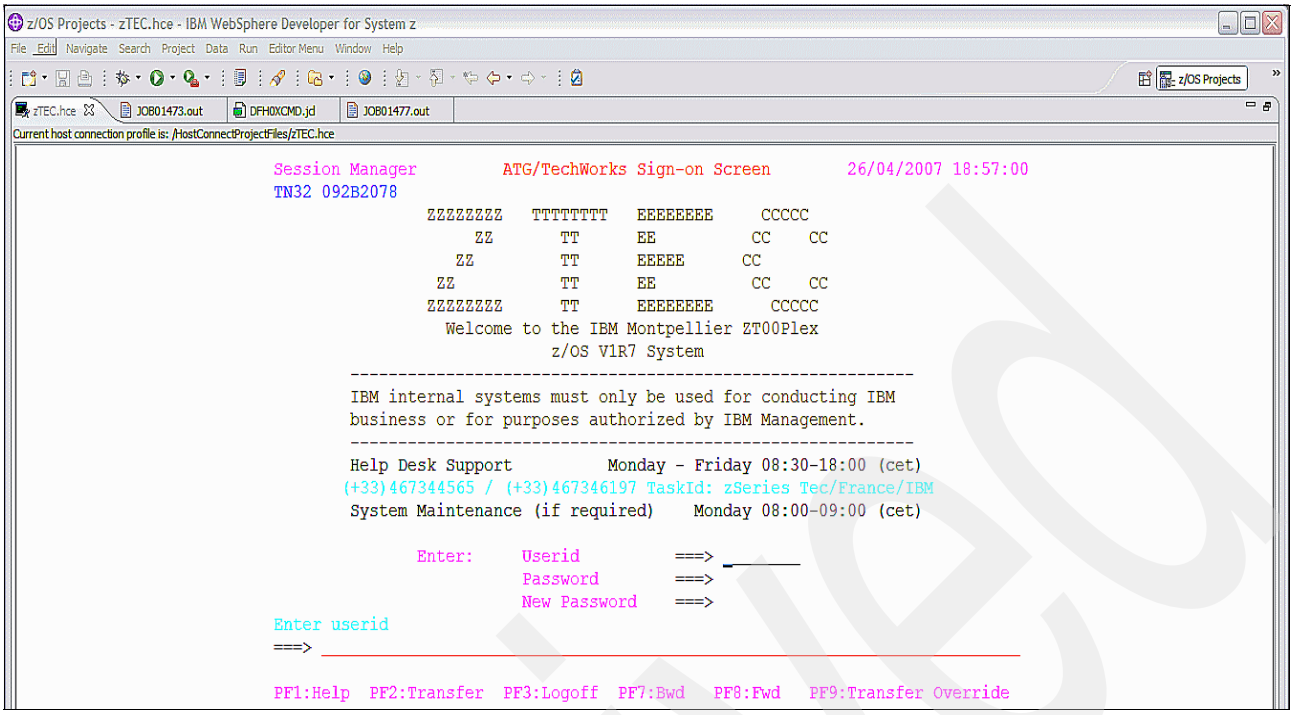


Figure 6-26 The connection emulator to access this CICS system.

5. Connect to your CICS.
6. Type `CEMT PERFORM PIPELINE(`*pipename*`) SCAN` on a clear window.

   You should see a message saying `RESPONSE:NORMAL`, as shown in Figure 6-27.



Figure 6-27 The pipeline scan result

Archived

7. To view the status of the Web service, type `CEMT INQUIRE WEBSERVICE` and press Enter. See Figure 6-28.

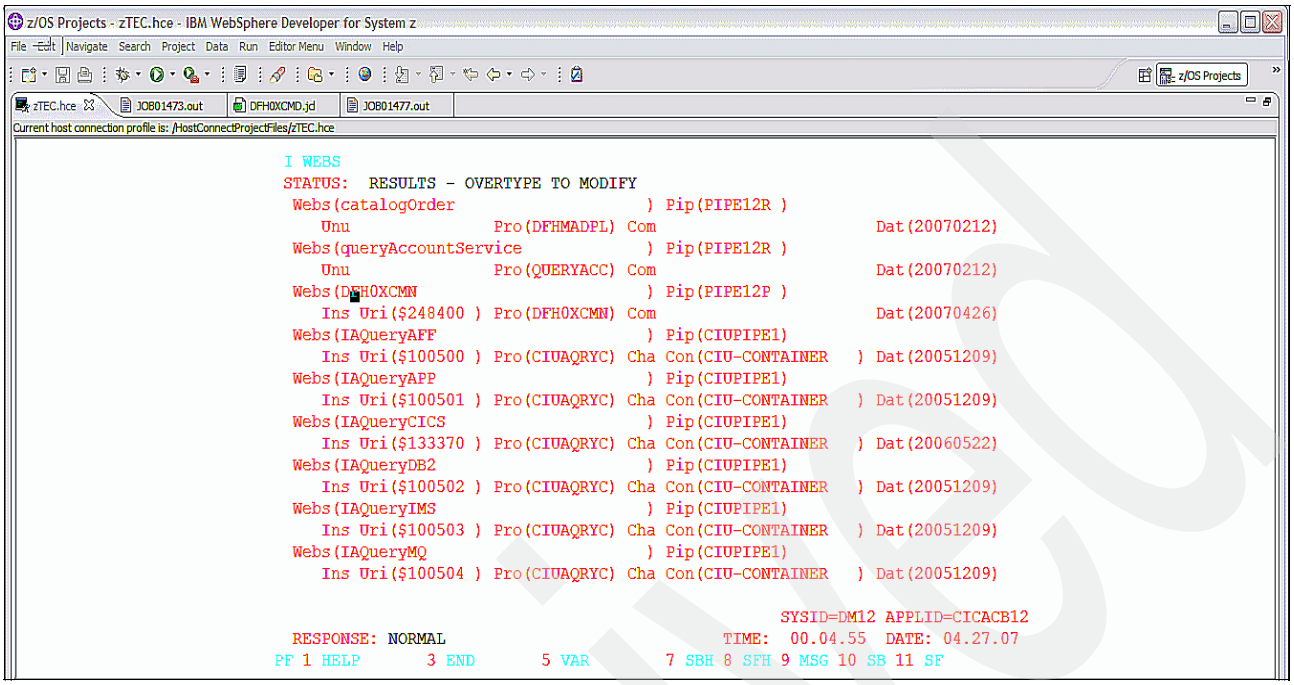


Figure 6-28 List of the Web Services deployed

8. To check the details of your Web service named DFH0XCMN, click its name. See Figure 6-29 on page 163.

Archive

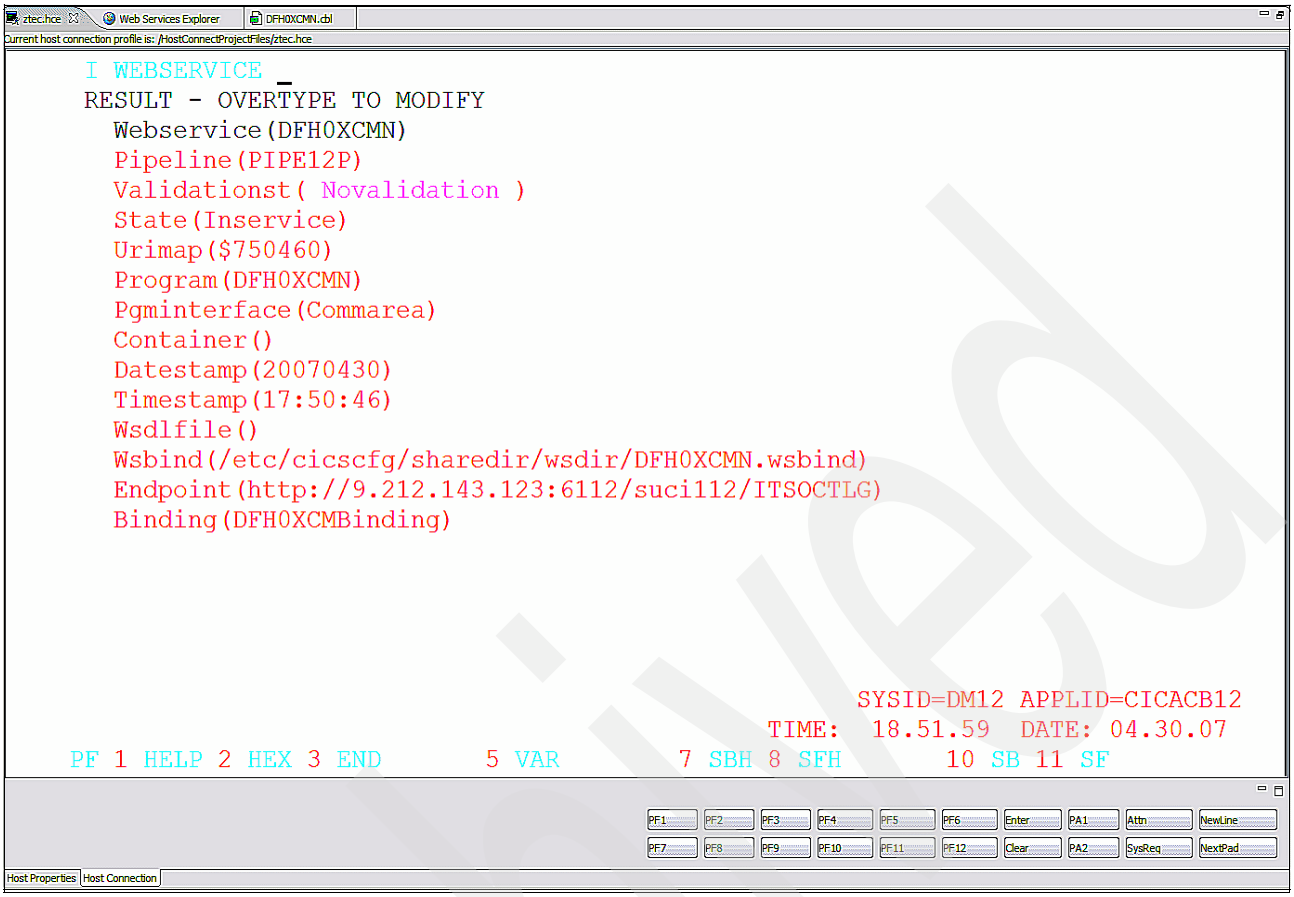


Figure 6-29 Details of your Web service

Your Web service is now ready to be tested.

Test your Web service using WebSphere Developer for System z

Do these steps:

1. Within WD/z, switch to the Enterprise Service tools perspective.
2. Select your generated WSDL file. Right-click it and select **Web Services** → **Test with Web Service Explorer**.
3. The Web Service Explorer view opens. Expand the **Binding** folder. Click the **Operation** folder.

Note: In the Actions pane, the Endpoints field should show the default endpoint. If you need to edit the endpoints, select **DFH0XCMNBinding** and use the Actions to add, change, or remove endpoints.

Archived

4. Type the required input values to invoke the Web service, as shown in Figure 6-30.

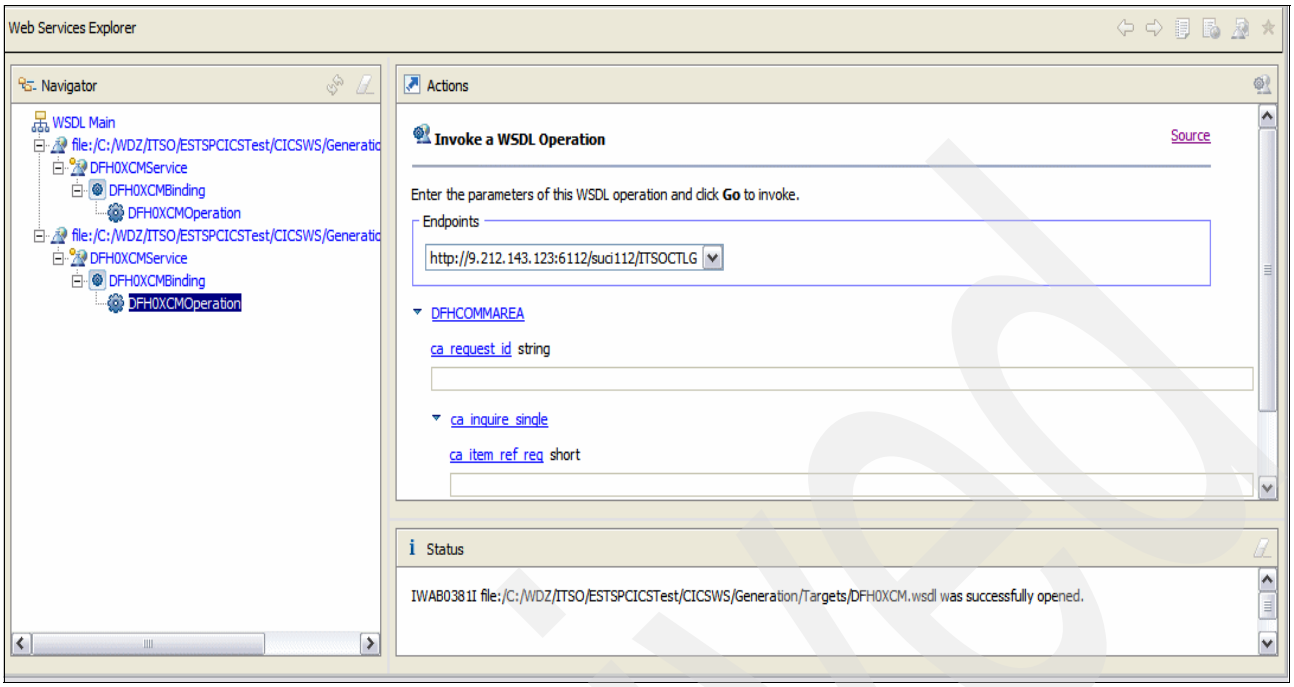


Figure 6-30 Test with Web Services Explorer

5. Click the **Go** button to invoke the Web service.

   The result of the invocation is the displayed in the status part of the window. You can double click the title **Status** to expand the window.

6. Click the word **Source** towards the top right in the Status window to see the detailed Web service (SOAP) request sent to CICS, and the response. You should see a window similar to Figure 6-31 on page 165.

Archive

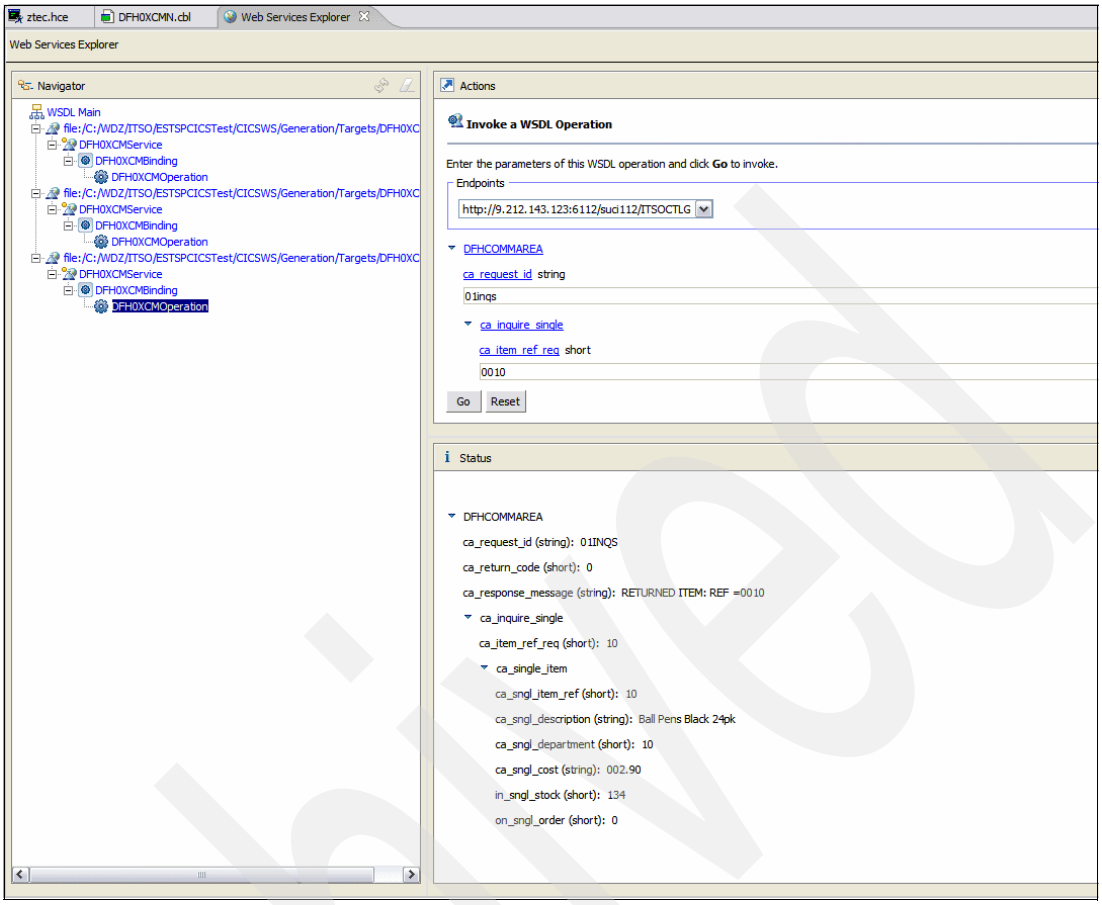


Figure 6-31 Results form testing the CICS Web service with Web Services Explorer

7. Double click the word **Status** to return to the previous Web Services Explorer view, where you can test your Web service with different values.

6.2 When to use interpretive versus programmatic generated adapters

The choice of conversion type involves considering the benefits and cost of either technology.

6.2.1 Compiled XML conversion

WebSphere Developer for System z has the capability to generate a driver and the converters program to transform the SOAP message into a COBOL data structure and vice versa.

The driver and the converters are often named the XML adapter. In order to simplify the CICS deployment, WD/z has the option to generate the driver and the two converters in the same program. Then, only one load module needs to be deployed under CICS.

The compiled XML conversion type provides more extensive support for language structure data types and constructs than the interpretive XML conversion type, but compilation of XML converters is required and there are more artifacts to manage.

Archived

Use the Compiled XML conversion when your CICS program's language structures are not interpretive, that is, mainly when "Redefines" or "Depending on" definitions are used, or when you need or prefer a readable conversion code.

As shown in Figure 6-32, the driver program interacts with the run time and invokes bundled converters code to provide conversion of XML to and from language structures.

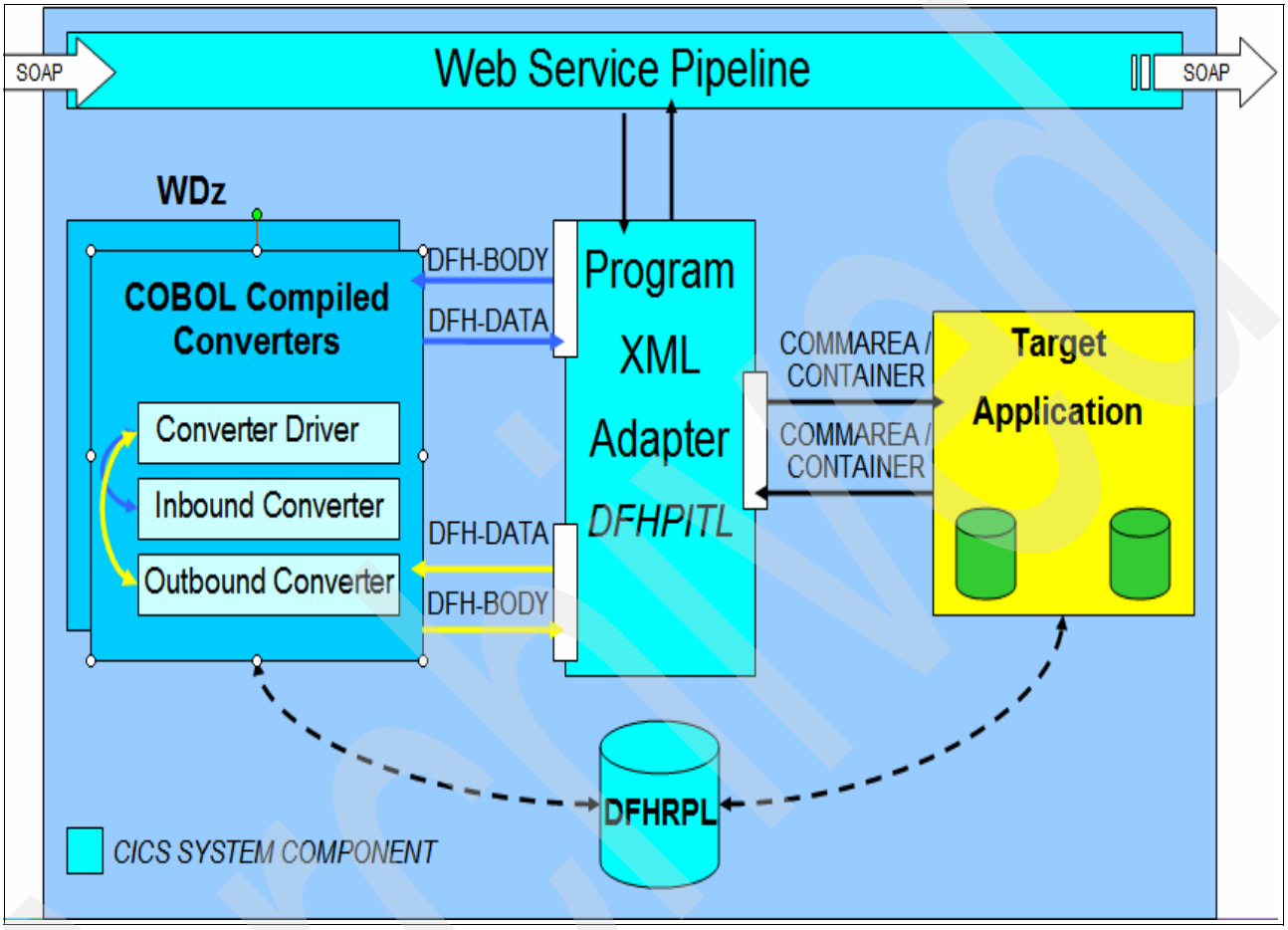


Figure 6-32 Compiled XML conversion process within CICSTS V3.1

6.2.2 Interpretive XML conversion

While the Interpretive XML Conversion type has limited support for language structure data types and constructs, it requires fewer artifacts and does not involve compilation of XML converters.

Use the Interpretive XML Conversion when your CICS program has simple language structures.

There is no need for compilation or source code management.

As shown in Figure 6-33 on page 167, the XML conversion is accomplished using generated meta data and an XML conversion component built into the run time.

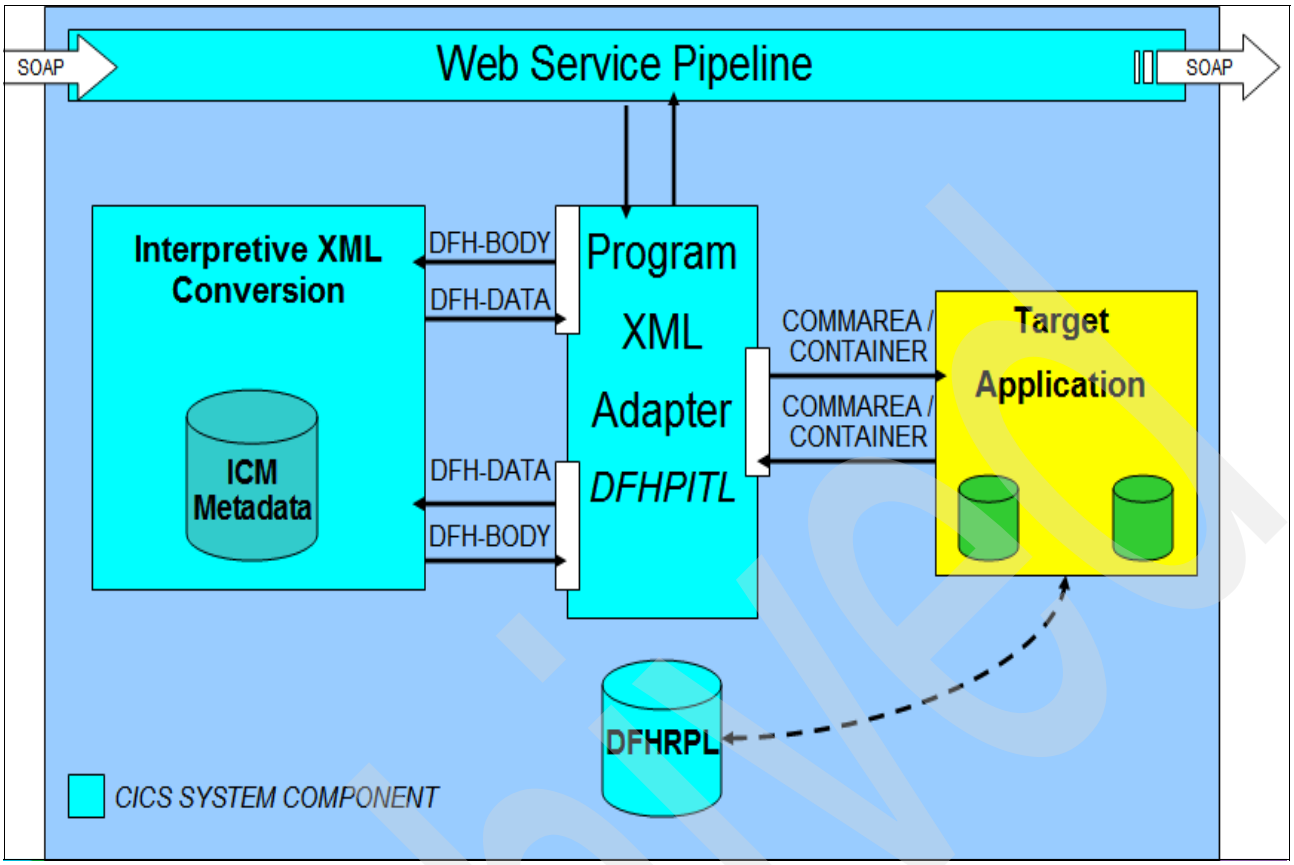


Figure 6-33 Interpretive XML conversion process within CICSTS V3.1

6.3 Batch processor

The batch processor is a command-line interface for creating enterprise Web Services descriptions and message converters for CICS and IMS applications.

This tool is useful if you need to create many services from many COBOL source files.

The Web Services artifacts generated from the command-line interface are identical to those generated in the EST single-service projects that we have already seen, such as:

- PGMNAME.wsdl: The WSDL file that describes the CICS based Web service.
- PGMNAMED.cbl: The conversion program which consists of the converter driver and input and output converters.
- PGMNAMI.xsd and PGMNAMO.xsd: The XML Schema Definition files that describe the input and output XML documents respectively.
- The *.log file that is the generation log.
- PGMNAME.wsbind: The WSBind file that tells CICS how conversion is to be performed and what the URI map looks like.

Archive

Figure 6-34 shows the generated artifacts.

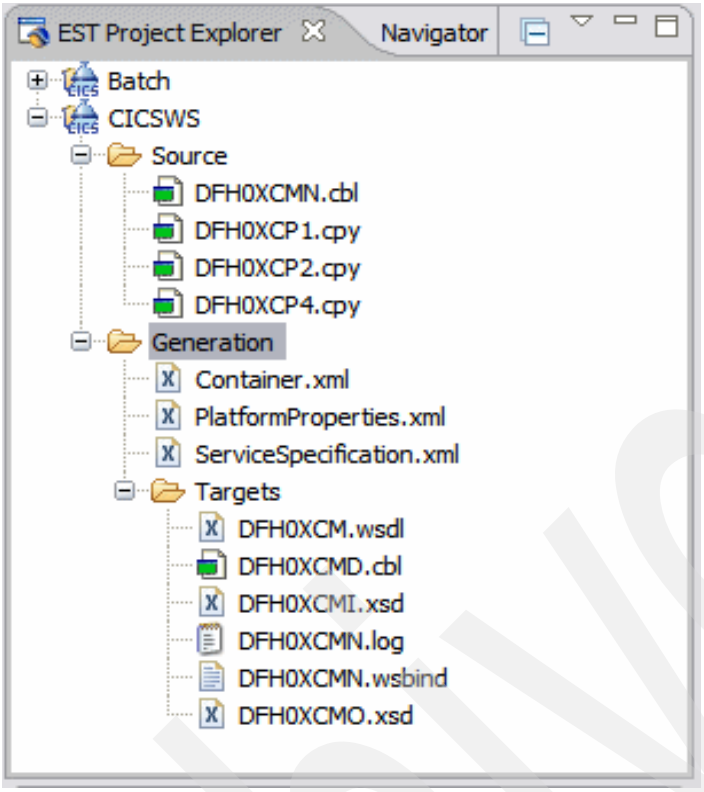


Figure 6-34 The WD/z Enterprise Service Tools generated artifacts for CICSTS V3.1

Note: The commands from the batch processor can also be stored in a script file in order to run numerous enablements at the same time and in order to schedule the run when needed.

6.3.1 Specify the options for the batch processor

You have three files in which to provide the necessary options to the batch processor:

- Container.xml
- PlatformProperties.xml
- ServiceSpecification.xml

Specifying most of the options in XML files allows automation of the options generation by WD/z service generation wizards.

PlatformProperties.xml file

This file sets the default options properties that reflect your target runtime environment.

The options affect, for example, the processing of the language types (such as COBOL data types) that are used in producing XML schema descriptions of Web service messages that are based on that language type. See Figure 6-35 on page 169.

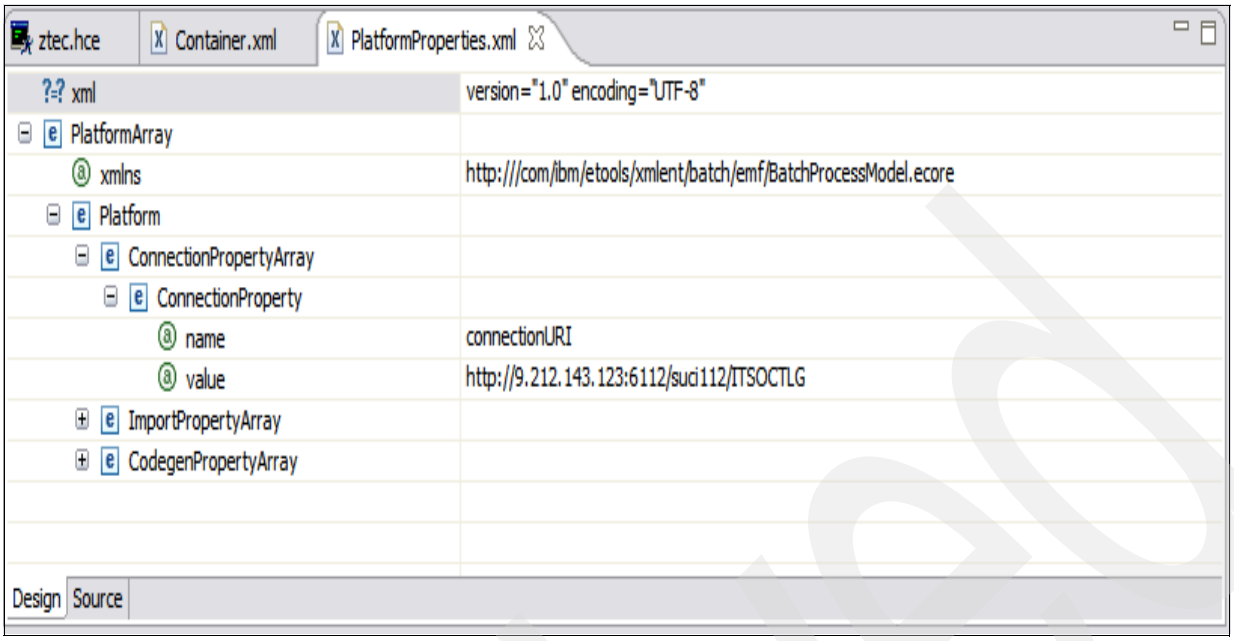


Figure 6-35 The PlatformProperties.xml File

ServiceSpecification.xml file

Using the ServiceSpecification.xml file, you can specify the options required to generate new Web service interfaces or Web service implementations.

The ServiceSpecification.xml allows you to create:

- A Web service interface for existing CICS and IMS applications. The generated artifacts include a Web service description and message converters.
- A new CICS Web service provider application from a Web service description. The generated artifacts include language structures, message converters, and an application template.
- A new CICS Web service requestor implementation from a Web service description. The generated artifacts include language structures, message converters, and an application template.

Archived

In the Service Specification.xml file, you can also override certain options that you specify in the PlatformProperties.xml file. See Figure 6-36.

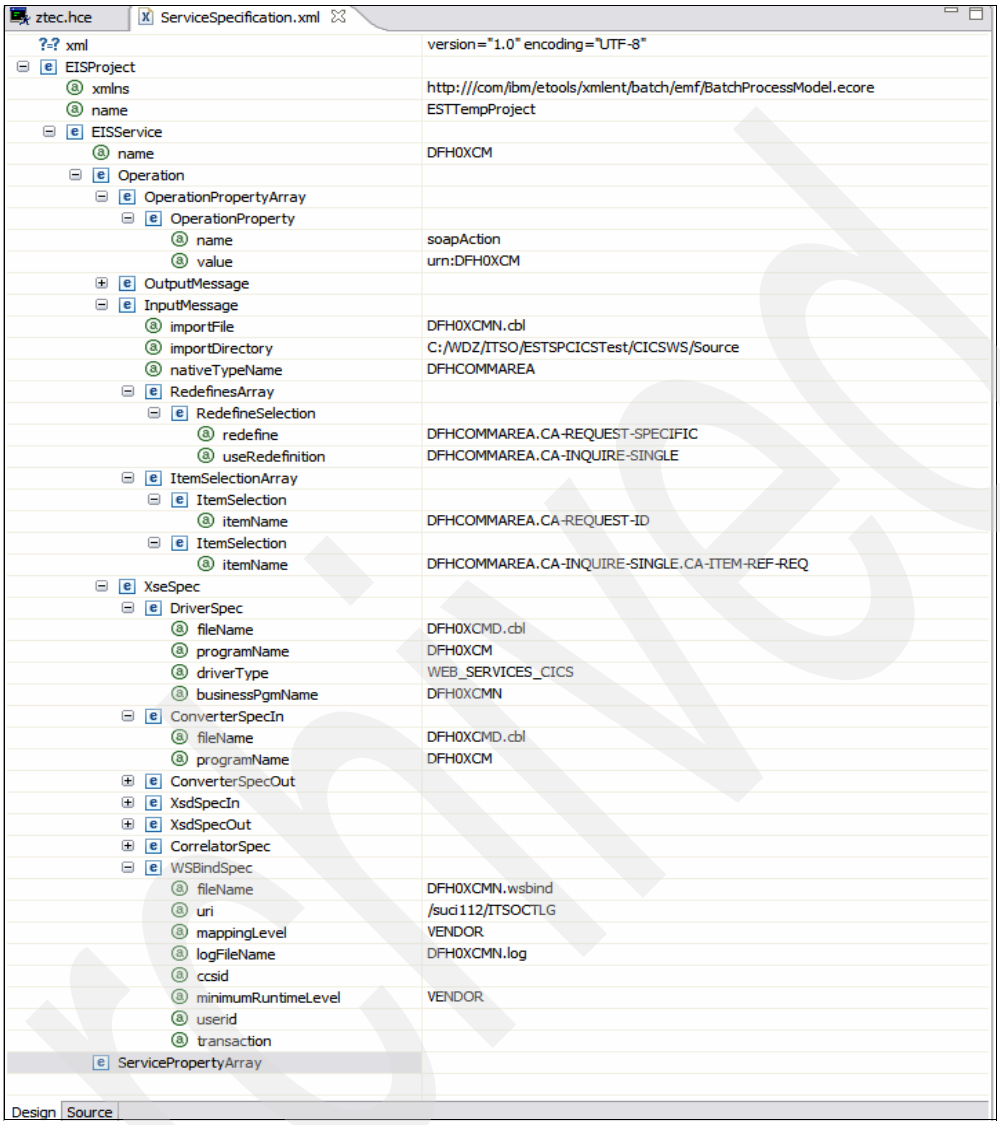


Figure 6-36 The ServiceSpecification.xml file

Container.xml file

This file contains the generation options and references to the Platform.xml and ServiceSpecification.xml file.

The GenerationSpec file provides information about the ServiceSpecification.xml file, which specifies the options for generating individual sets of converters and Web Service Definition Language (WSDL) files.

The GenerationSpecArray provide information that globally affects the generation of converters and service definitions. It is defined using the attributes shown in the Properties pane in Figure 6-37.

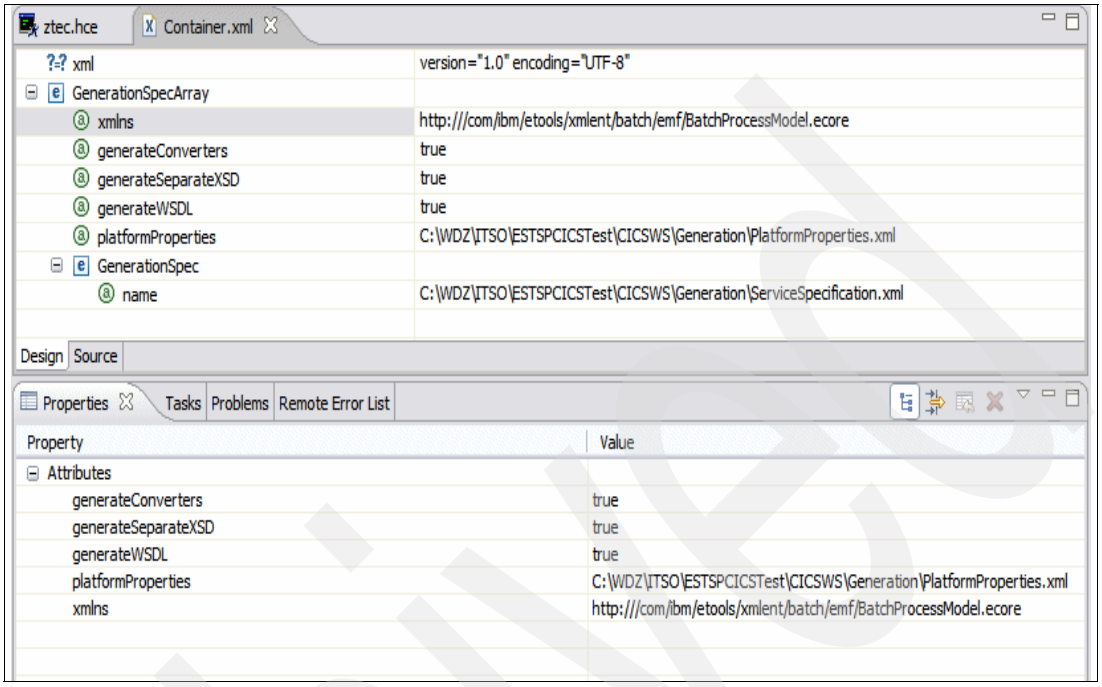


Figure 6-37 The Container.xml file

Archived

Using the WebSphere Developer for System z Copy and Paste facility, select the .xml files from the EST perspective, switch to the z/OS project perspective, and paste the files into a new directory, as shown in Figure 6-38.

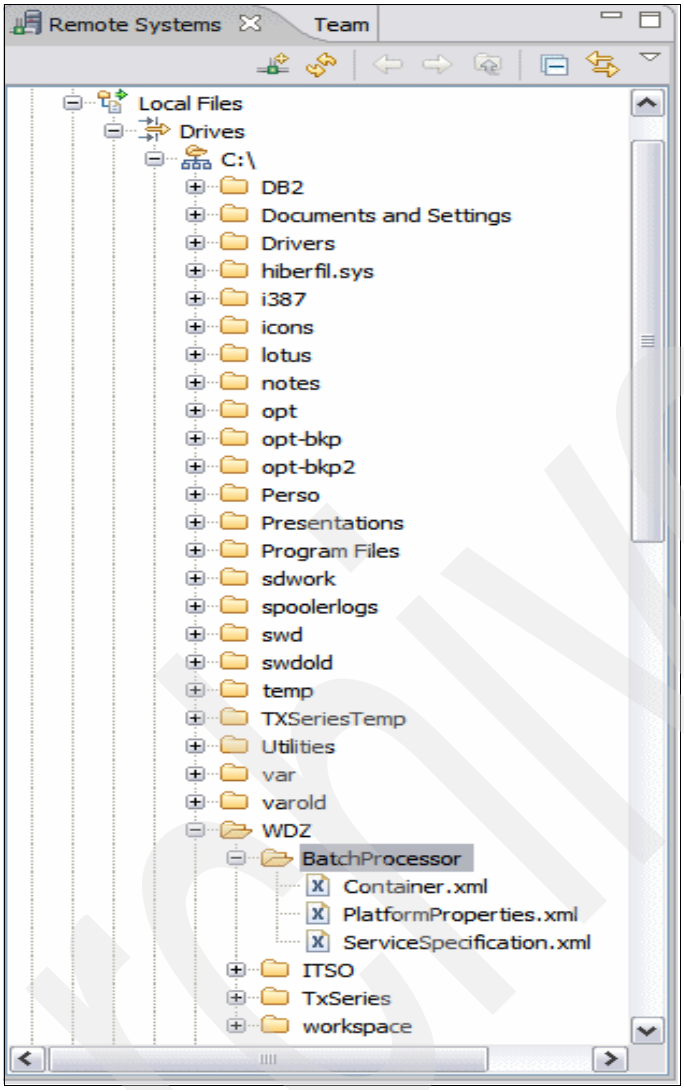


Figure 6-38 Copy the .xml files to a local directory

Change the XML files to, for example, generate files with a different name. Double-click the ServiceSpecification.xml file to open it with the XML editor. The editor window opens and you see an Outline view of the .xml file that gives you a quick view of the values. Change the required name to the new value, as shown in Figure 6-39 on page 173.

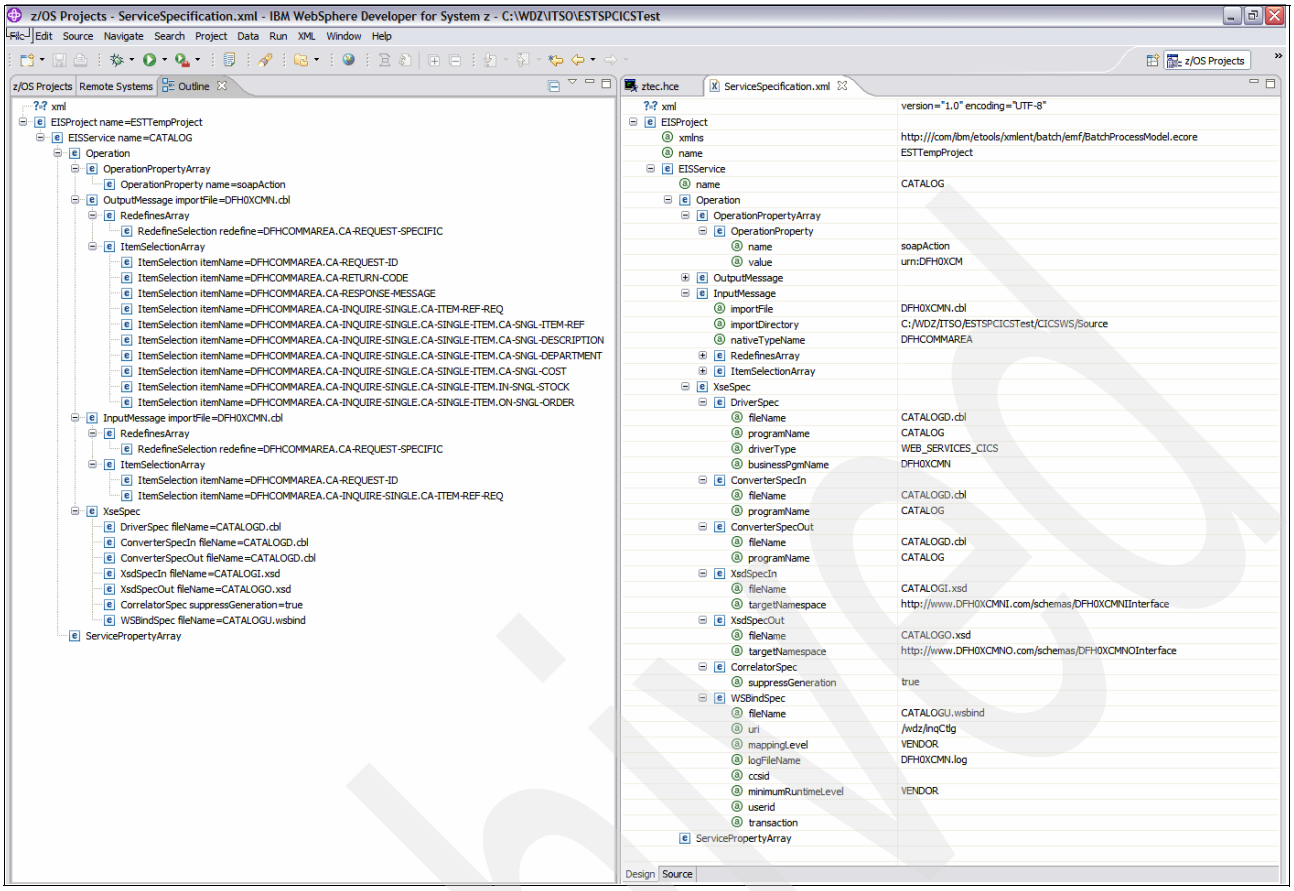


Figure 6-39 Modify the ServiceSpecification.XML file

By editing the ServiceSpecification .xml file and with the help of the Outline view, you can easily modify the structure of the Input and Output message expected and produced by the CICS program to transform.

You may also change the URI value by modifying the PlatformProperties.xml file, as shown in Figure 6-40.

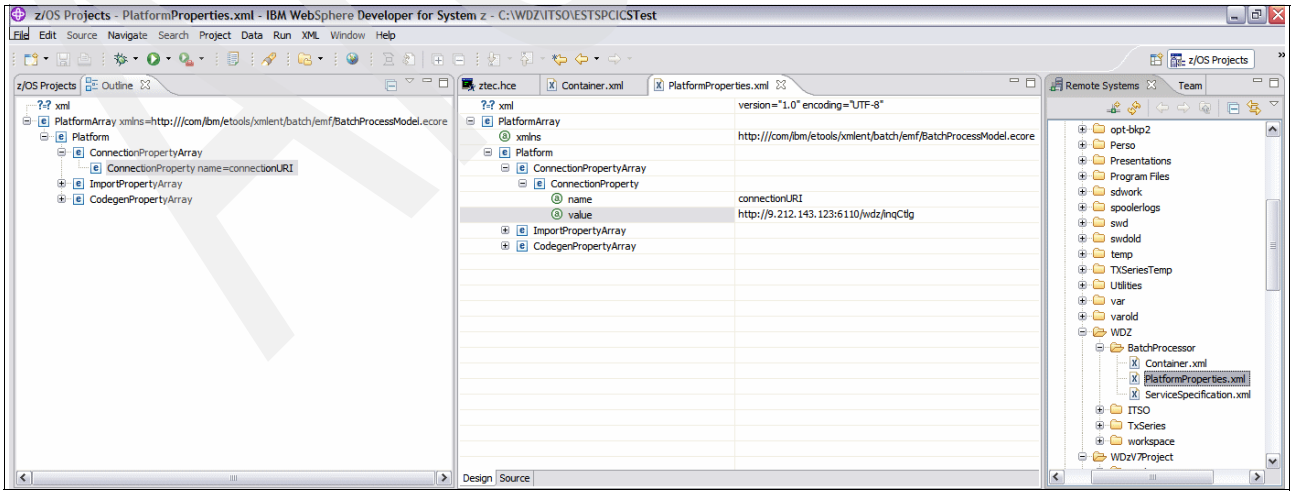


Figure 6-40 Modify the PlatformProperties.xml file

You have to modify the container.xml file to make it reference the new PlatformProperties.xml and ServiceSpecification.xml file locations, C:\WDZ\BatchProcessor\PlatformProperties.xml and C:\WDZ\BatchProcessor\ServiceSpecification.xml, in our implementation, respectively, as shown in Figure 6-41.

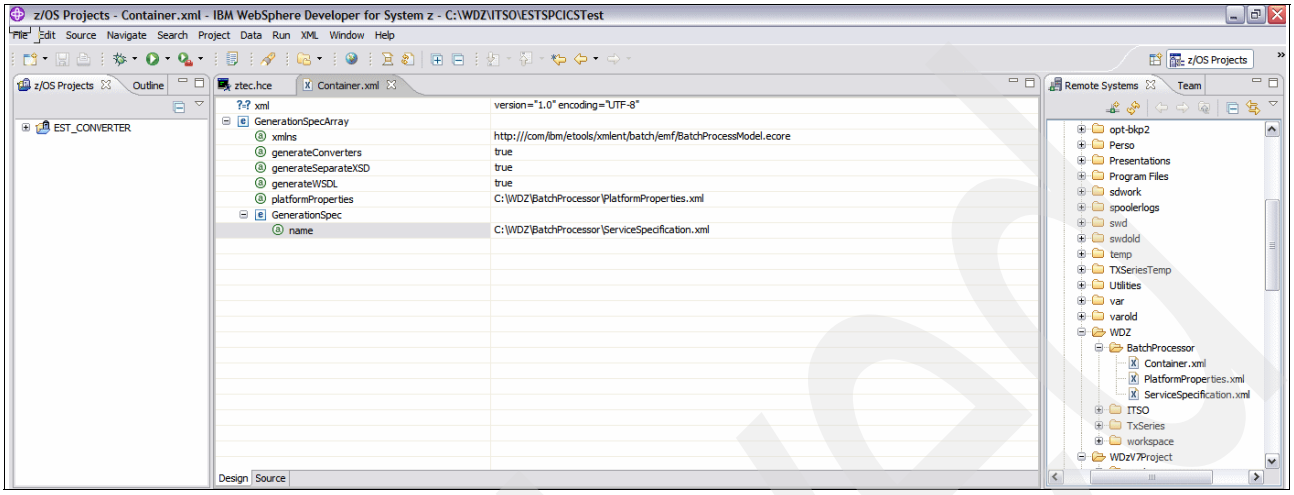


Figure 6-41 Modify the container.xml file

Run the batch processor

The batch processor can be started using either the command line or a script file.

Let us start the batch processor using the command line.

Note: You have to close any WebSphere Developer instance that uses the target folder before running the batch processor;

Do these steps:

1. Select **Start** → **All Programs** → **IBM Software Development Platform** → **IBM WebSphere Developer for System z** → **Command Environment for local compiler**.

   This opens a command prompt.

Archive

2. Switch to the wdz bin directory and type the following command, for our implementation:

   ```
   xsebatch -f c:\wdz\batchprocessor\container.xml -d c:\wdz\batchprocessor
   ```

   You should see a command prompt window similar to Figure 6-42.



Figure 6-42 Run the xsebatch command

Archived

3. Now, start the WD/z workspace, named c:\wdz\batchprocessor in our implementation.

The **xsebatch** command has generated the XML enablement files in the directory specified.

You can look at it in the Local directory from the Remote System view. See Figure 6-43.

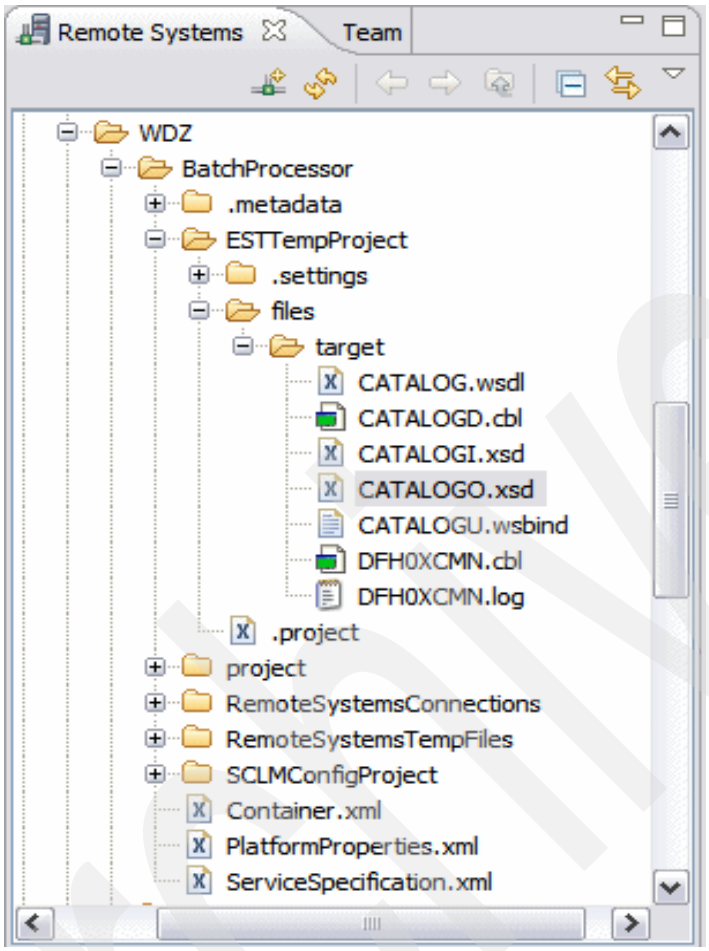


Figure 6-43 The XML enablement files generated

You can see your new EST project generated by switching to an EST perspective, as shown in Figure 6-44 on page 177.

Archived

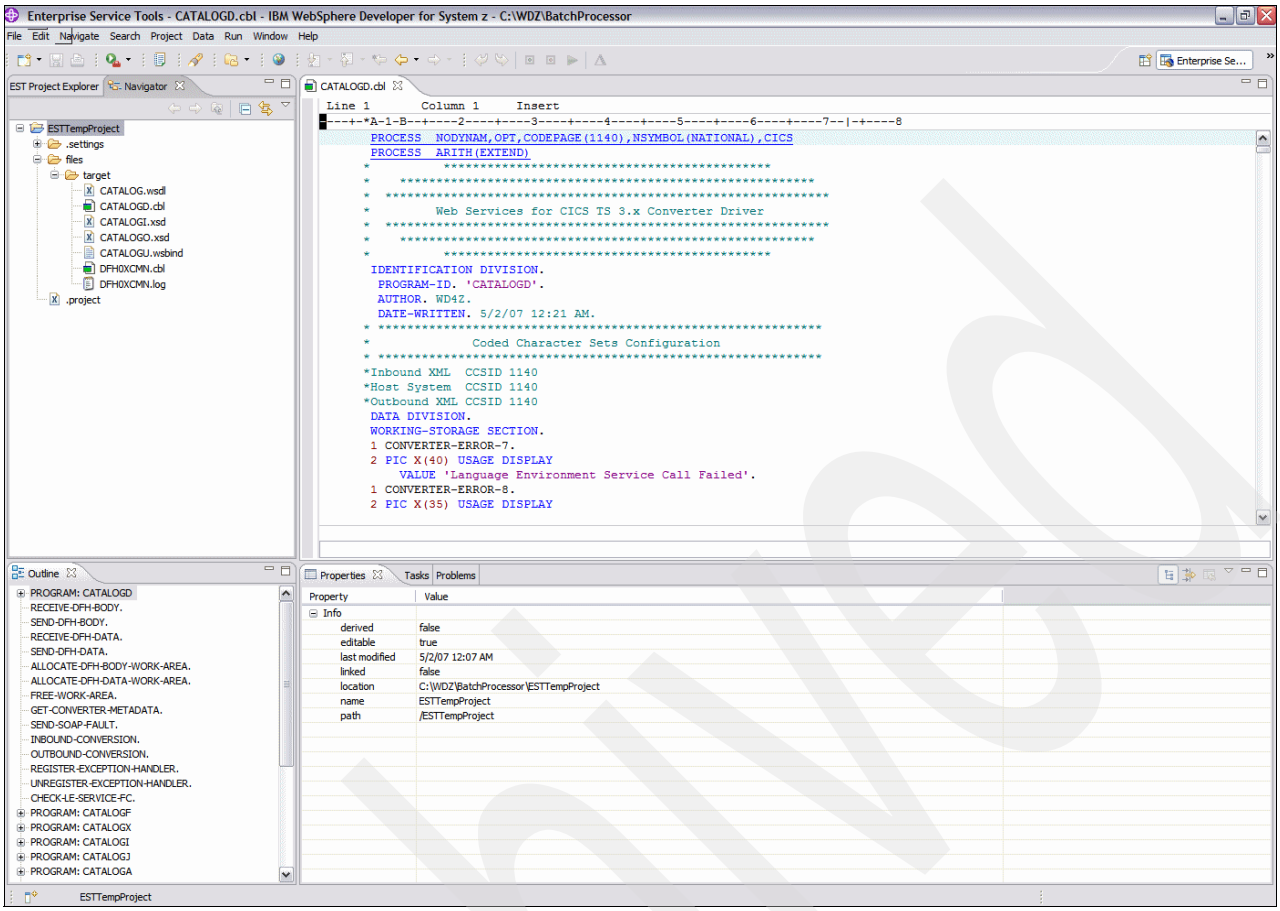


Figure 6-44 The new generated EST project

The new EST project is now ready to be deployed.

Using a script file to store and run the **xsebatch** command is a convenient way to handle a large number of Web service implementations.

Archived

Archived

7

Web service enablement for an existing IMS application

This chapter provides a reference to assist practitioners in enabling an IMS application as a Web service under IMS Transaction Manager.

IMS provides a variety of solutions for enabling access to IMS applications and data as Web Services using:

- Host Access Transformation Services (HATS) Web Services
- JCA Connector: IMS Connect / IMS Connector for Java
- JMS Connector: MQ to IMS Bridge
- SOAP Access: IMS SOAP Gateway

In this chapter, we describe the different steps to enable an IMS transaction as a Web service using SOAP Access, that, is the IMS SOAP Gateway.

We will use IMS Soap Gateway V9.2.1.

Note: You can download the SG247482-Chapter 7.zip file by referring to Appendix B, “Additional material” on page 645.

© Copyright IBM Corp. 2009. All rights reserved.

7.1 What is the IMS Soap Gateway

The IMS SOAP Gateway is a SOAP engine that builds on top of the Apache Axis Server to allow the exposure of an IMS application as Web Services. The Apache Axis Server provides the underlying implementation for handling SOAP messages, running a Web Services application as well as tools for deploying WSDL files and monitoring SOAP messages.

The IMS SOAP Gateway extends the functionality of the Apache Axis Server to allow a client application to invoke an IMS transaction using the XML and SOAP message protocol with IMS Connect. See Figure 7-1.

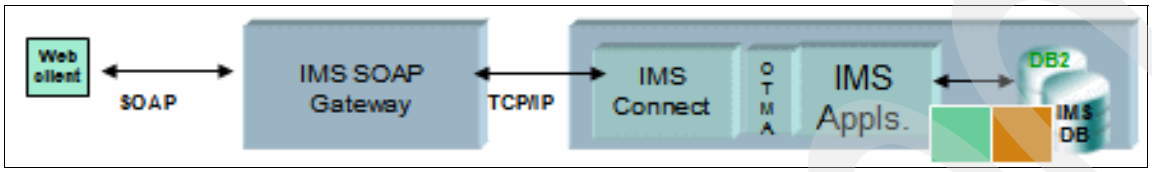


Figure 7-1 Soap access: IMS Soap Gateway

The IMS Soap Gateway is a solution that leverages existing IMS applications as Web Services without the need of a J2EE server.

It makes your IMS application a Web service with easy deployment and configuration; there is no programming needed.

IMS SOAP Gateway consists of two main components:

IMS SOAP Gateway deployment utility
This end-to-end deployment utility enables you to set up properties and create runtime code that IMS SOAP Gateway uses to enable IMS applications as Web services.

IMS SOAP Gateway server
The IMS SOAP Gateway server processes SOAP messages. It receives the SOAP message from the client application, converts it to an IMS input message, and sends it to IMS through IMS Connect. It then receives the output message from IMS and converts it to a SOAP message to send back to the client.

Note: You can download for the IMS Soap Gateway and get the latest information about it from the IBM Web site found at:

`http://www.ibm.com/software/data/ims/soap/`

With WebSphere Developer for System z, we will see how to transform XML data using the converters generated, eliminating the need to modify the IMS application code.

This is a key point: The IMS application is unchanged in order to run exactly the same way with other existing applications and data as before the enablement.

This requires IMS V9.1 level or later with integrated IMS Connect + XML Adapter.

Archived

7.2 How to transform your IMS Application in a Web service

Here is an overview about how WebSphere developer for System z tools provide the artifacts to enable the IMS Transaction as a Web service.

Figure 7-2 describes what you generate using WD/z and what will be deployed under IMS. It then shows the execution process when the deployed IMS Web service is invoked by a Web client.

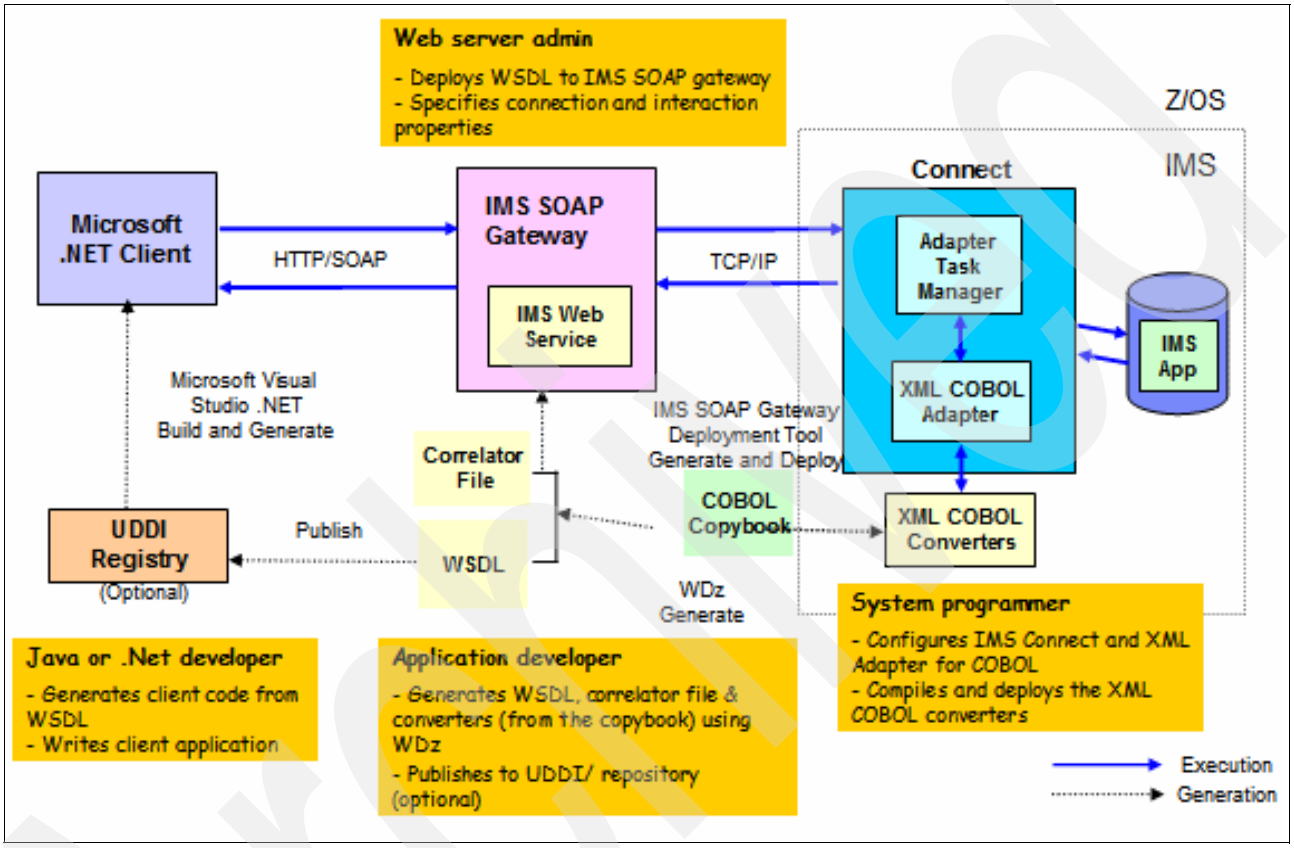


Figure 7-2 IMS V9.1 - Web Services support overview

The XML transformation is handled by IMS Connect XML Adapter and WD/z converters. The XML Adapter code is included in IMS Connect and supports COBOL only.

WebSphere Developer for System z takes the COBOL copybook of an IMS application as input and generates WSDL, correlator, and XML Converters files.

This generation process is what is described in this chapter.

To enable your IMS application as a Web service using WebSphere Developer for System z, you will:

- Generate the XML Converters and the WSDL and correlator files.
- Deploy the WSDL and correlator (.xml) files to IMS SOAP Gateway.
- Compile the XML Converter and link it to the IMS Connect STEPLIB data set.

Let us start with the generation process.

7.2.1 Generation of the Web enablement components

Let us create an Enterprise Service Tools project to handle the Web service enablement components. Do these steps:

1. Start WebSphere Developer for System z with a new workspace.
2. Select **File** → **New** → **Project**. Expand the Enterprise Service Tools project and select **IMS SOAP Gateway Project**, as shown in Figure 7-3.

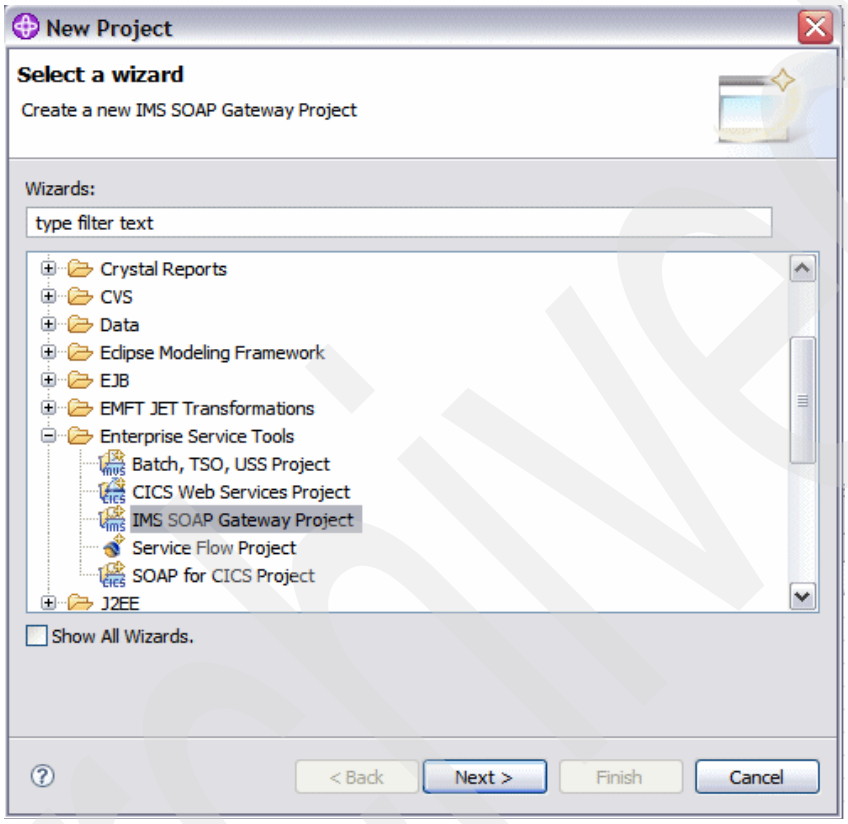


Figure 7-3 EST IMS SOAP Gateway Project selection

3. Give a name to your project, as shown in Figure 7-4 on page 183.

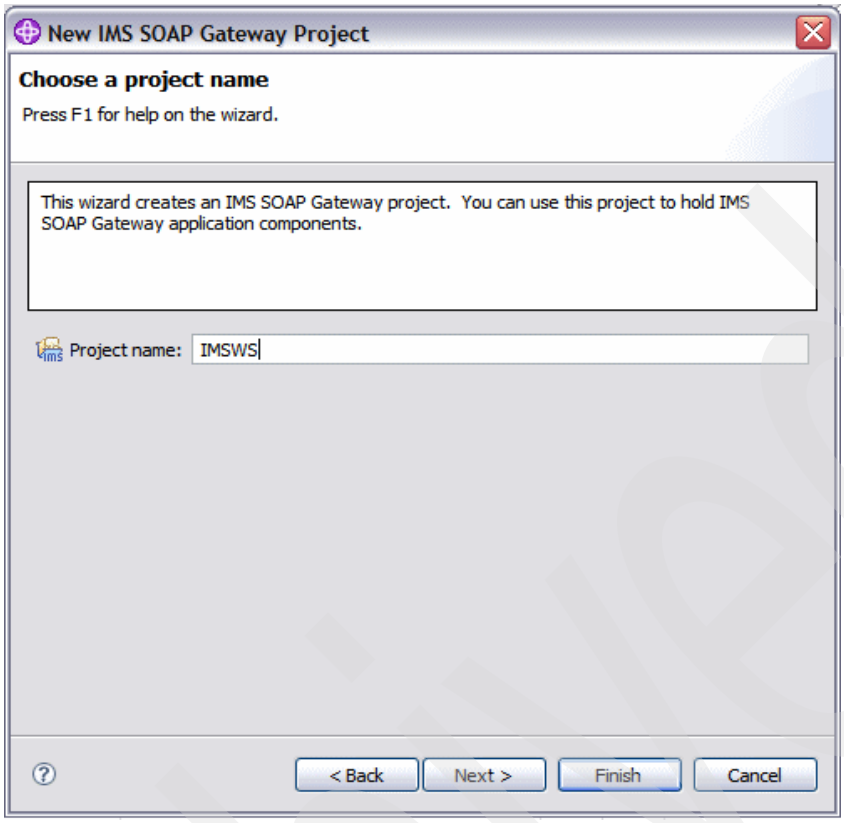


Figure 7-4 Give a name to you IMS single service enablement project

4. Click **Next**.
5. In the new project created, you have to import the program that you want to enabled as a Web Service or the copybook that describes the input and output message format used by your program. In our implementation, we import the Copybook used by the Telephone Directory IVP application :IMSPHBK.cpy.

Archive

In the Import Source File window, click **File System** and locate the copybook to import. See Figure 7-5.

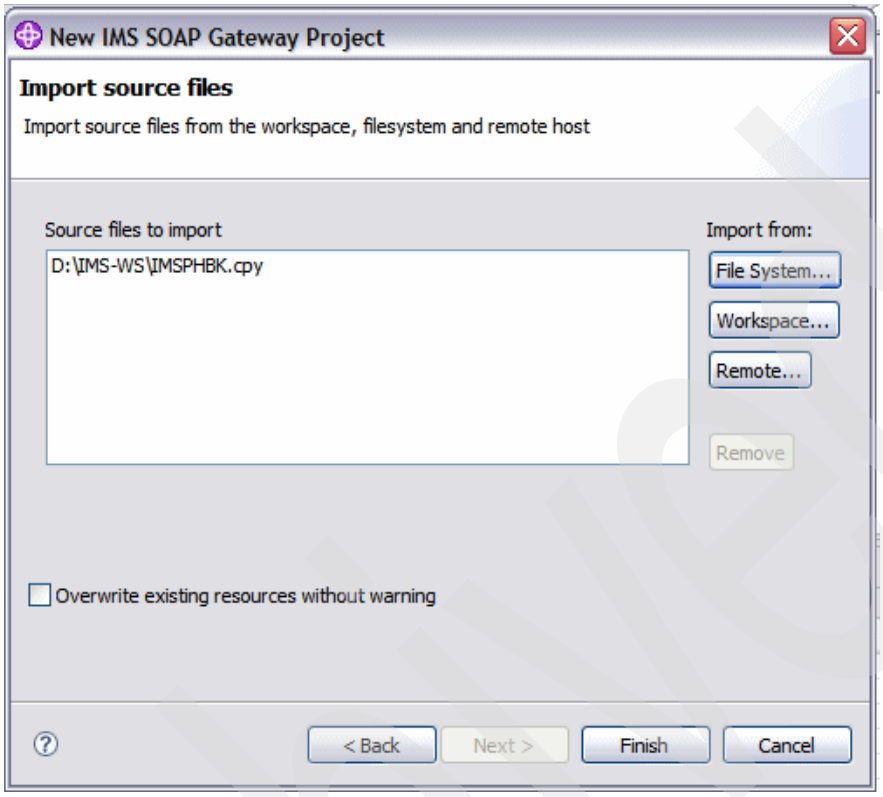


Figure 7-5 Import the telephone directory copybook in your project

6. Click **Finish**.
7. From the EST project explorer view, in the source folder, double click **IMSPHBK.cpy** to open it in the editor view. You can see what is expected as the input of the transaction and what is provided as the output, as shown in Figure 7-6 on page 185.

Archive

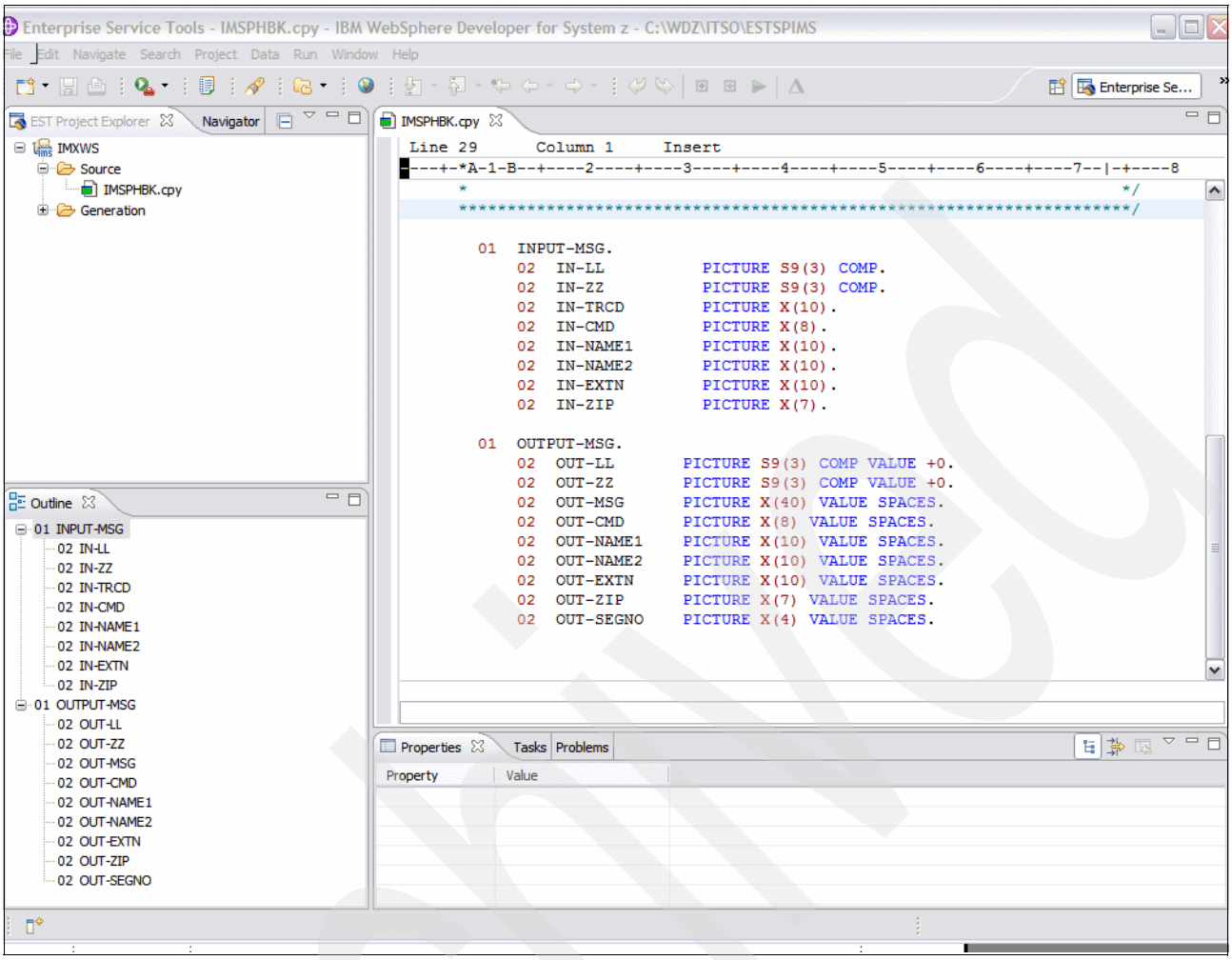


Figure 7-6 The Enterprise Service Tool perspective shows your new project and the copybook IMSPHBK.cpy

8. From the EST project Explorer view, select your copybook, **IMSPHBK.cpy**, and from the contextual menu, select **Generate IMS SOAP Gateway resources**.

   In this context, only one scenario is available, the Bottom Up one, because we have a COBOL artifact as the input to generate all the artifacts to enable it as a Web service.

   The conversion type is set to Compiled XML conversion, as we use the adapter generated by WD/z to do the XML conversion.

Figure 7-7 shows the EST Wizard Launchpad.

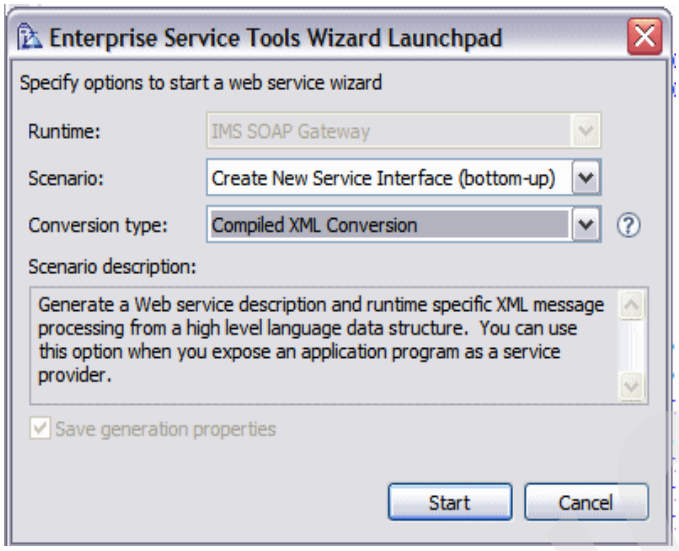


Figure 7-7 The EST Wizard Launchpad

Note: The **Save generation properties** button is already checked, which will keep a record of the generation options and language structure selections that you will specify for regeneration, audit, or debugging purposes.

9. Click **Start** to start the Web service for the IMS Soap Gateway wizard.
10. In the Language structure window, for the inbound data structure, select the COBOL data structure that corresponds to the input message of the IMS Application, as shown in Figure 7-8 on page 187.

Archived

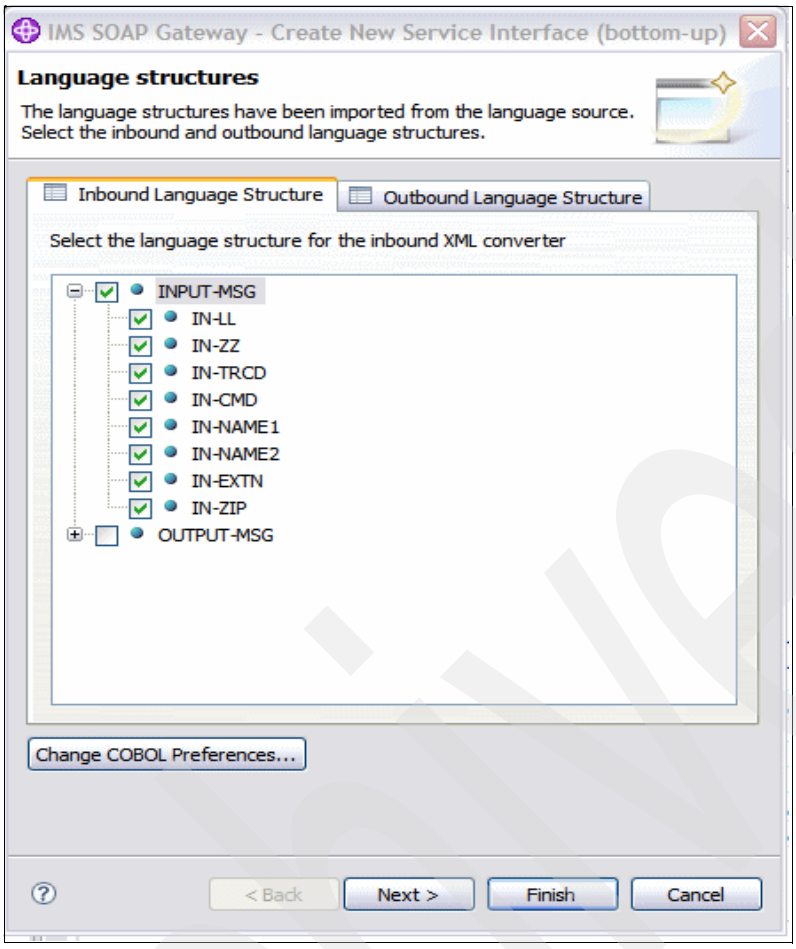


Figure 7-8 Inbound language data structure selection

11. Select the Outbound data structure tab.

Archive

12. Select the COBOL data structure that corresponds to the output message of the IMS application, as shown in Figure 7-9.

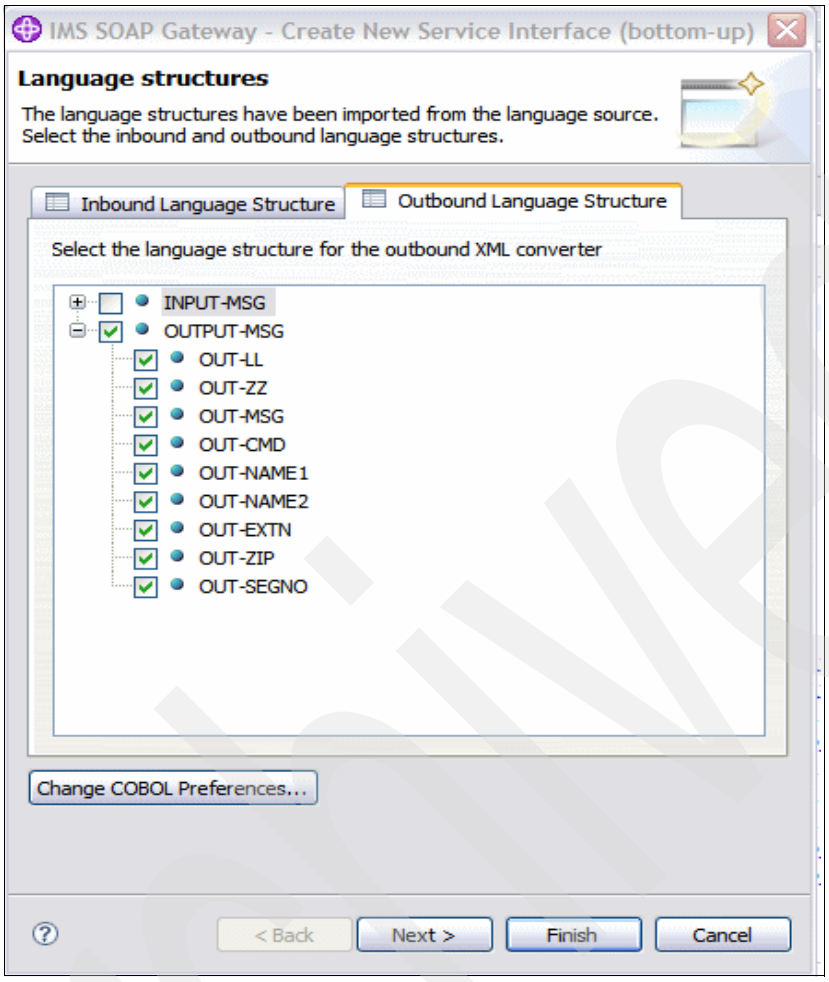


Figure 7-9 Outbound language data structure selection

13. Click **Next**.

14. In the Generation Options window, in the Host code page field, select the code page that the host uses. Do not change the settings for the inbound and outbound code pages, as IMS SOAP Gateway supports only UTF-8 encoding.

Archive

15. Select the **WSDL and XSD** tab. In the Service location field, specify the host name and the port name to the location of the IMS SOAP Gateway. This URI specifies the address of our new Web service, as shown in Figure 7-10.

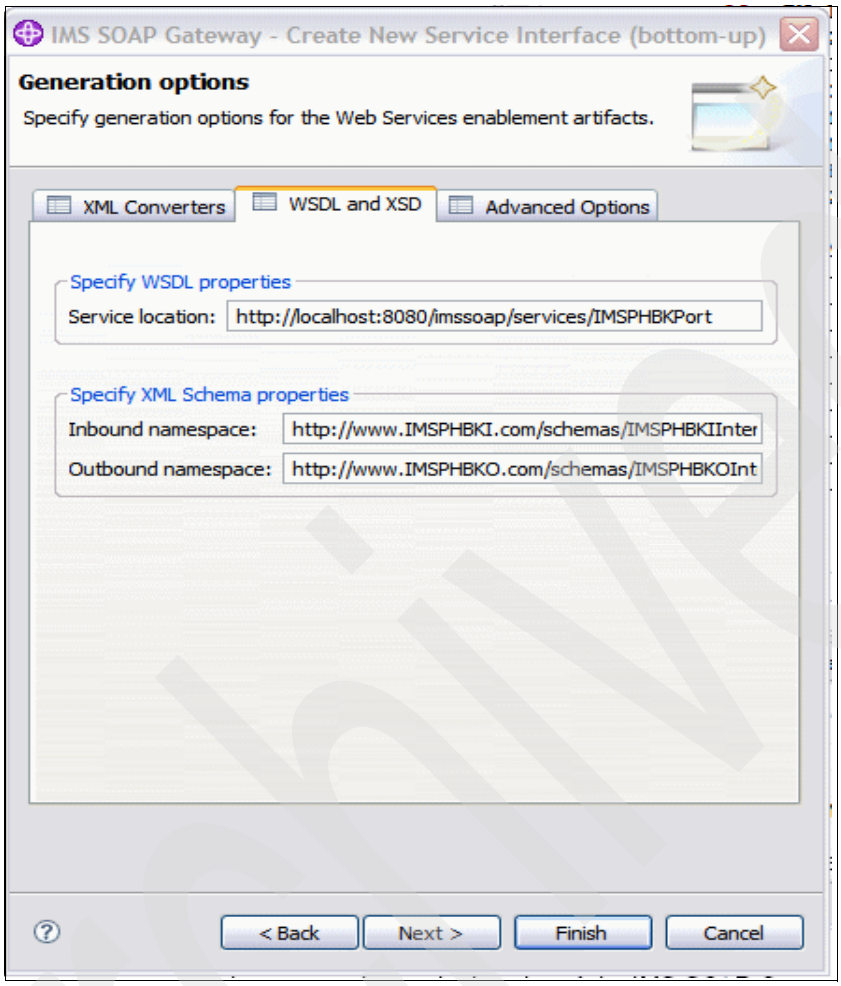


Figure 7-10 Specify the address of the Web service provider

Archive

16. Click **Next**.

You may keep the default IMS SOAP Gateway correlator properties or specify the ones that define how you want to interact with your IMS application. Let us keep the default ones for your telephone book application, as shown in Figure 7-11.

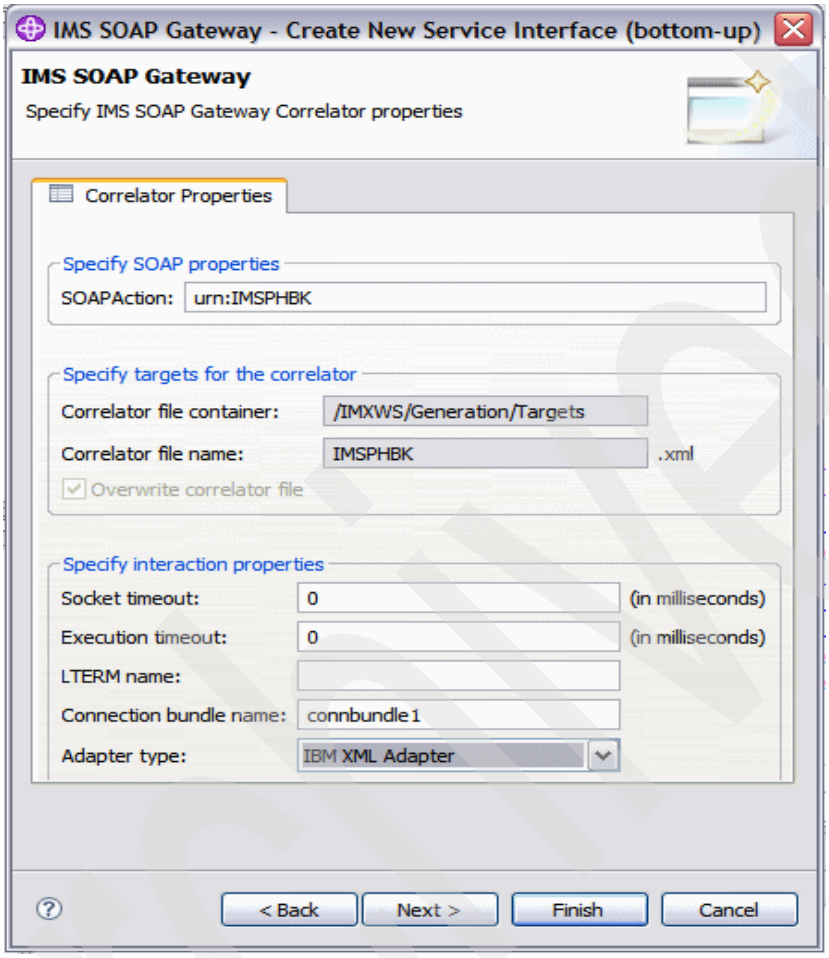


Figure 7-11 The Correlator Properties

17. Click **Next**.

In the File, data set or member selection window, you may change the default name of the COBOL converters and drivers. Ensure that **Generate all to driver** is selected. This will result in a single COBOL program being deployed under IMS. See Figure 7-12 on page 191.

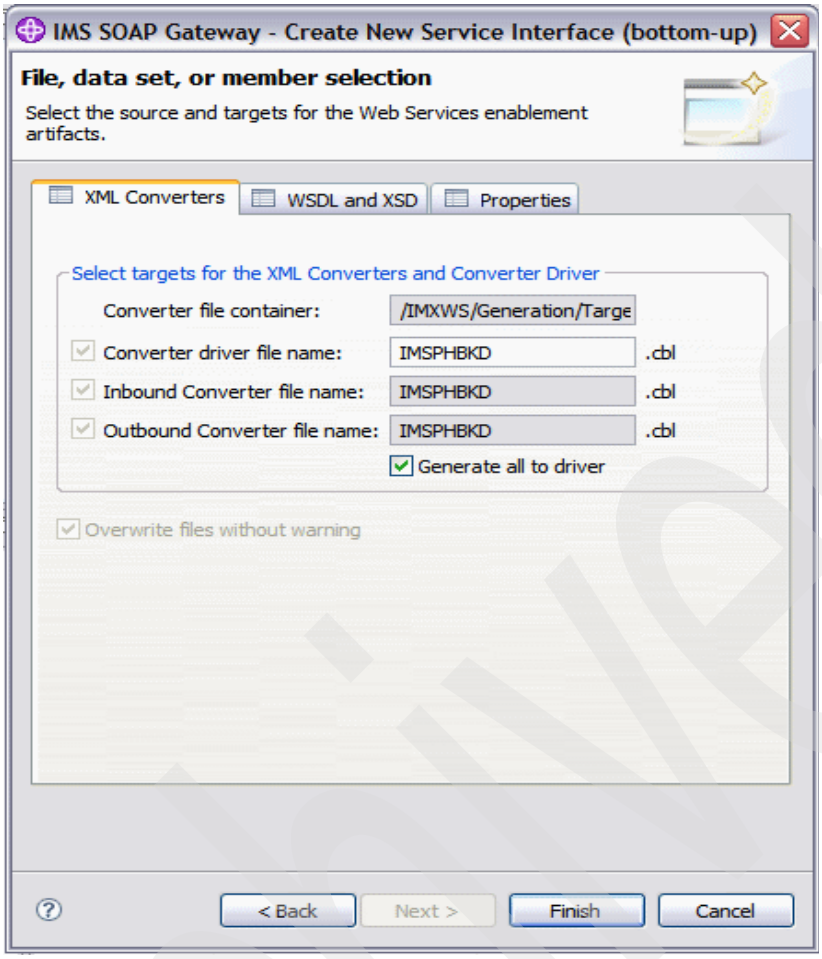


Figure 7-12 The Driver and Converters file names

18. Click **Finish**.

After the Web Service Enablement wizard completes its generation, you should see XML artifacts in the IMSWS Project under the folder Generation. The .xml files are the saved properties that will be used in the next wizard generation run for this project.

Under the sub folder Target, you should see the generated artifacts:

- IMSPHBK.wsdl: The WSDL file that describes this IMS Web service.
- IMSPHBKD.cbl: The conversion program that is made up of the converter driver, and input and output converters.
- IMSPHBKI.xsd and IMSPHBKO.xsd: The XML Schema Definition files that describe the input and output XML documents respectively.
- IMSPHBK.xml: The correlator file.

You can double-click the generated conversion program and do a find for “XML Parse”. You can see that the XML parsing is done by the converter program. See Figure 7-13.

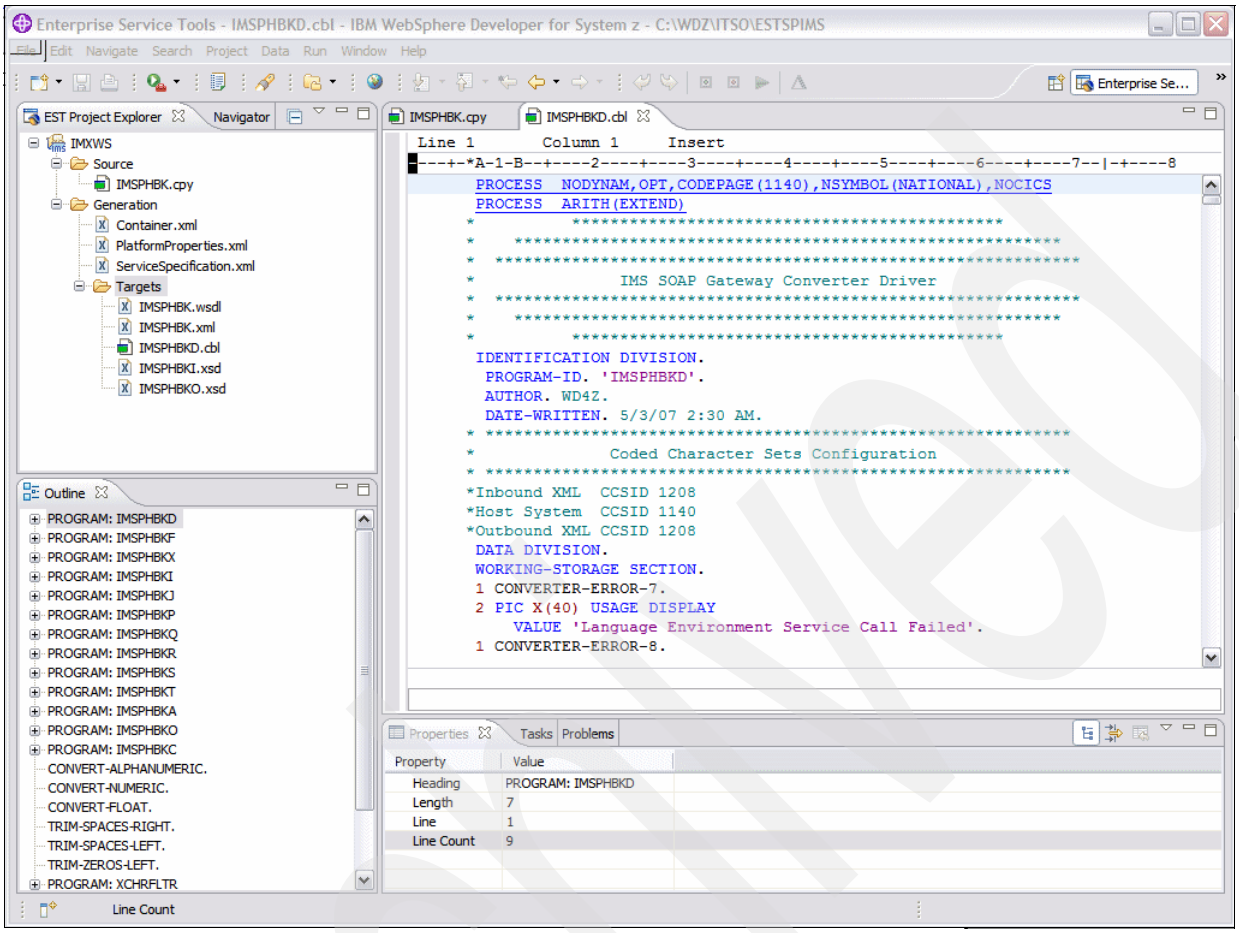


Figure 7-13 The XML converter program generated

Note: The converter COBOL program is handling the required XML parsing. The XML parsing code requires Enterprise COBOL Version 3.1 or later. The existing COBOL business programs can be written in older versions of the COBOL language.

Let us look at the correlator file generated, as shown in Figure 7-14 on page 193.

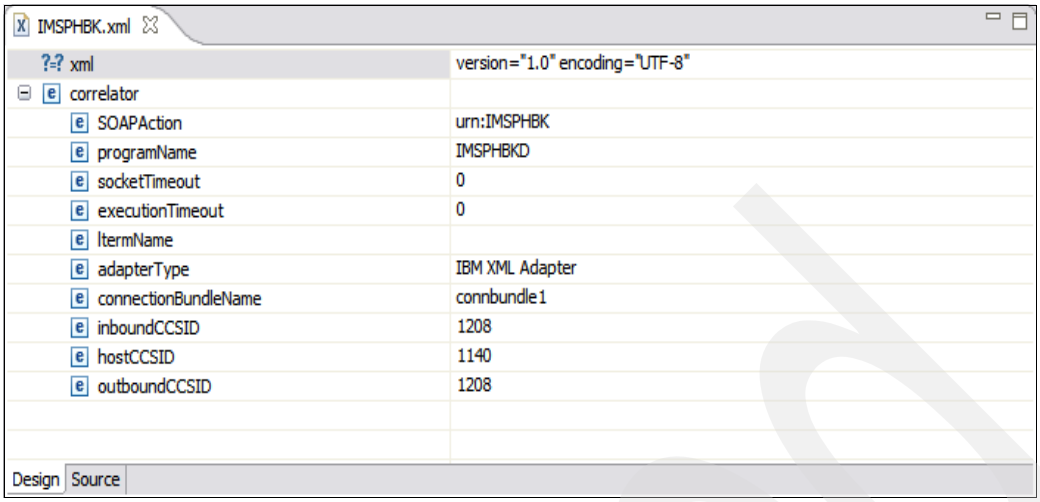


Figure 7-14 The correlator file generated

The correlator file specifies the program name, the runtime properties, and the information that the IMS SOAP Gateway needs to match incoming requests to our backend IMS application.

Following this generation step, you now have to deploy the WSDL file and the correlator file to the IMS SOAP gateway.

7.2.2 Deploy the Web Service converter source file

Do these steps:

1. Using the Enterprise Service Perspective and the EST project explorer, right-click **DFH0XCMD.cbl** and select **Copy**.
2. To paste the program to your z/OS COBOL directory, switch to the z/OS perspective, expand the MVS files folder, right-click your source directory (here *<Userid>*.WDZ.COBOL), and select **Paste**.

The file IMSPHBKD is then copied to your z/OS source directory. You have now to compile your converter file

It is convenient to create a z/OS project and move the assets that you are working on to a subproject. This makes the task simpler because you have all assets being used grouped in the same location. Do these steps:

1. From the z/OS Projects view, right click the blank area and select **New z/OS Project**.
2. Type EST_CONVERTER as the project name, select **Create an MVS subproject**, and click **Finish**.
3. Type EST_CONVERTER_IMS as the subproject name and click **Next**. Note that the High Level Qualifier will be your user ID.
4. Change the JCL job card to your own standards and specify the loadlib where the converter load module will be stored.
5. Click **Finish** to create the project and the subproject.

To make the resource is available to the subproject, you have to add it. Do these steps:

1. From the Remote System view, expand your source files directory. Select your converter program **IMSPHBKD.cbl**, right-click it, and select **Add to Subproject**.
2. Select the project and subproject names you just created and click **Finish**.
3. Switch to you z/OS project view. You can see that your converter program has been added to your subproject.

Now you have to generate the JCL to compile and linkedit the converter program. Right-click the program name and select **Generate JCL** → **For Compile and Link**. The JCL member generated is added to your subproject.

To check the Job execution, locate the job you just submitted in the JES folder in the remote system view. Double-click it. The output listing opens in the editor view.

The converter is now in place to be activated.

7.2.3 Deploy the WSDL file and the correlator file to IMS SOAP Gateway

The deployment is the step where you set up properties and create the runtime code that will be used by the IMS SOAP Gateway to enable your IMS application as a Web service.

The IMS SOAP Gateway provides a deployment utility function that helps:

- Specify the WSDL file to use for the deployment.
- Specify the interaction properties of the Web service with the correlator file.

Deploying the Web service to IMS SOAP Gateway

The deployment utility can be used in either Interactive or Expert mode.

We use the Interactive mode to deploy our application as a Web service. The Interactive mode provides an interactive user interface that walks you through each command needed to deploy your application from end to end. Do these steps:

1. Start the deployment utility in the interactive mode by selecting **Start** → **All Programs** → **IBM IMS SOAP Gateway Version 9.2.1** → **Deployment Utility**.

   The deployment utility interaction mode starts and shows the different tasks you can perform, as shown in Figure 7-15 on page 195.

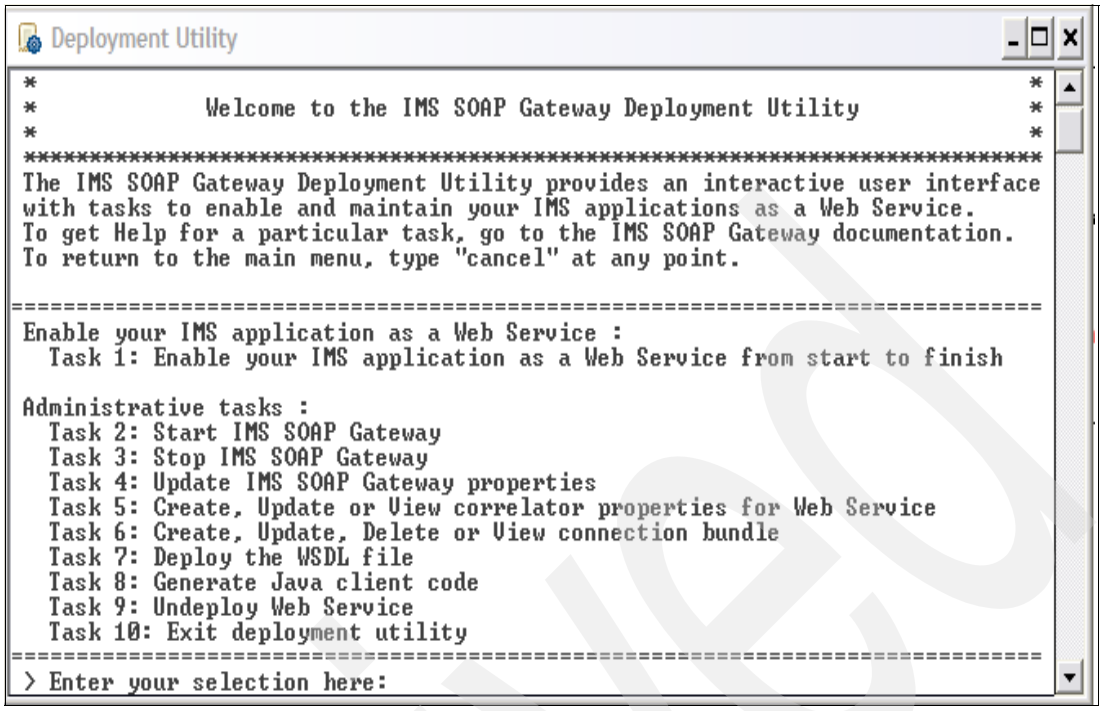


Figure 7-15 The Interactive Deployment Utility

To deploy the WSDL and the XML files, and set up the properties to make your IMS application accessible as a Web service, use Task 1.

2. When prompted to enter your selection, type 1 and press Enter.

Define the WSDL file to use

Do these steps:

1. Specify the path and name of your WSDL file by copying and pasting the path of the WSDL file generated by WD/z and typing the name (here IMSPHBK.wsdl).
2. In response to “Do you want to view all existing connection bundles?”, type n.
3. In response to “Do you want to create a connection bundle?”, type y.
4. Provide the connection bundle properties shown in Table 7-1.

Table 7-1 The connection bundle properties

Property	Value
Name of the connection bundle	connbundle1
IMS Host name or IP address	Your IMS connect IP address
IMS port number	Your IMS connect port number
IMS Datastore Name	Your IMS datastore name
RACF User ID (optional)	Your user ID, if needed
RACF password (optional)	Your password, if needed
RACF group name (optional)	Your groupname, if needed
Do you want to set SSL properties?	n

Archived

The Deployment Utility should show the message `The connection bundle 'connbundle1' has been created`, as shown in Figure 7-16.

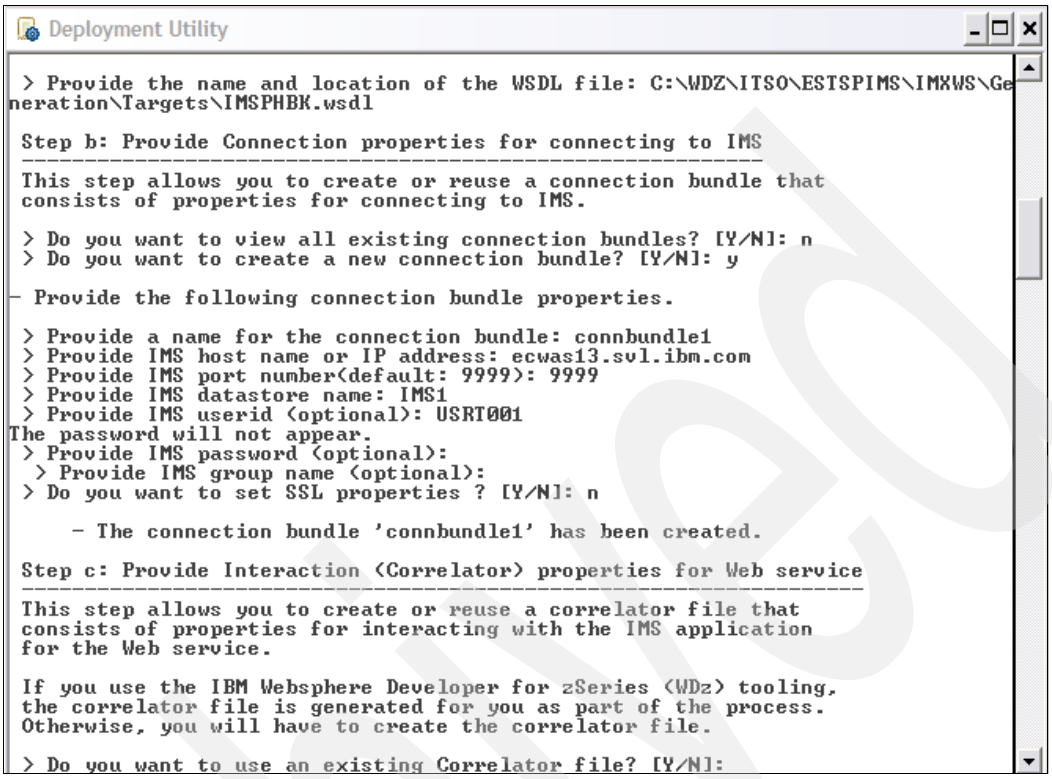


Figure 7-16 WSDL file deployment

Let us now provide the interaction properties for the application.

Provide interaction properties for the application

The correlator .xml file as been generated by WD/z.

Notice that the Connection Bundle Name must correspond to the one provided in “Define the WSDL file to use” on page 195, as shown in Figure 7-17 on page 197.

Archived

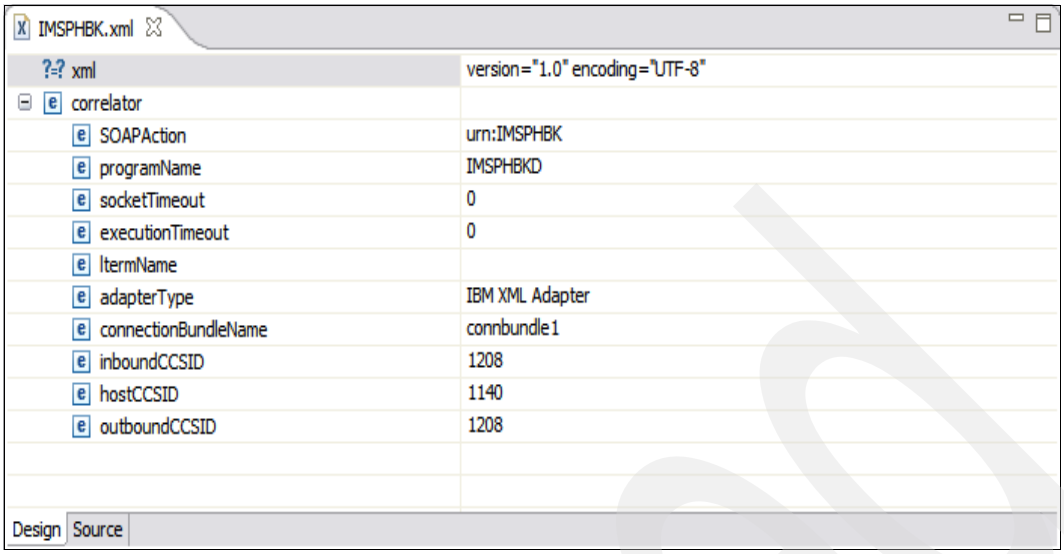


Figure 7-17 The correlator file generated by WD/z

When prompted by the deployment tool, use the values shown in Table 7-2.

Table 7-2 Correlator file values

Prompt	Correlator file value
Do you want to use an existing Correlator file?	y
Provide the name and full path to the Correlator file.	The path where WD/z has generated the correlator file + name (IMSPHBK.xml).
Do you want to view the correlator file?	y

Archived

The deployment utility should show the message `The connection bundle 'connbundle1' has been created`, as shown in Figure 7-18.

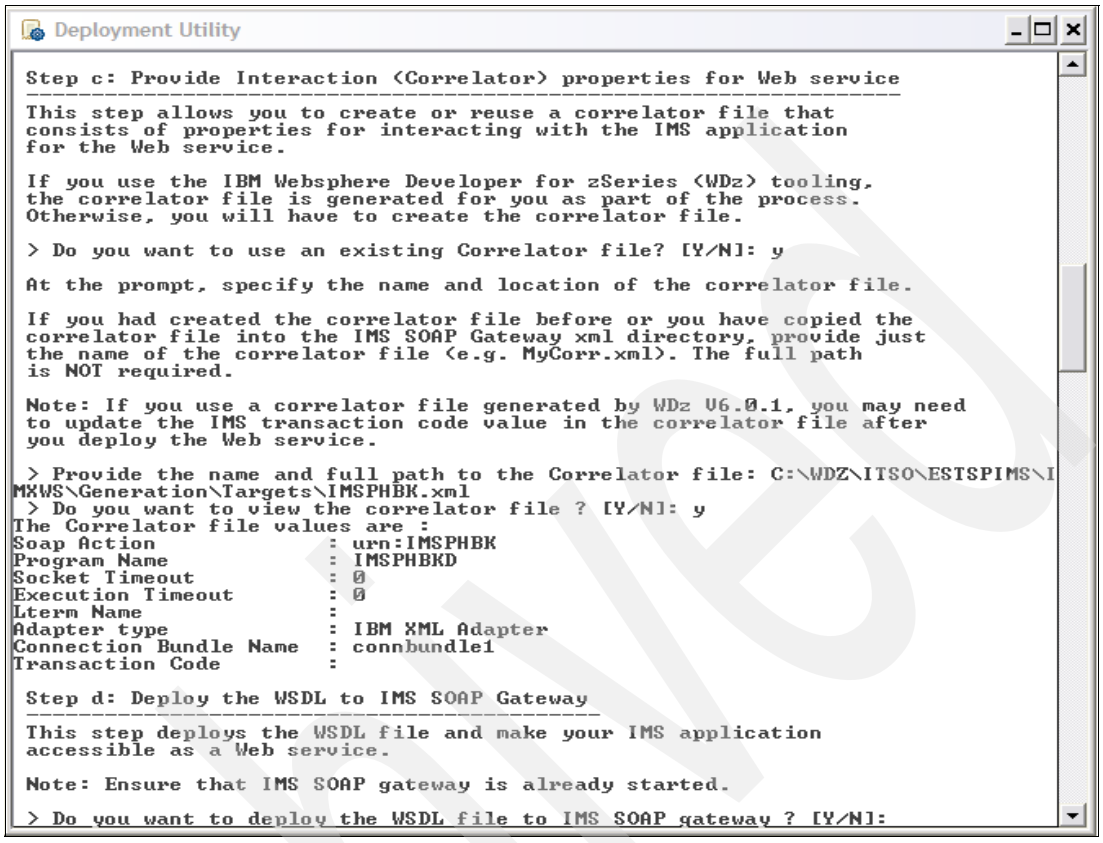


Figure 7-18 Provide Interaction Properties for the application

You may check the correlator file values here.

Deploy your application to IMS SOAP Gateway

Now the deployment continues with the deployment of your application to the IMS SOAP Gateway. When prompted by the application, use the values shown in Table 7-3.

Table 7-3 The application deployment dialog

Prompt	Value
Do you want to the deploy the WSDL file to IMS SOAP gateway?	y
Is your SOAP Gateway already started?	n (If you enter n, it will start the SOAP Gateway for you. If your SOAP Gateway has already started, enter y instead.)

The deployment process starts, as shown in Figure 7-19 on page 199.

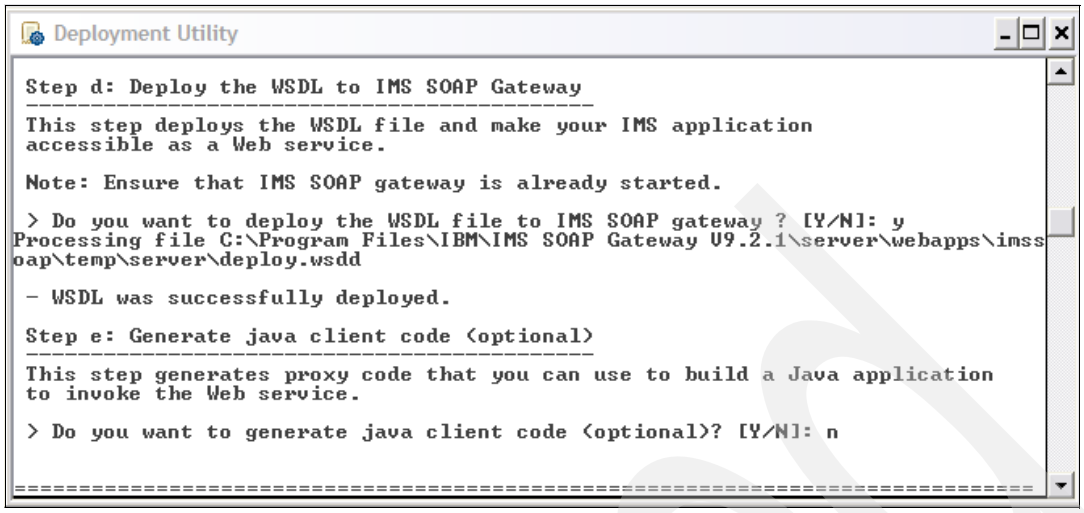


Figure 7-19 Deployment process

You should see the following message when it ends:

```
WSDL was successfully deployed.
```

Restart the IMS SOAP Gateway and check the deployment

You can stop the IMS SOAP Gateway by using option 3 - Stop IMS SOAP Gateway and start it again by using option 2 - Start IMS SOAP Gateway at the Deployment Utility menu.

Your IMS application is ready to be accessed as a Web service. To verify that the deployment has completed successfully, do the following steps:

1. Start the IMS SOAP GAteway Administrative Console by selecting **Start** → **All Programs** → **IBM IMS SOAP Gateway 9.2.1** → **Administrative console**.

   This action opens a Web browser and displays the Administrative console.

Archived

2. Click **View Deployed Web Services**. You should see a page similar to Figure 7-20.

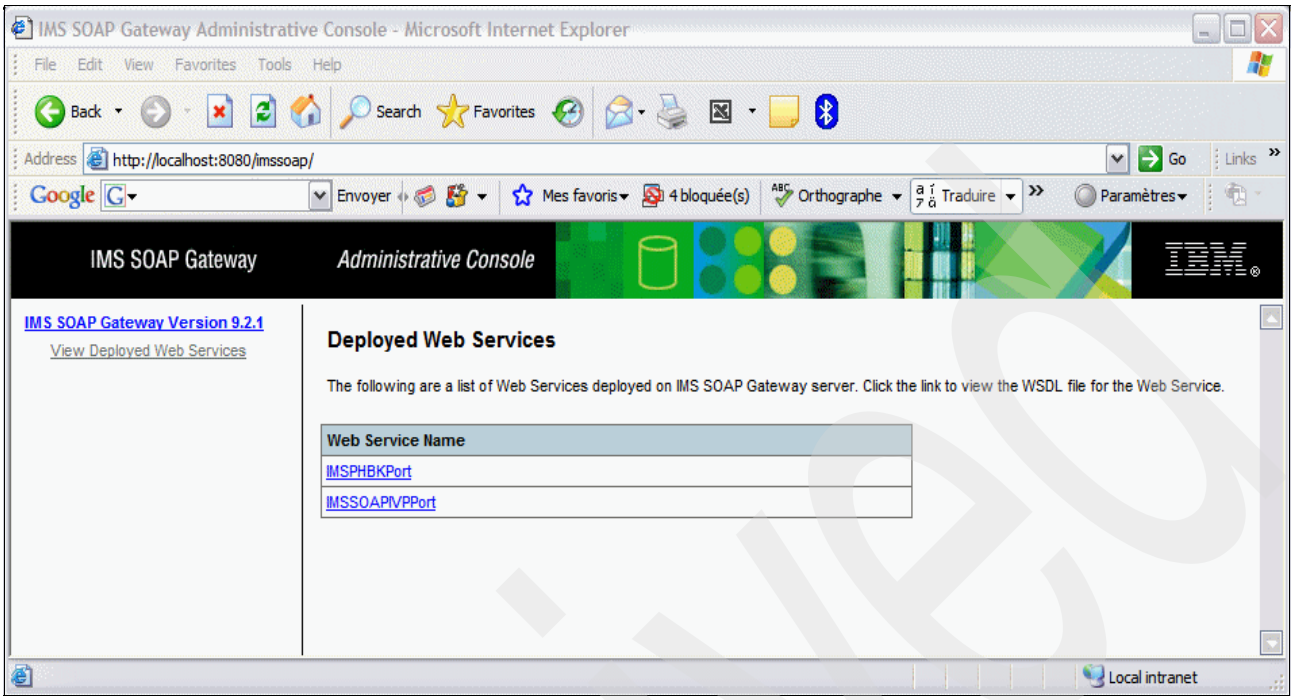


Figure 7-20 The IMS SOPA Gateway Administrative Console

7.2.4 Test your IMS Web Service using WebSphere Developer for System z

Do these steps:

1. Within WD/z, switch to the Enterprise Service tools perspective.
2. Select your generated WSDL file. Right-click it and select **Web Services** → **Test with Web Service Explorer**.
3. The Web Service Explorer view opens. Expand the **Binding** folder. Click the **Operation** folder.

> **Note:** In the Actions pane, the Endpoints field should show the default endpoint. If you need to edit the endpoints, select **DFH0XCMNBinding** and use the actions to add, change, or remove endpoints.

Input the required input values to invoke the Web Service, as shown in Figure 7-21 on page 201.

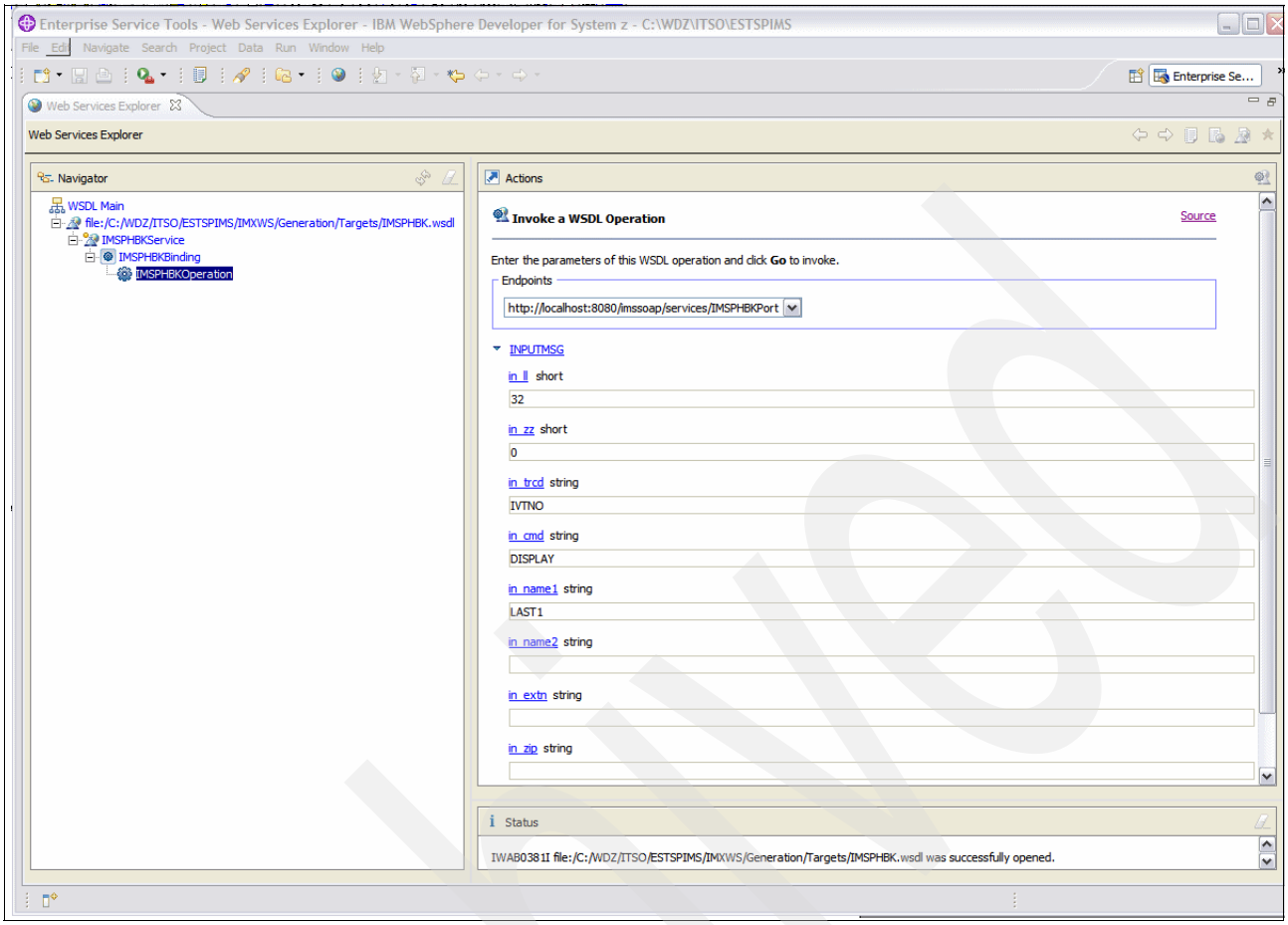


Figure 7-21 Test your IMS Service using the Test Service Explorer

Archived

Archived

8

Service Flow tools

This chapter provides information about using the Enterprise Service Tools Service Flow capabilities of WebSphere Developer for System z to model a business service from existing CICS applications.

This chapter provides descriptions of the following development tasks:

1. Create a service flow project.
2. Record terminal interactions with the sample Catalog Manager application.
3. Generate runtime code for deployment.
4. Test the deployed service.

Before describing the different development steps of a service flow, let us describe what a service flow is, and why and when it should be used.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

8.1 Service flow: What, why, and when

A service flow is a non-interruptible micro-flow that is constructed from a collection of nodes that represent the invocation of CICS resources. The flow describes the navigation of the nodes and allows data mapping between the nodes.

A single request may cause the execution of many CICS resources, which allows us to compose several CICS interactions into a single business service, as shown in Figure 8-1.

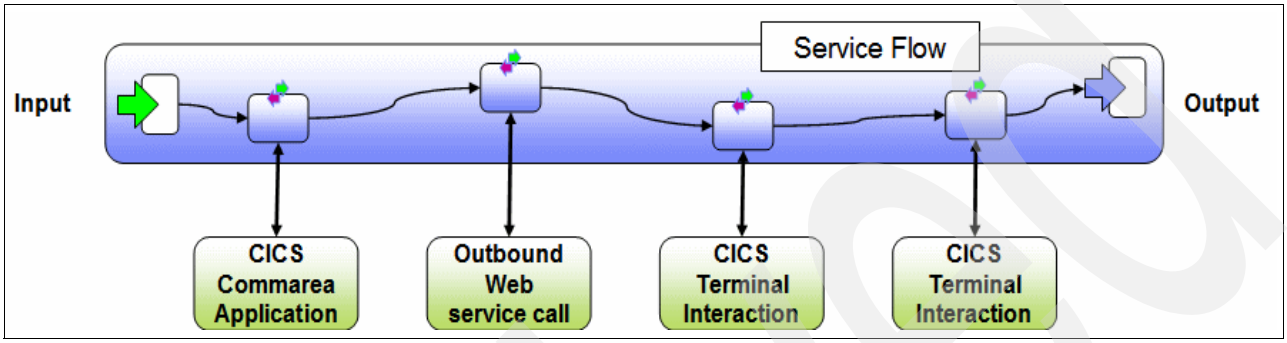


Figure 8-1 A Service Flow representation

Three categories of nodes can make up a flow:

- Input/Output
- Invoke
- Flow control

Figure 8-2 on page 205 lists the components of a service flow within the three categories.

Archive

	Receive	Input node for flow
	Reply	Output node from flow
	Throw	Throw fault node
	Invoke	Generic invoke (target not yet defined)
	Invoke Operation	Invoke a non-terminal operation
	Invoke Terminal	Invoke a terminal screen operation
	Invoke Outbound Web service	Invoke an outbound Web service call
	Invoke Flow	Invoke a flow
	Assign	Map data between messages in the flow
	Switch	Flow control decision node
	While	Flow control loop node

Figure 8-2 Components of a service flow

At the core of the Service Flow Model is a flow editor that allows visualization and visual editing of the sequence of host interactions needed to implement a business operation. Flows can be built from host interactions, subflows, loops, and branches. Each flow represents a business operation.

The CICS Service Flow Feature introduces the capability required to compose fine-grained CICS application interfaces, both 3270 and COMMAREA, to form a CICS business service that exposes a coarse-grained business level interface. Using the CICS Service Flow Feature, existing CICS application interfaces can be imported as reusable components. The CICS Service Flow Modeler delivers the flow modeling and mapping capabilities required to transform the behavior of the application components to form a business service without changing the underlying implementation.

8.2 Create the Service Flow project

We use the bottom-up approach by creating a service from an existing CICS terminal application.

The CICS TS V3.1 sample Catalog Manager application is used. This application is invoked from a CICS terminal through the EGUI transaction.

We create a service that will return a list of items that are available from the catalog.

Do these steps:

1. We create a service flow project to record interactions with a terminal application.

2. Start WebSphere Developer for System z.
3. Switch to the Enterprise Service Tools perspective.
4. Click anywhere in the EST Project Explorer view.
5. Right-click and select **New** → **Service Flow Project**.
6. In the New Service Flow Project window, specify a project name, like SFM, as shown in Figure 8-3.

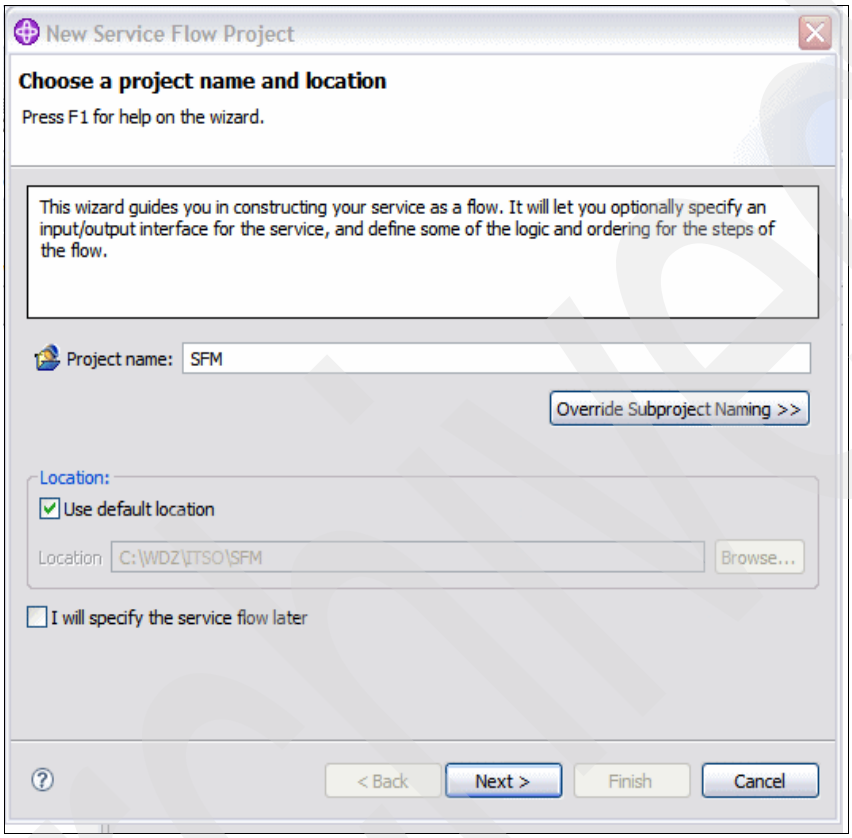


Figure 8-3 Specify a project name

7. Click **Next**.
8. In the New Service Flow Project - Associate service interface window, select **Define later** and click **Next**.
9. In the New Service Flow Project - Define service steps window, click **Recording interactions with a terminal application**, as shown in Figure 8-4 on page 207.

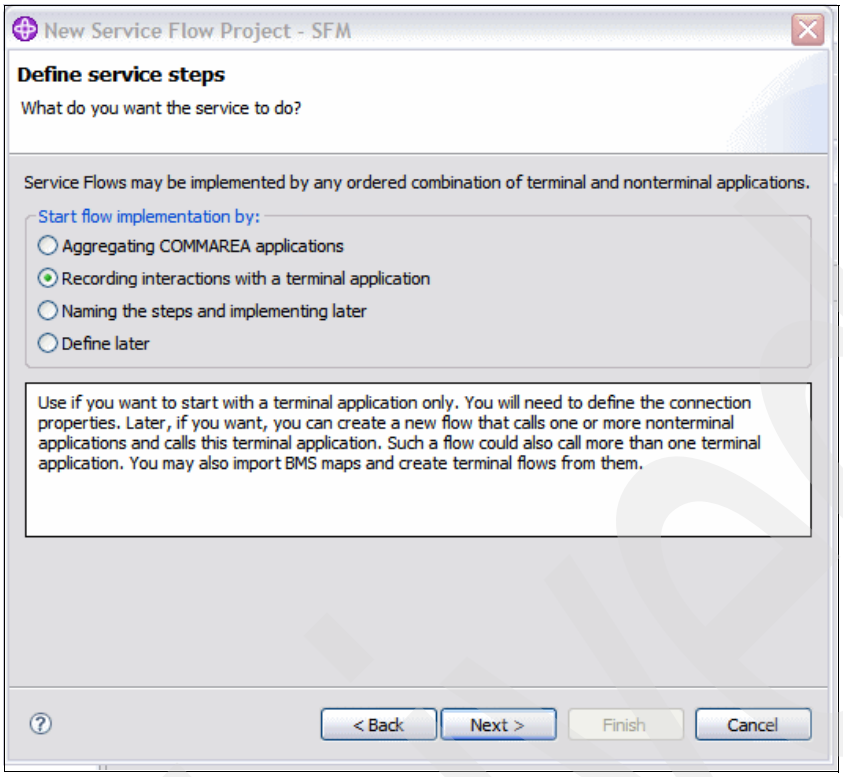


Figure 8-4 The service steps window

10. Click **Next**.

Archive

11. On the New Service Flow Project - Define host connection window, specify a File name and the Host name to access your CICS system, as shown in Figure 8-5.

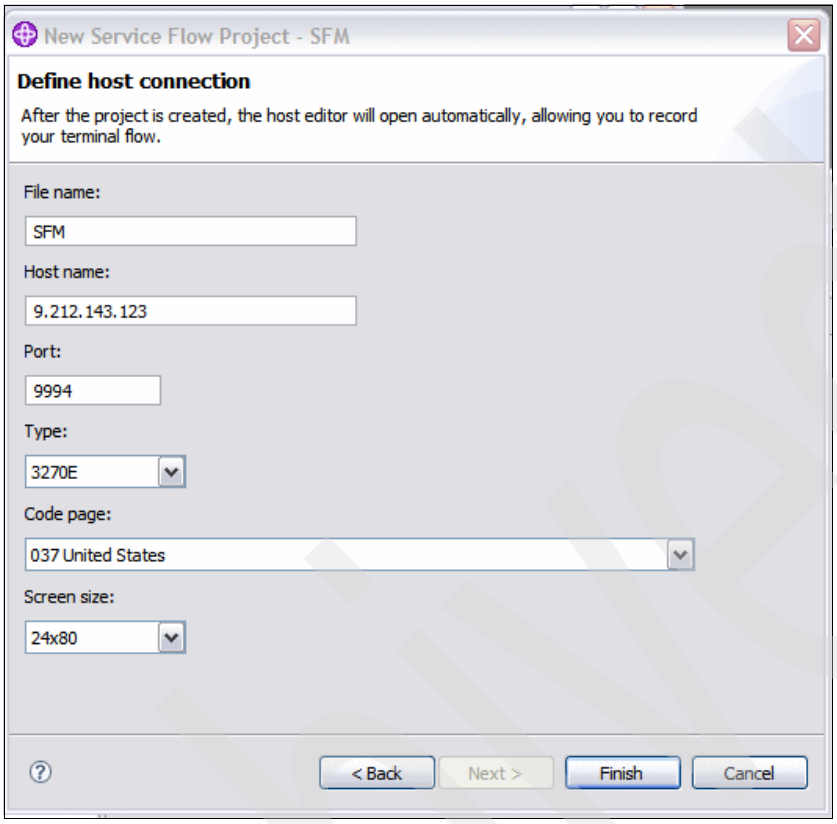


Figure 8-5 Define the host connection

12. Click **Finish**.

    Our SFM service flow project is now in the EST Project Explorer view and a host terminal emulation session to our mainframe system has been opened.

13. On the EST Project Explorer view, click the **+** sign to the left of the SFM project.

This project contains now four subprojects, as shown in Figure 8-6 on page 209.

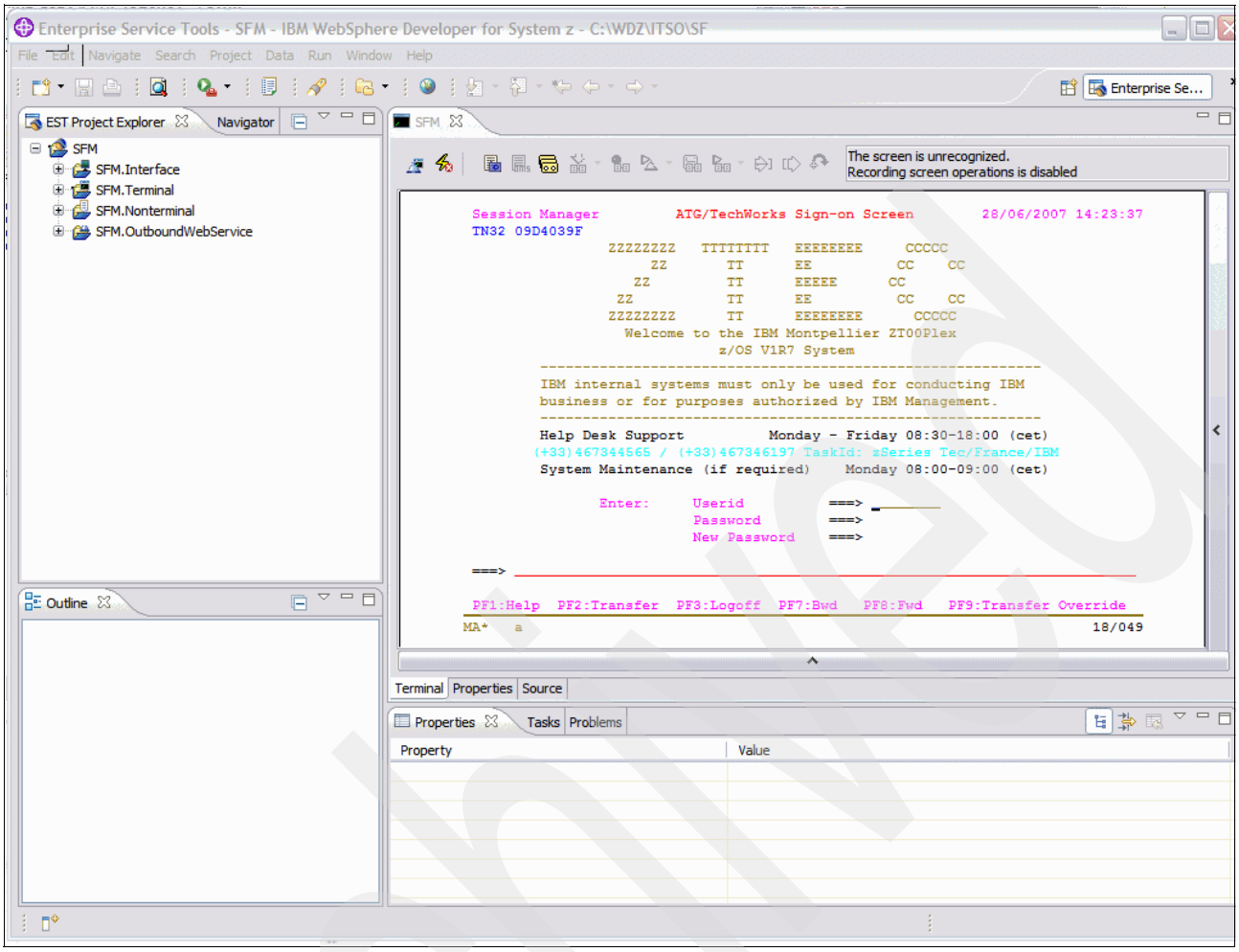


Figure 8-6 Creation of the service flow project

A service flow project provides the framework to organize the source and generated files associated with a service flow. Different artifacts are stored in a predefined set of three folders and four subprojects.

A folder is used to store a single class or type of artifact.

A subproject contains operations and messages that are used to invoke a particular type of entity or application.

The definition of those types of subprojects are shown in Table 8-1.

Table 8-1 Definition of the subproject types

Subproject type	Content
Interface definition	Operations and messages for invoking flows
Terminal applications	Operations and messages for invoking terminal or screen-based applications
Nonterminal applications	Operations and messages for invoking nonterminal (that is, COMMAREA based) applications

Subproject type	Content
Outbound Web services	Operations and messages for invoking outbound Web Services

In our implementation, we will only be dealing with the interface definition and terminal applications subprojects.

8.3 Record the terminal interactions

Here we record interactions with a terminal application.

The application we use is the CICS TS V3.1 sample Catalog Manager application that is invoked through the transaction EGUI.

To facilitate recording interactions with terminal applications, we provide definitions in the form of BMS maps. This also allows us to generate the service flow runtime application using Link3270 bridge facilities versus Front-End Programming Interface (FEPI). We import two BMS maps associated with the screens for the EGUI transaction.

Do these steps:

1. Select the **SFM.Terminal** subproject from the EST Project Explorer view.
2. Right-click and select **Import** → **BMS**.
3. In the Import - BMS window, make sure you have the right project selected and click the **File System** button.
4. In the Open window, navigate to the location where you have stored the two bms maps (dfh0xs1.bms and dfh0xs2.bms). Use the Ctrl key to select both.
5. Click **Open** to apply your selections.
6. Back in the Import - BMS window, verify that those two files are on the import list and click **Finish**.

Note the additional artifacts created in the Messages folder of the SFM.Terminal subproject. These messages represent the maps defined in dfh0xs1 and dfh0xs2, as shown in Figure 8-7 on page 211.

Archived

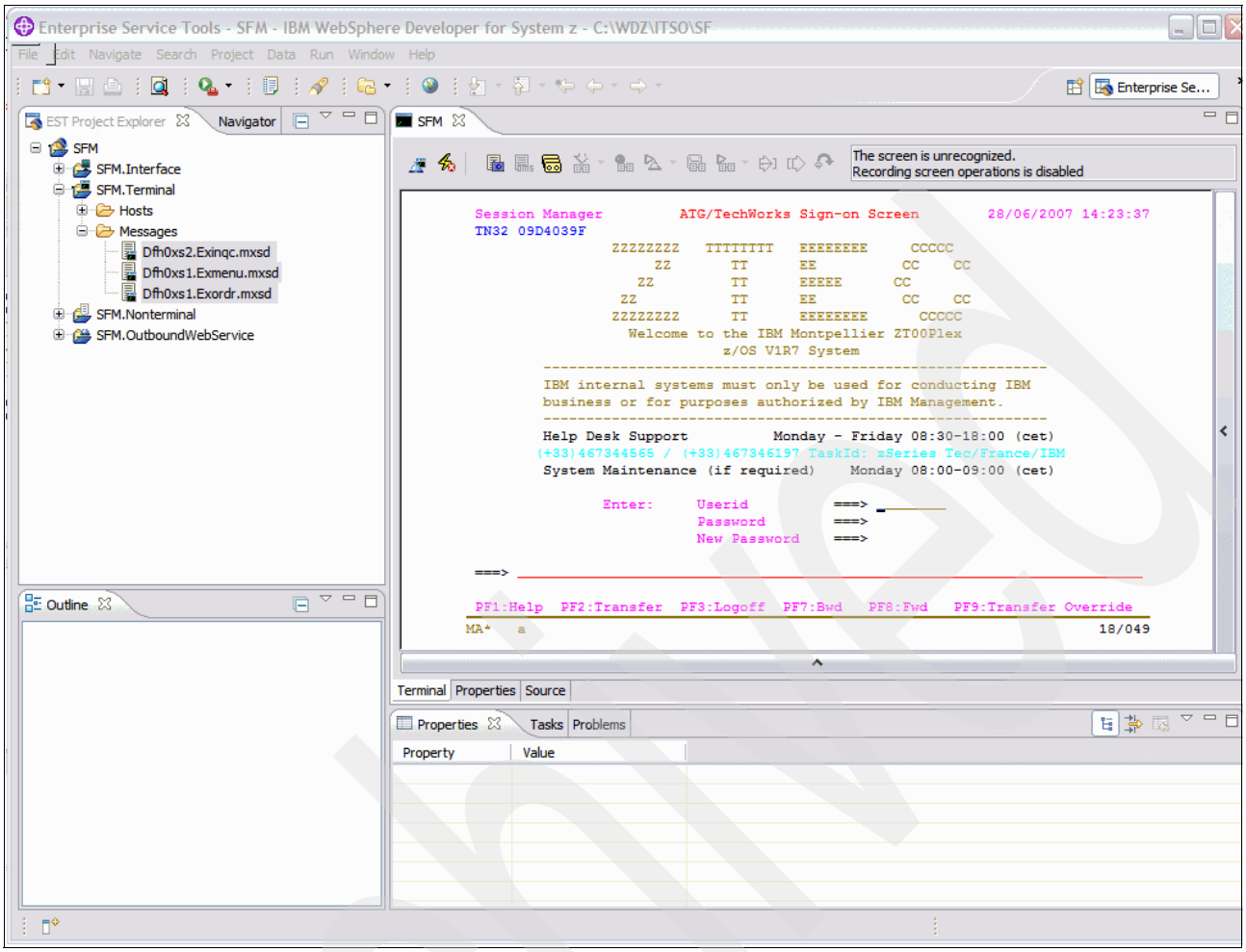


Figure 8-7 The messages of the service flow project

Move the cursor over the files with the *.mxsd extension, wait a few seconds, and you get a view of the window.

We use the Host editor to record the terminal interactions. The Host editor has a toolbar, an information area, and a terminal emulation area.

Do these steps:

1. Switch to the Host editor and click the **Reload Screen Descriptions** button on the toolbar to activate the descriptions that were created in the previous step.
2. Log on to your CICS system.

3. In the blank window, type the sample transaction code EGUI and press Enter (use the Ctrl key). Notice that the information area indicates recognition of the current window. Also notice that the Start Recording Flow button on the toolbar is now active, as shown in Figure 8-8.

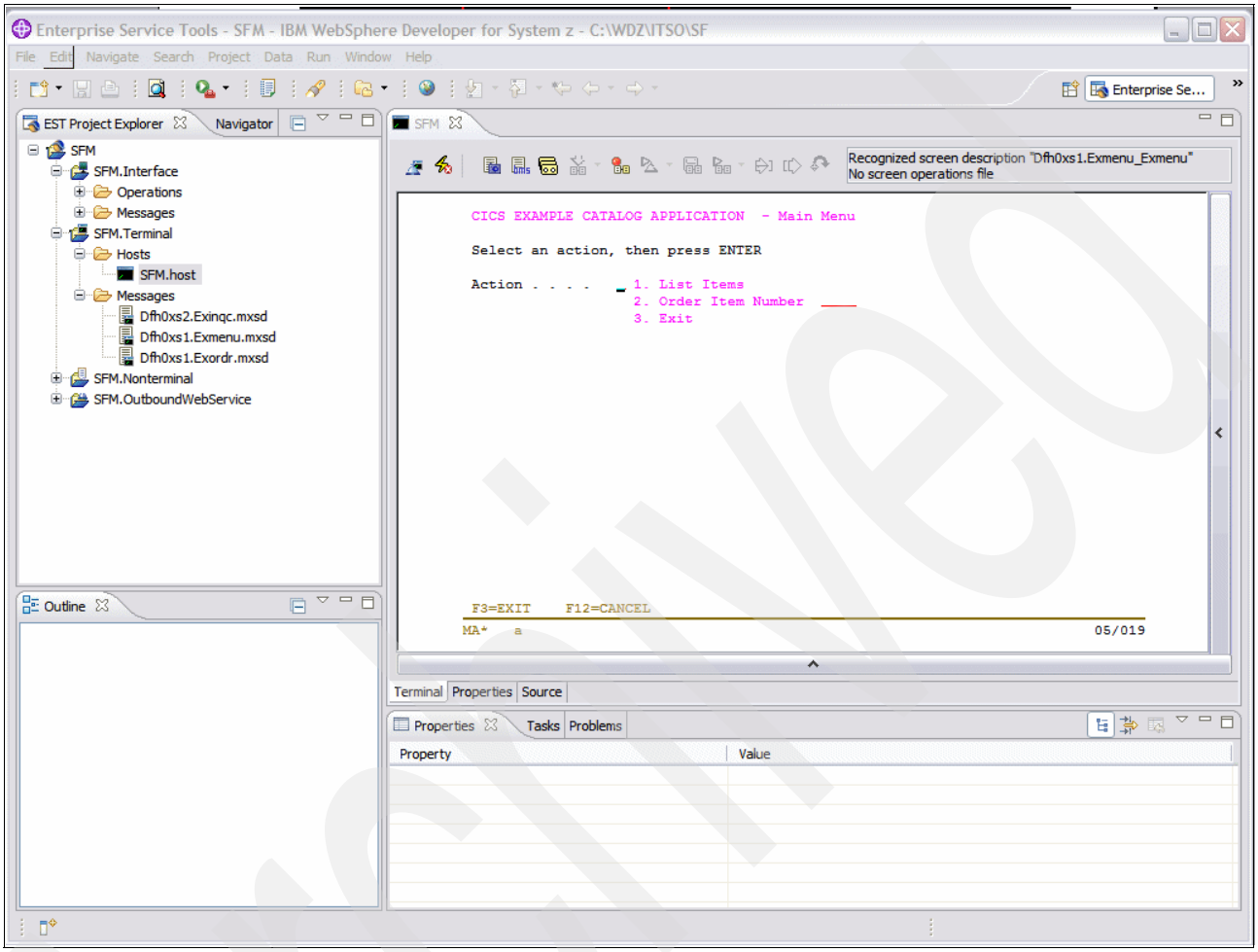


Figure 8-8 The EGUI transaction

We start our recording from this point. The transaction needed to initiate this application will be specified as part of the configuration for the runtime application. We will be recording the flow through the terminal application by entering a 1 to list items in the catalog and capturing the list of items that were returned.

4. Click the **Start Recording Flow** button on the toolbar.
5. Type 1 on the Action field and press Enter (use the Ctrl key).

You should see something similar to Figure 8-9 on page 213.

Archived

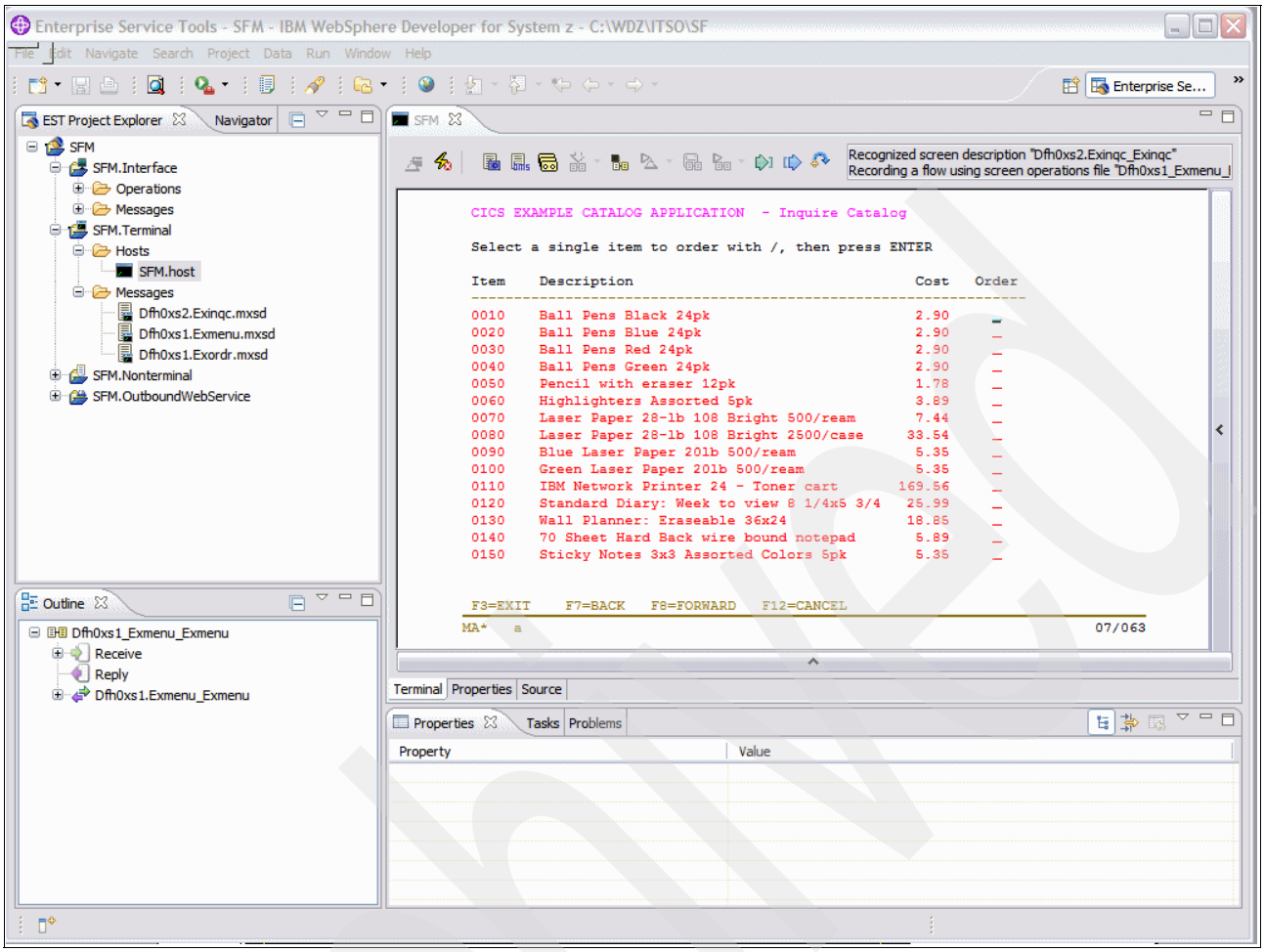


Figure 8-9 The EGUI recording

6. We extract the items from the window.

Archive

7. Position the cursor on the corner edge of the first item number, 0010. Click and hold the mouse button and stretch the rectangular frame (yellow border) so that it covers the entire list, ending with the cost of the last item, 5.35, as shown in Figure 8-10.

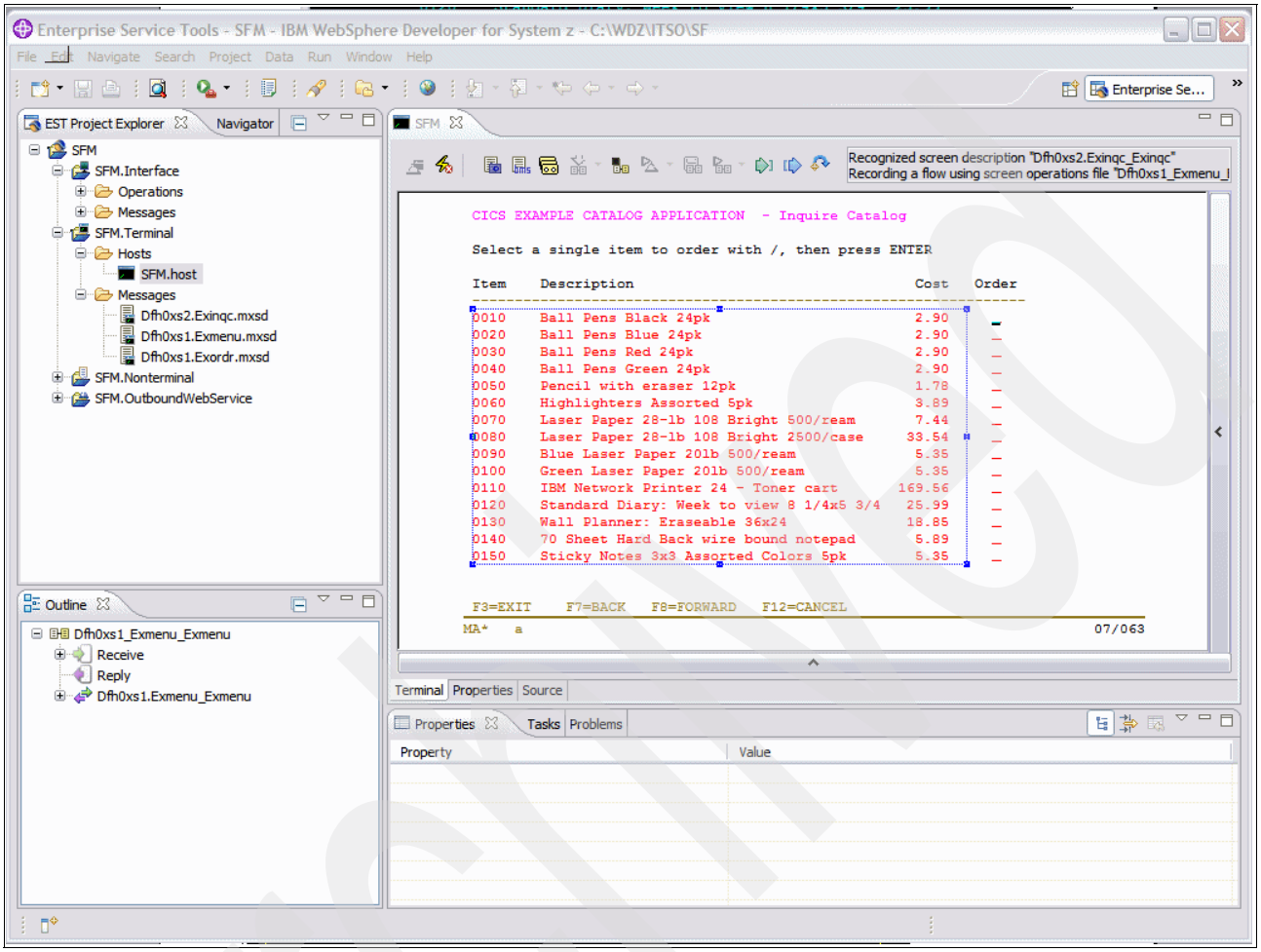


Figure 8-10 Extract the Items from the window

8. Click the **Extract Data from Screen** button on the toolbar.

   The Select Variable - Variables window opens, as shown in Figure 8-11 on page 215.

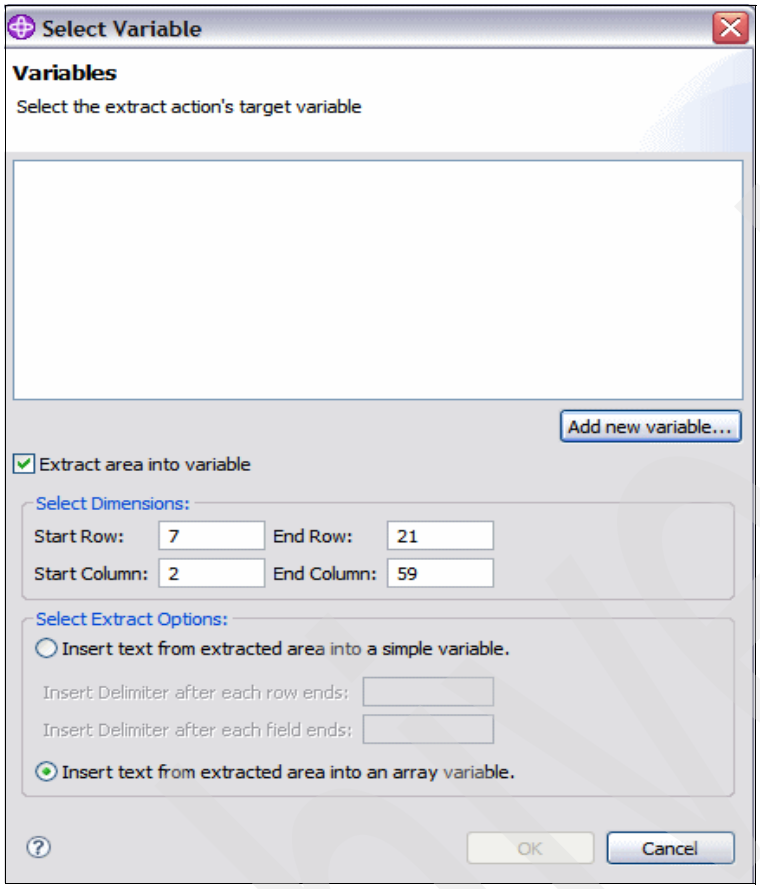


Figure 8-11 The Select Variable window

9. Click the **Add new variable** button.
10. In the Add New Variable window, specify a variable name of CatalogItems, as shown in Figure 8-12.

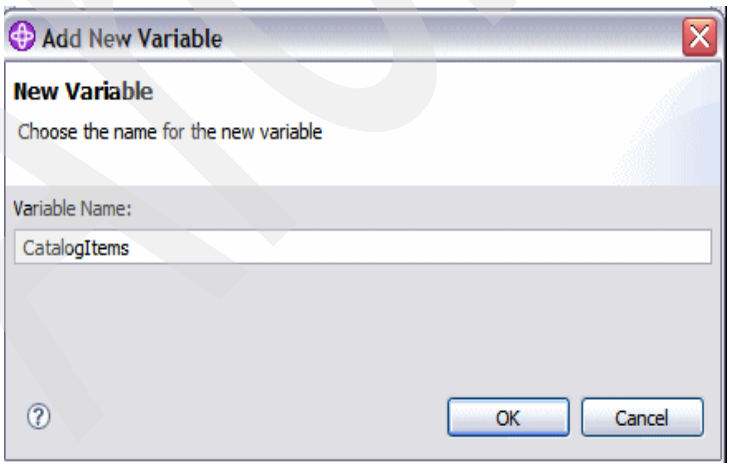


Figure 8-12 Add the variable name

11. Click **OK**, and click **OK** again to close the Select Variable dialog.

We have now recorded a basic flow through a terminal application.

12. Click the **Stop Recording Flow** button on the toolbar.
13. Click the **Save Flow** button on the toolbar to generate the terminal based service flow.

An artifact, SFM.seqflow has been created in the Flows folder, as shown in Figure 8-13.

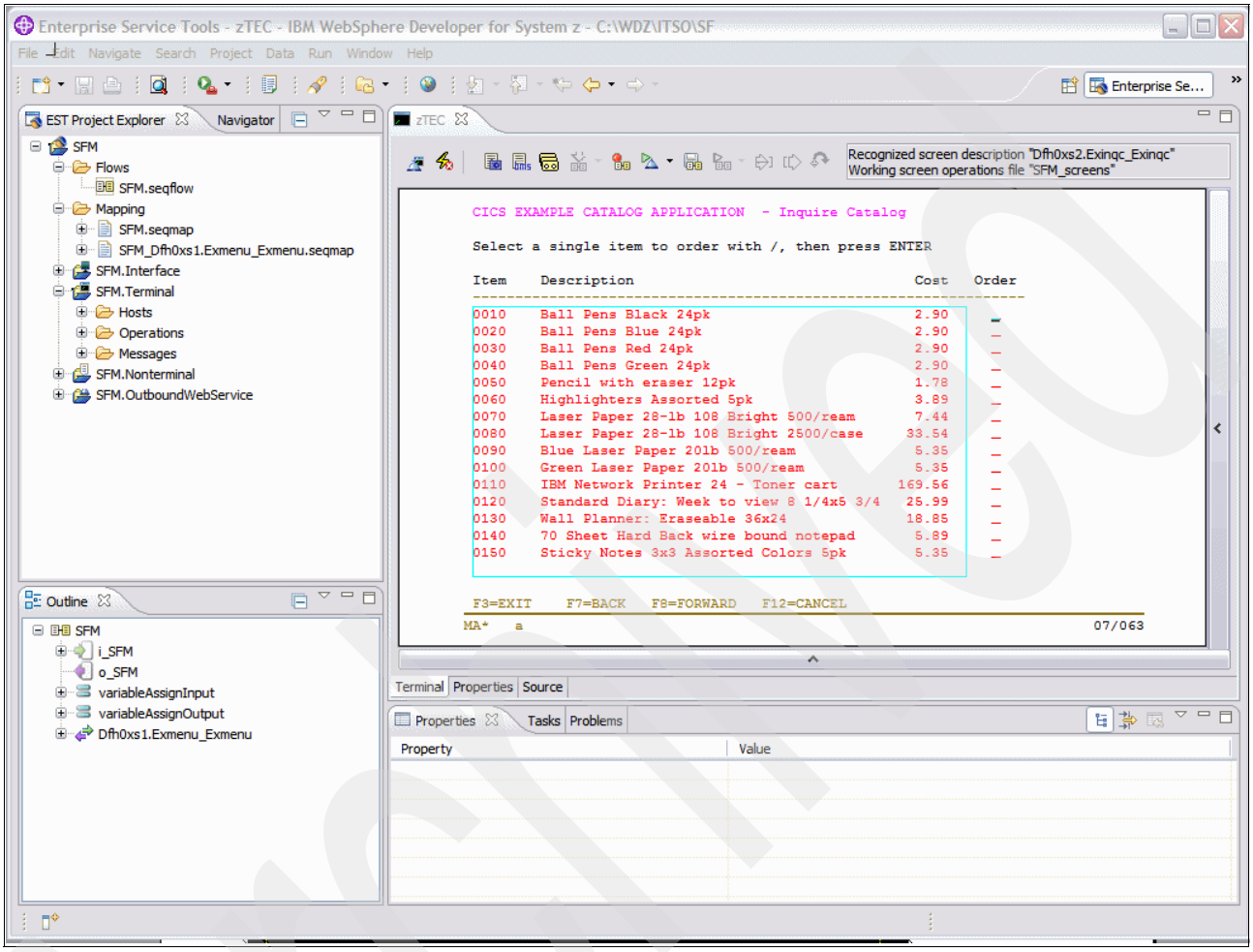


Figure 8-13 SFM_seqflow added in the Flows folder

Let us take a look to the service flow. Double-click **SFM.seqflow** in EST Project Explorer view. This should launch the flow editor, as shown in Figure 8-14 on page 217.

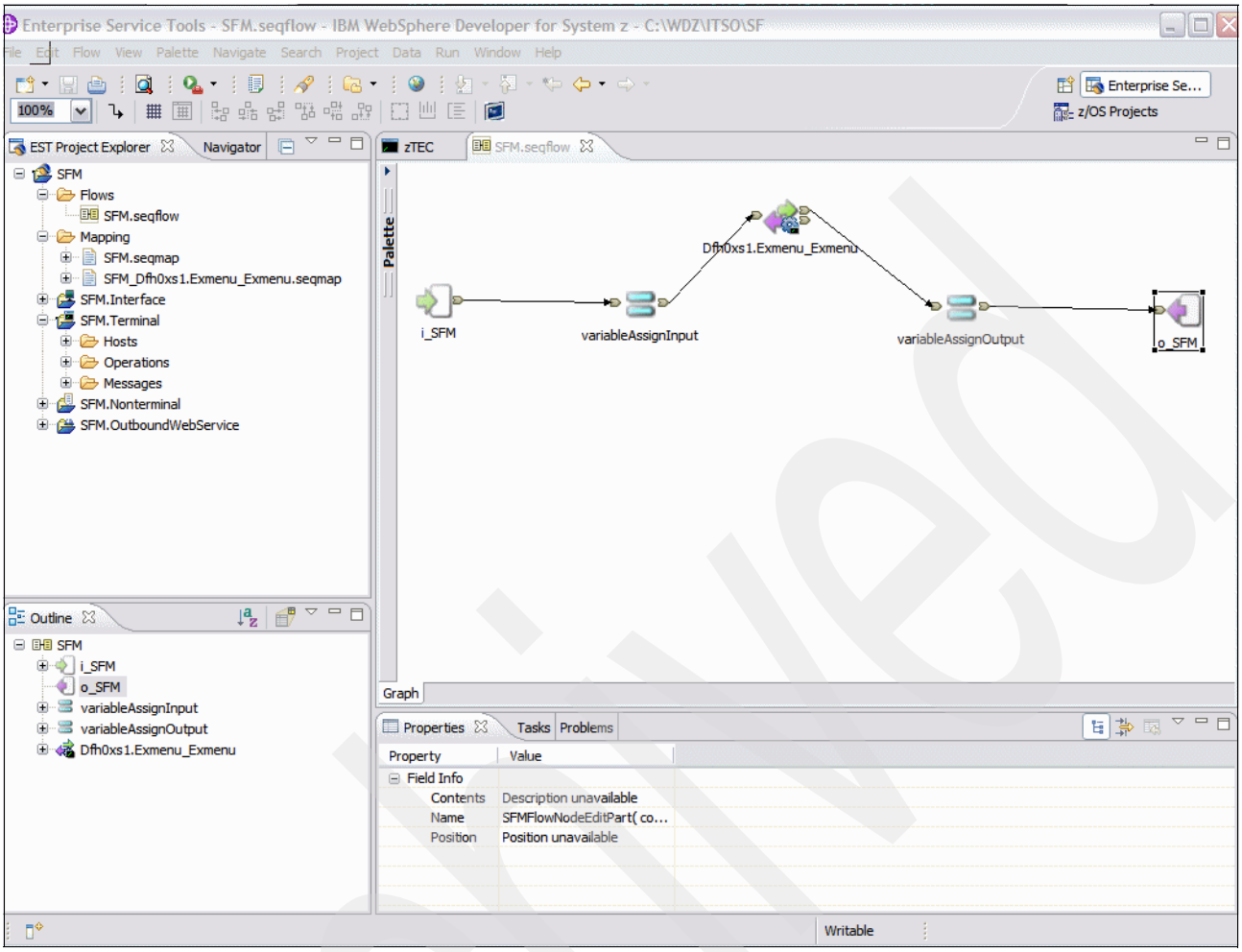


Figure 8-14 The Flow editor

The Flow editor is used to create new flows or edit flows captured or created by the Host editor. Since we used the Host editor to create the flow, all required mappings have been created for us.

Archived

With the Flow editor in focus, you can click the **Toggle Terminal Thumbnails / Icons** button to switch from an icon to a terminal thumbnail, as shown in Figure 8-15.

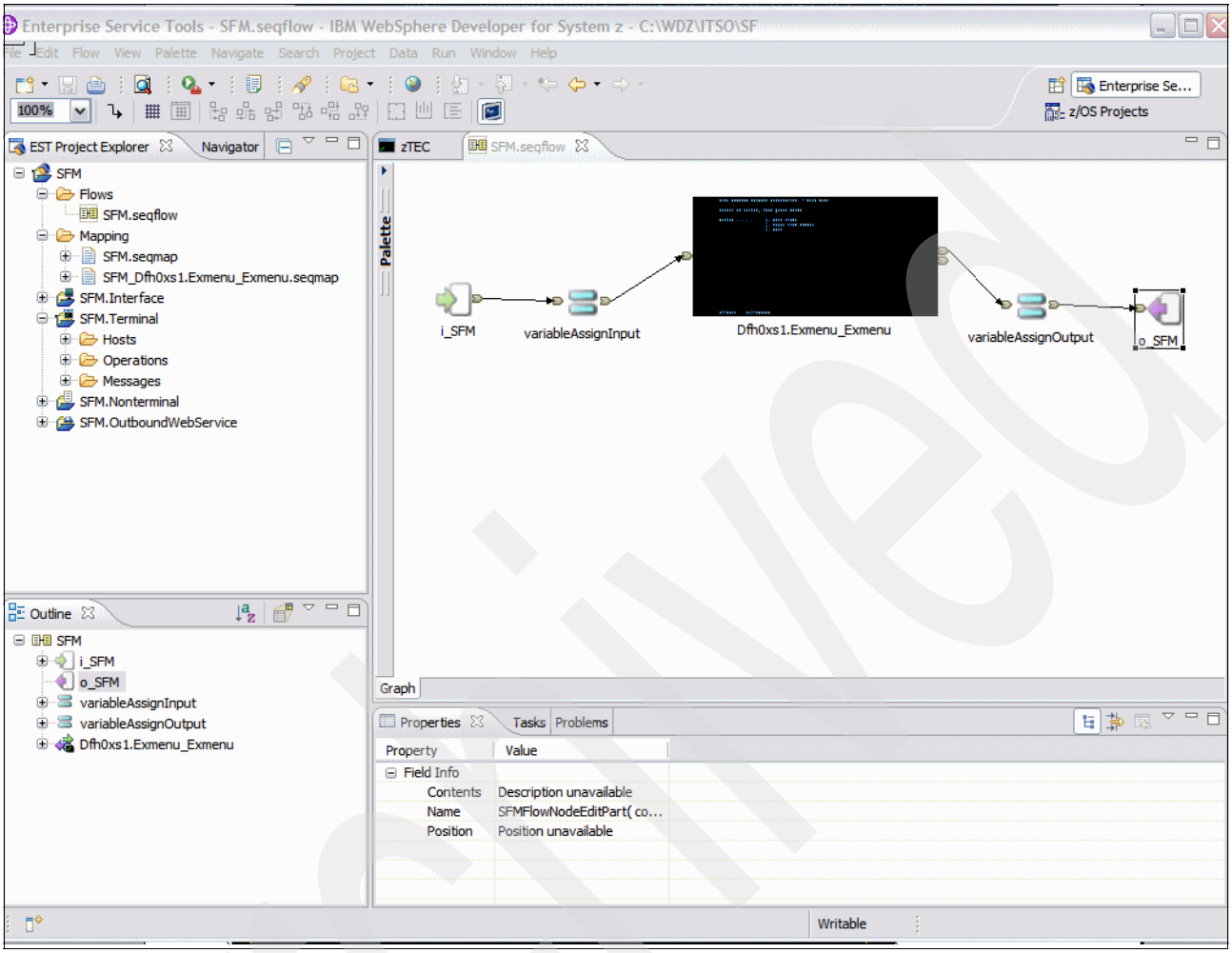


Figure 8-15 The Flow editor with terminal

You can move your cursor over the different nodes in the flow diagram to get their descriptions.

You can see a description for each node in Table 8-2.

Table 8-2 Node descriptions

Node	Type of node	Comments
i_SFM	Receive	Every flow starts with a receive node where the input/incoming message begins. This is the entry point for the flow.

Node	Type of node	Comments
variableAssignInput	Assign	An assign node is associated with mapping routines that map or transform data as it flows from one node to another. Here, the node takes the input message and does some mapping before the data is passed to the terminal invoke node.
Dfh0xs1.Exmenu_Exmenu	Invoke	There are different types of invoke nodes: nonterminal, terminal, and outbound Web service. Here we have a terminal invoke node because it is associated with screen operations.
variableAssignOutput	Assign	This assign node takes the output from the terminal invoke node and does some mapping before passing it to the reply node.
o_SFM	Reply	This is the normal exit from the flow where any output/outgoing message passes through.

Now that the flow is designed, let us generate the runtime code.

8.4 Runtime code generation and deployment

We create the generation properties file that will be used to generate the runtime artifacts needed to deploy and run the service flow application on the CICS TS V3.1 Service Flow Runtime. Do these steps:

1. Select **SFM.seqflow** in the EST Project Explorer view, right-click it, and select **New Generation Properties File**.

Archive

2. In the New Generation Properties window, specify the name SFM_generation-props.wsdl. Make sure the **Validate flows against runtimes** box is checked, as shown in Figure 8-16.

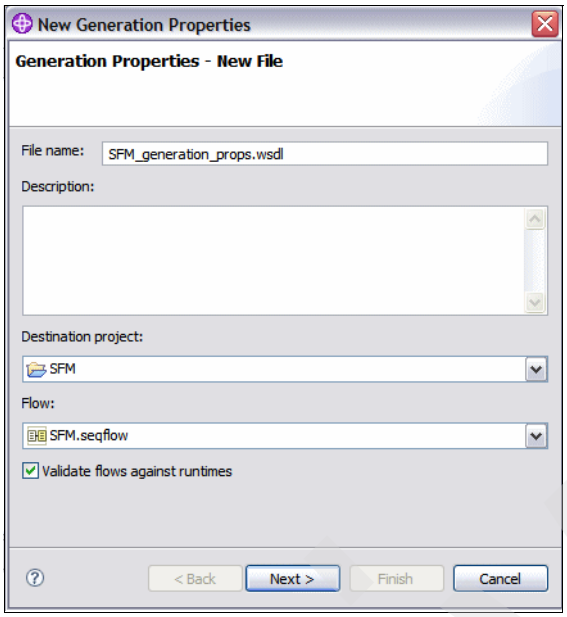


Figure 8-16 Generation properties

3. Click **Next**.
4. In the Generation Properties - Runtime selection window, select **CICS Service Flow Runtime** as the target run time and click **Finish**.
5. In the Generation Properties editor, click the Flow Type drop-down menu and select **Link3270 Bridge**. Input the rest of the specifications, as shown in Figure 8-17 on page 221.

Archived

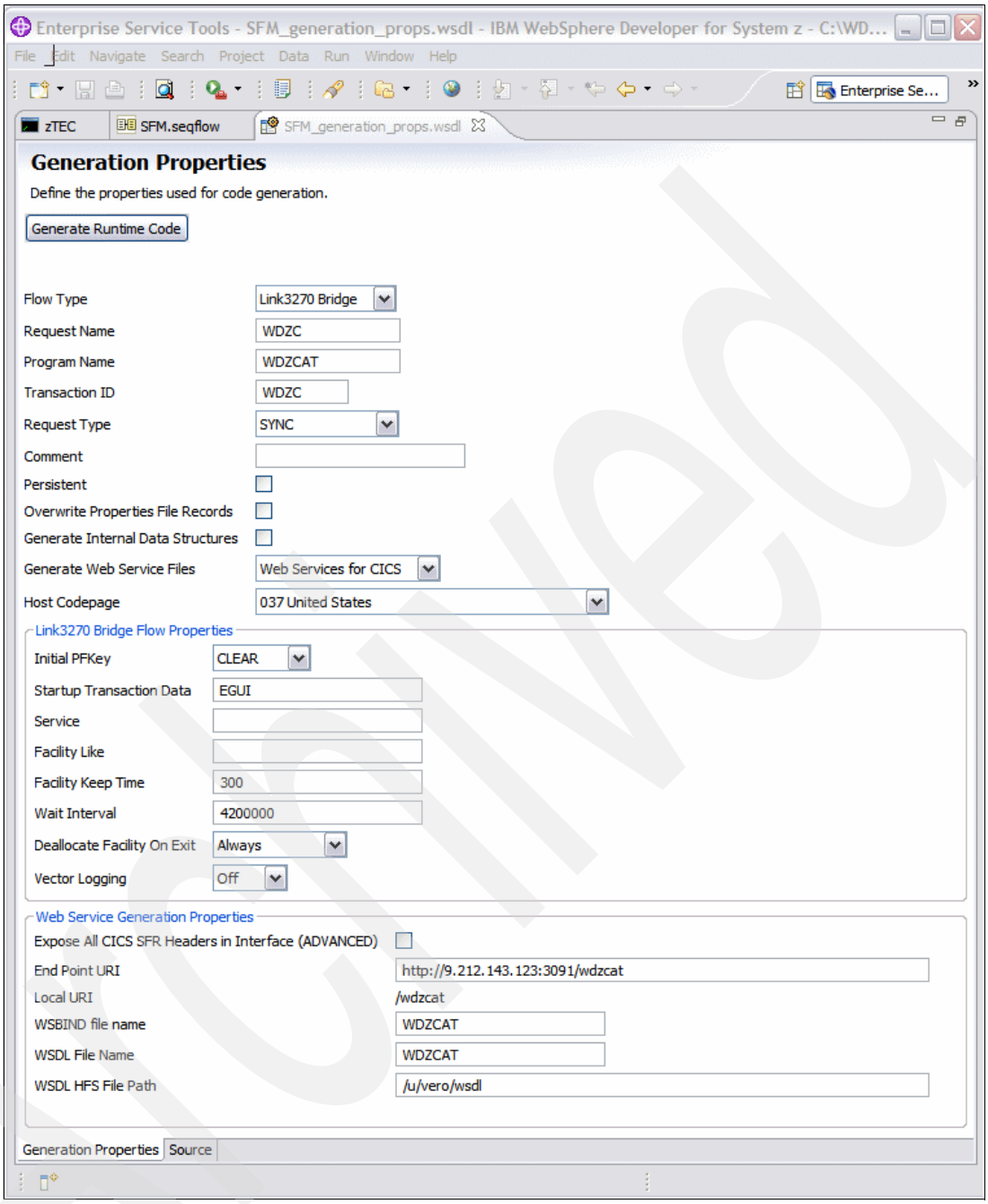


Figure 8-17 The Generation Properties specifications

6. Press Ctrl+S to save your changes.

Notice that the Generate Runtime Code button is now active (go to the top).

We will now generate the runtime artifacts needed to deploy a service flow application. These artifacts include COBOL programs, copybooks, and the jobs needed to define resources and build the service flow application. We have the option of generating these artifacts directly to the target z/OS system to facilitate the build and deployment. This will also ensure that CICS Web services artifacts like the WSBind file is copied correctly.

Archived

Do these steps:

1. Click the Generate Runtime Code button.

   In the Generate run time code dialog, we have two options:

   – Generate the runtime code directly on the mainframe.

     i. Make sure the **Deploy to remote target location** box is checked, as shown in Figure 8-18.

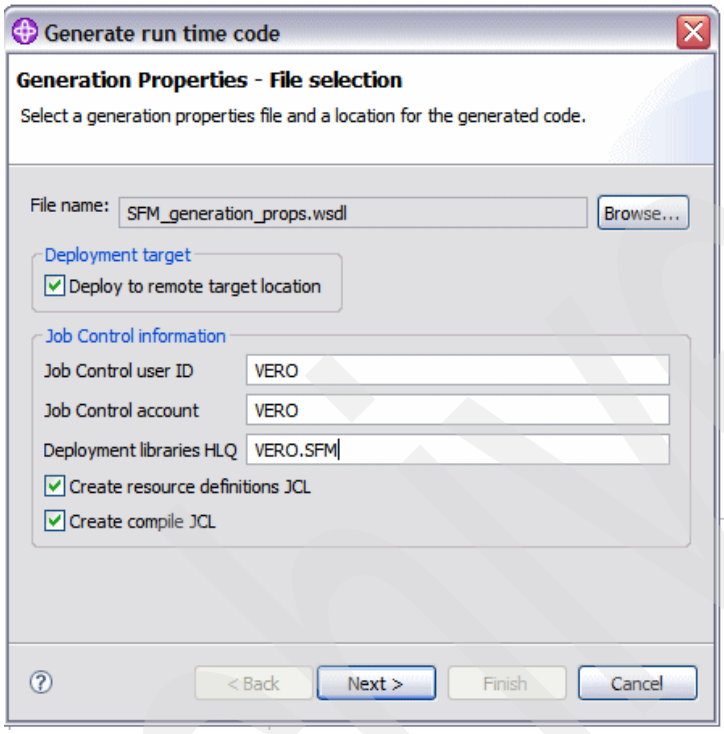


Figure 8-18 Generation Properties - deploy to remote target location

     ii. Click **Next**.

Archived

iii. In the Remote deployment selection window, specify the MVS and the UNIX System Services directories where you want to deploy, as shown in Figure 8-19.

Notice that the MVS and the UNIX System Services directories must already exist.

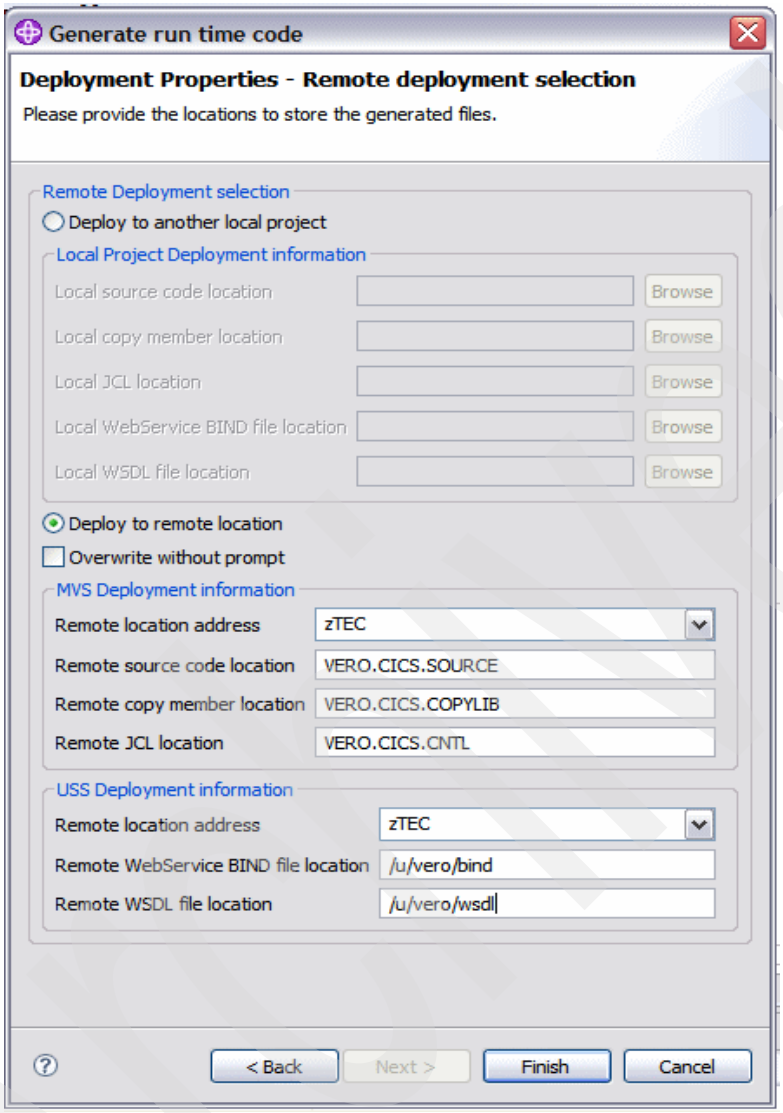


Figure 8-19 Specify the remote deployment locations

iv. Click **Finish**.

The generation process starts. You should get a completion message, as shown in Figure 8-20.

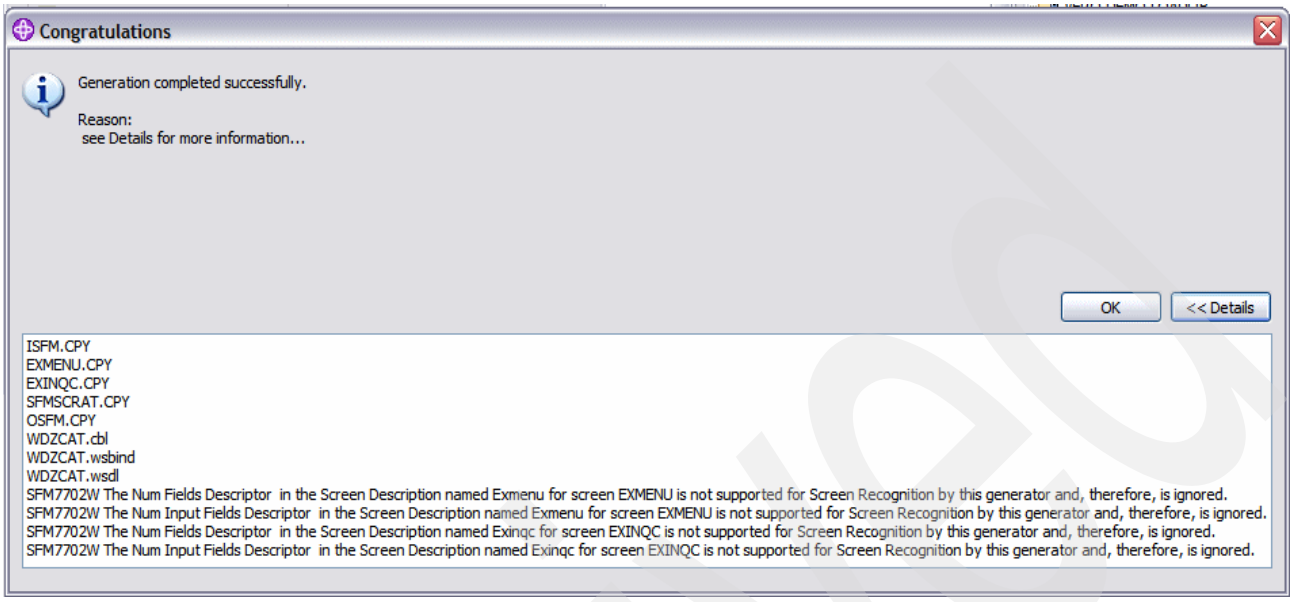


Figure 8-20 Generation successful message

– Generate the runtime code locally on the workbench.

i. Make sure the Deploy to remote target location box is not checked, as shown in Figure 8-21.

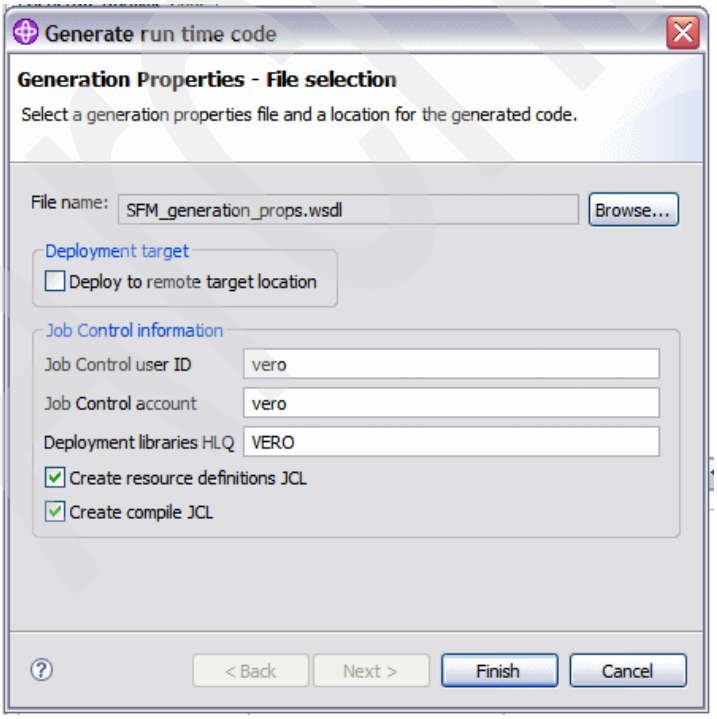


Figure 8-21 Generation Properties - deploy locally

ii. Click **Finish**.

You should get a completion message, as shown in Figure 8-22.

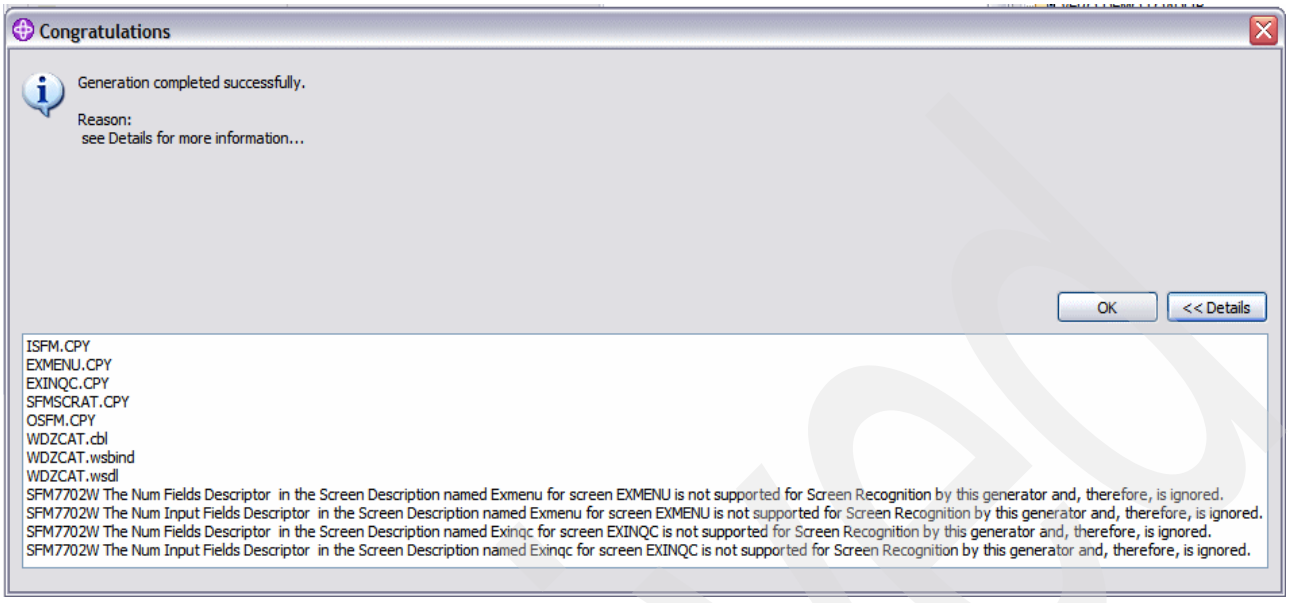


Figure 8-22 Generation successful message

Let us look at the generated artifacts, as shown in Table 8-3.

Table 8-3 Description of the generated artifacts

Generated artifact	Description
#WDZC.jcl	A job that defines the appropriate program, transaction, and process type resource to the target CICS.
@WDZC.jcl	A job that updates the CICS Service Flow Runtime properties file with information about the new flow application.
WDZCAT.cbl	A COBOL program that represents the flow application.
WDZCAT.jcl	A job that compiles and linkedits WDZCAT into a load module.
WDZCAT;wsbind	A CICS Web service WSBind file that describes the Web service interface to WDZCAT.
WDZCAT.wsdl	A WSDL file that describes the CICS based Web service.

Switch to a z/OS projects perspective to be able to work with our MVS and UNIX System Services systems.

If you choose to deploy locally, then you now have to copy the JCLs and the program generated to MVS. You also have to copy the WSBind file to a CICS Web Services pickup directory. Figure 8-23 shows the generated files.

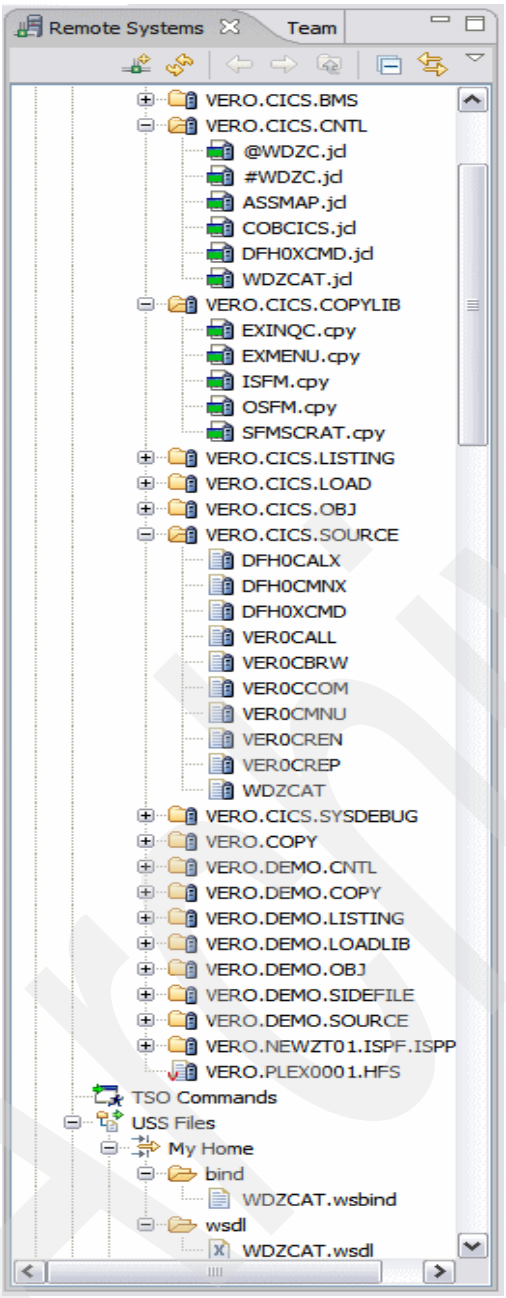


Figure 8-23 The generated artifacts in the Remote Systems View

For both the remote and the local deployment, we have to:

- Compile and linkedit the WDZCAT program by submitting the WDZCAT JCL.
- Update the CICS Service Flow runtime properties file by submitting the @WDZC JCL.
- Define the program, transaction, and process type to CICS by submitting the #WDZC JCL.
- Activate the newly defined CICS resources using CEMT I NEW commands.

Archived

- Pick up the new WEBSERVICE and URIMAP resources associated with WDZCAT by performing a CICS pipeline scan.

8.5 Test the deployed service

Do these steps:

1. Select **WDZCAT.wsdl** in the EST Project Explorer view, right-click it, and select **Web Services** → **Test with Web Services Explorer**, as shown in Figure 8-24.

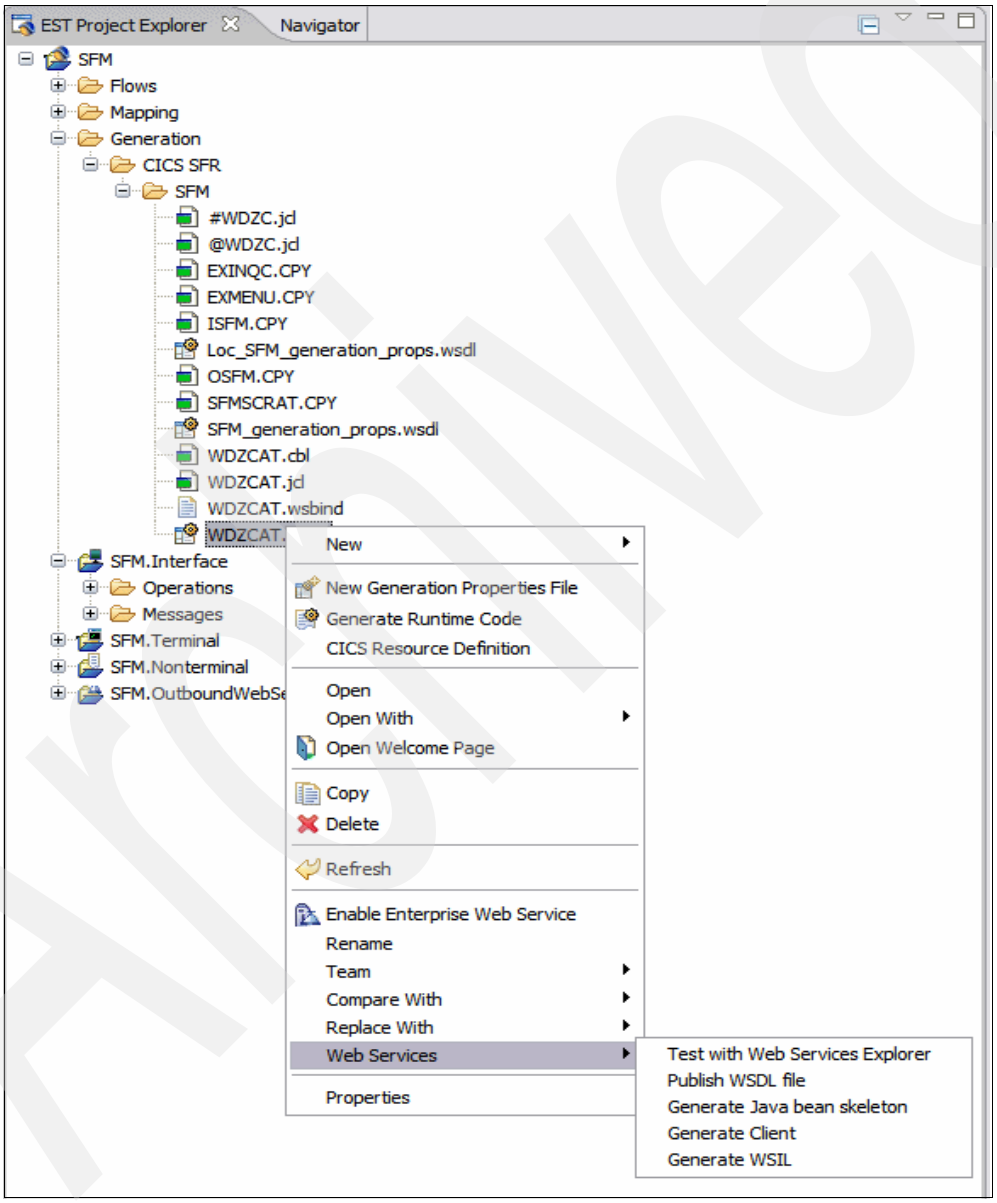


Figure 8-24 Activate the Test with Web Services Explorer

This should launch the Web Services Explorer.

Archived

2. Double-click the Web Services Explorer blue title to maximize this window. You should see something similar to Figure 8-25.

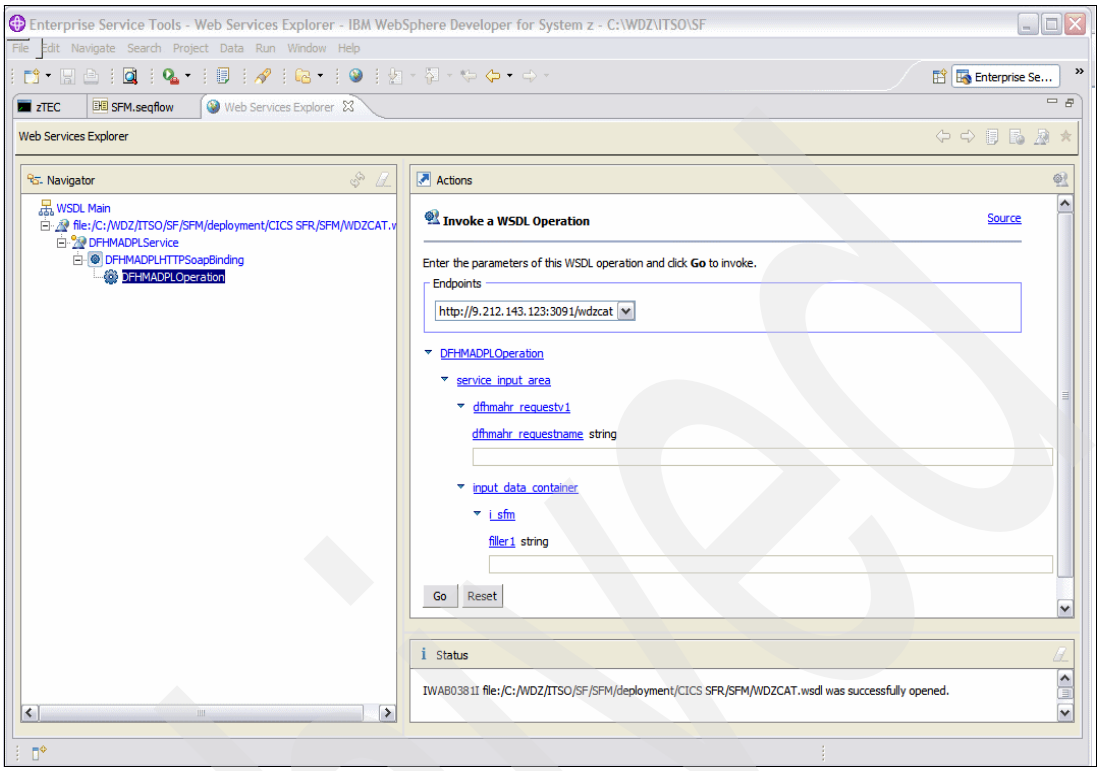


Figure 8-25 The Web services Explorer

3. Specify a value of WDZC for dhfmahr_requestname and click **Go**.

   You should see something similar to Figure 8-26 on page 229 in the Status pane of the Web Services Explorer.

Archive

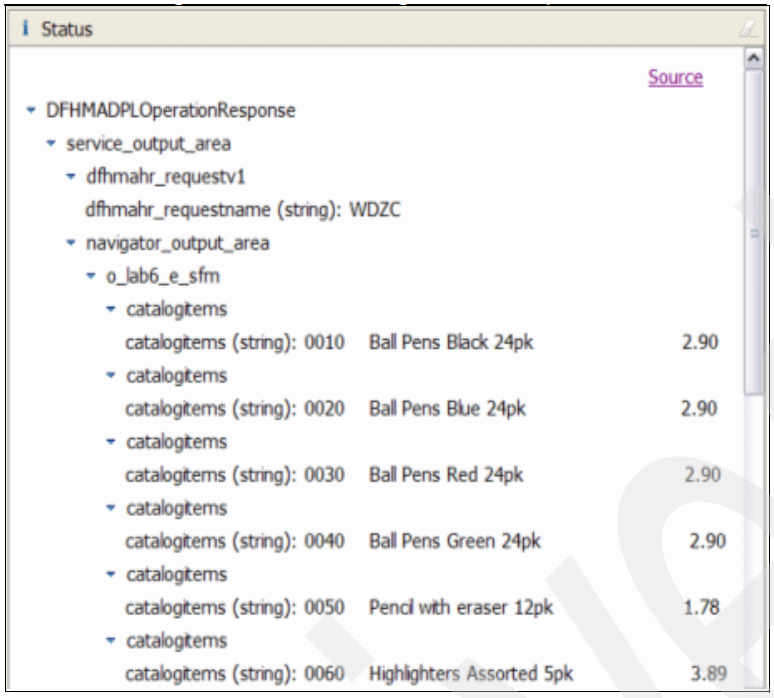


Figure 8-26 Service Flow Response

This is the easiest way to test the Service Flow from the development without developing any Client code.

Archived

Archived

9

Menu Manager

In this chapter, we introduce the Menu Manager feature provided in WebSphere Developer for System z.

The Menu Manager provides a graphical interface from the workbench Preferences window that you can use to create menus and actions related to subprojects and files.

This chapter provides a sample of using the Menu Manager to describe the different steps to implement a new action in a pop-up menu. The action will send a TSO command to MVS to execute. The action will then send back the result in a pop-up window.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

9.1 Menu Manager Preferences window

To access the Menu Manager Preferences window, select **Window** → **Preferences** in the workbench to open the Preferences window. In the left navigation pane, double-click the **Menu Manager** node to reveal the tree of available Menu Manager preferences.

The Actions and Menus preferences window can be used to create new action that can be local or remote, as shown in Figure 9-1.

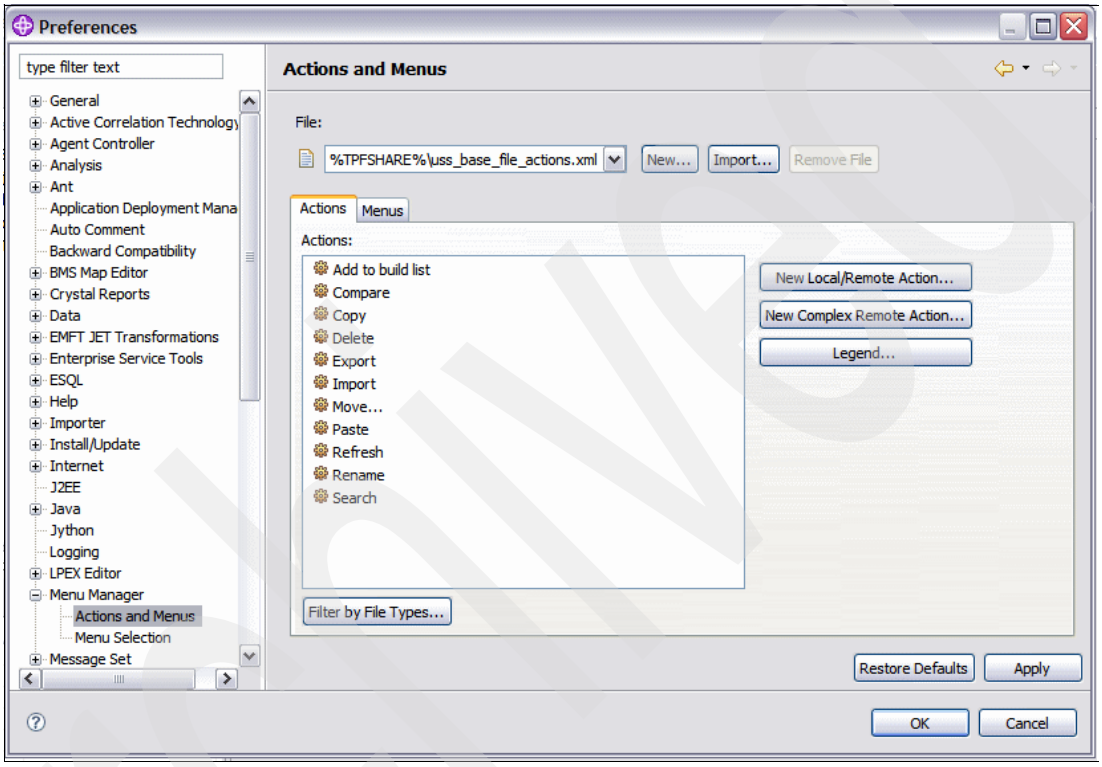


Figure 9-1 The Actions and Menus preferences window

The Menu Selection preferences window (see Figure 9-2 on page 233) is used to attach pop-up menus to subprojects and files in the Navigator view.

The three following tabs can be selected:

- MVS Projects
- z/OS LPEX Editor
- UNIX System Services Subprojects

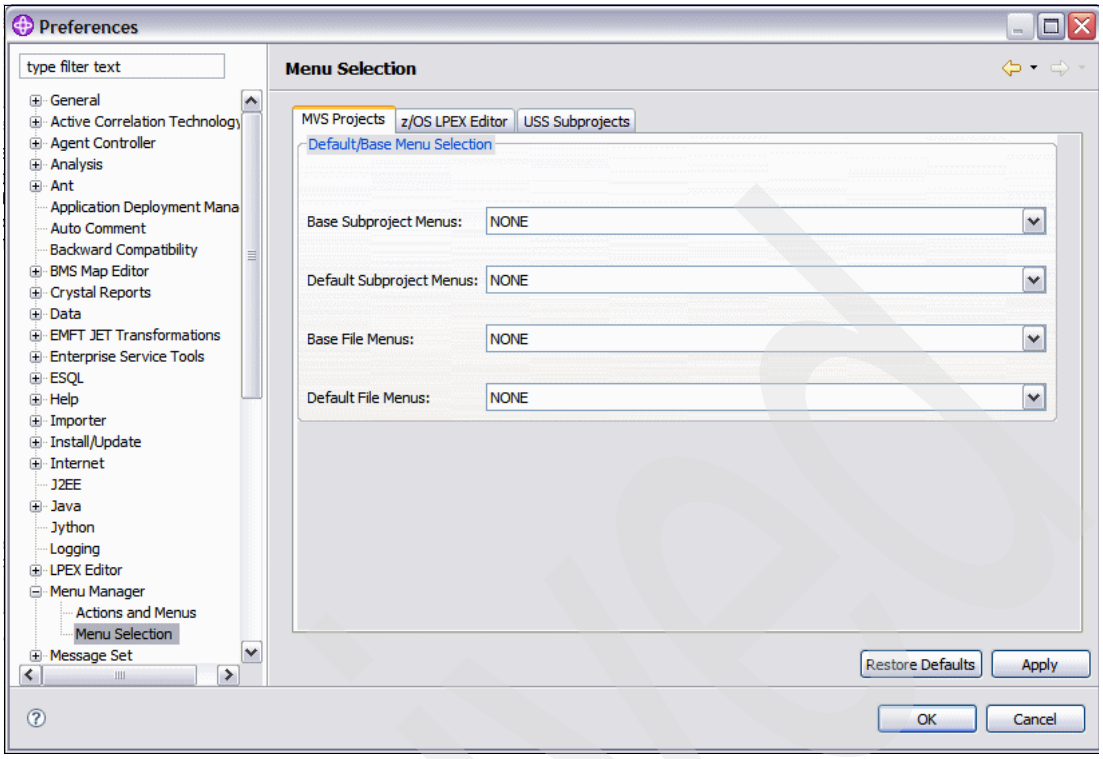


Figure 9-2 The Menu Selection preference window

Make sure that the MVS Projects tab is selected, and click **OK**.

Archive

9.2 Creation of custom menu items

Here we customize the pop-up menu in the z/OS Project view to add an action that executes TSO commands. Do these steps:

1. In the Actions and Menus preference window, click **File New...** to create a specific custom menu, ash shown in Figure 9-3.

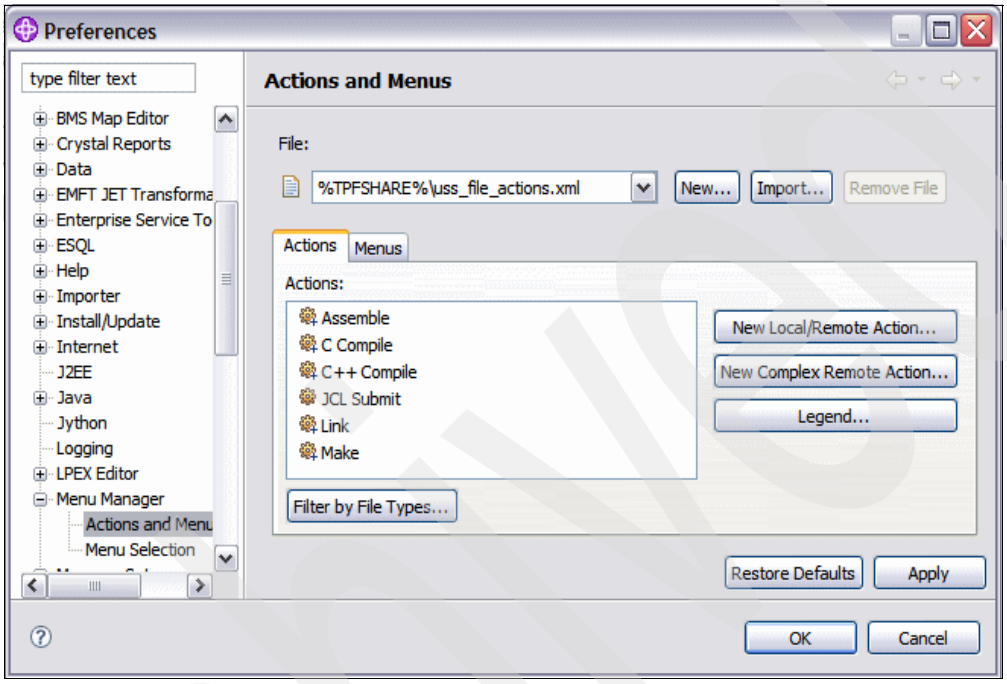


Figure 9-3 Create a new file from the Actions and Menus preference window

2. From the New File window, select **File** as the Resource Type. This file is an XML file that contains all the useful information that will be gathered from the wizards.
3. Enter a location for the new file, as shown in Figure 9-4 on page 235.

Archive

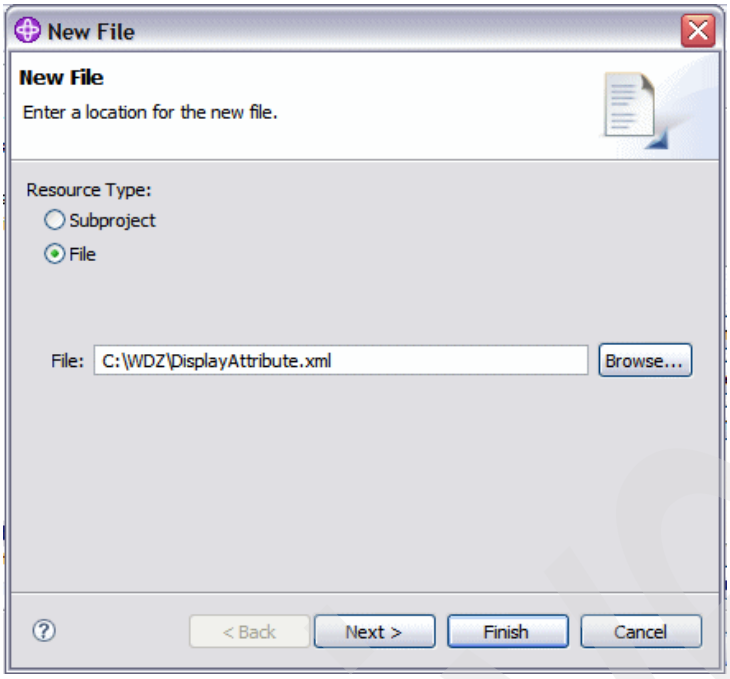


Figure 9-4 Specify a location for the new file

4. Click **Next**.
5. In the Create file content window, select **Create a new local/remote action**, as shown in Figure 9-5.

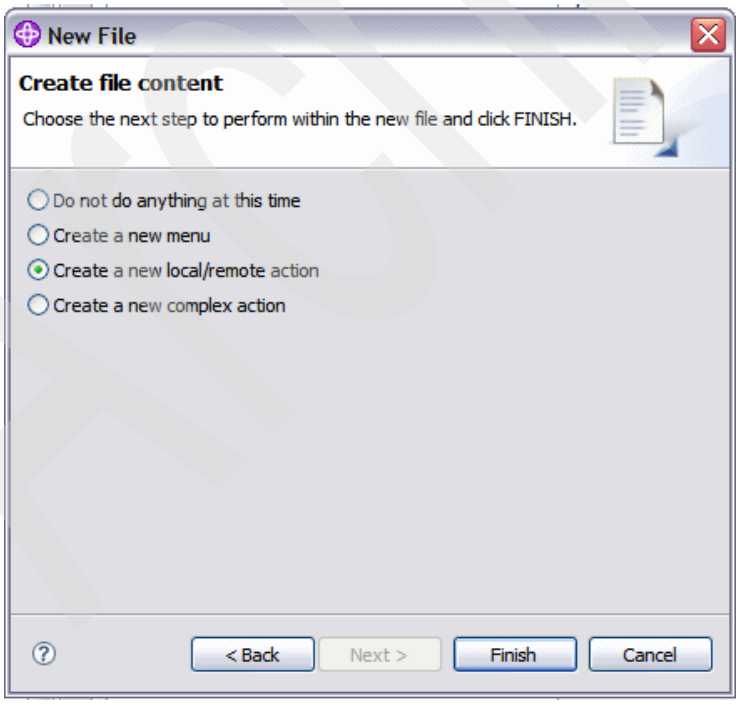


Figure 9-5 The Create File content window

6. Click **Finish**. This opens the New Action Page window, where we have to give a name to the new action that we are defining.

7. Type Display Attribute in the Name field, as shown in Figure 9-6. This is the command value that will appear in the pop-up menu.

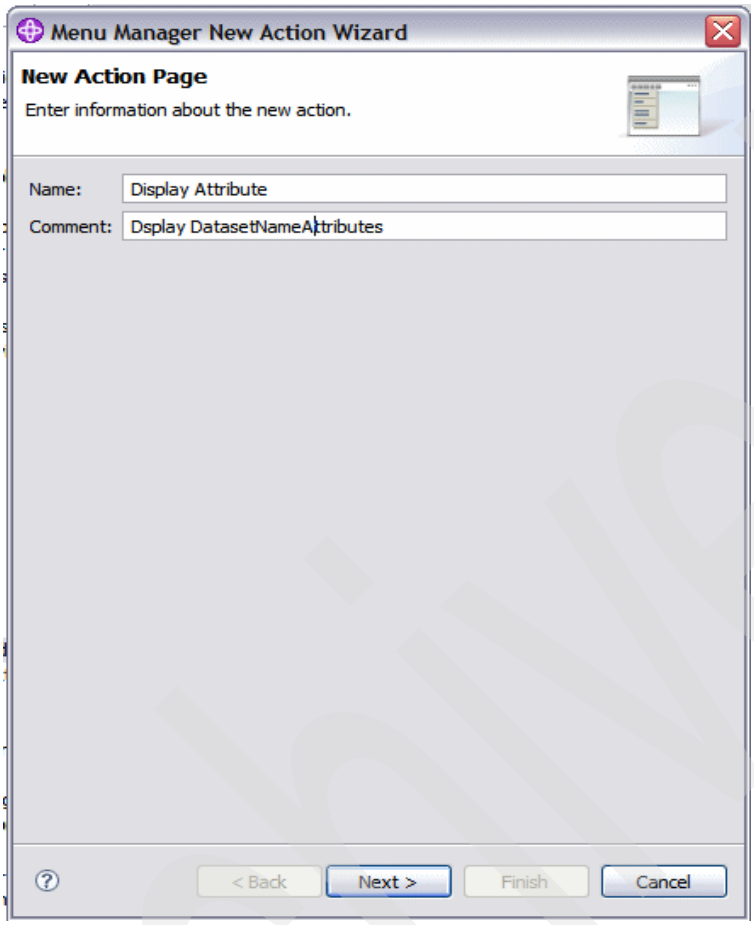


Figure 9-6 New Action Page

8. Click **Next**.
9. In the Files Associations window, select the files extensions that you want the new action to be associated with. As we want to get the attributes for MVS data sets, we select the following extensions, as shown in Figure 9-7 on page 237.
   - *.bms
   - *.cbl
   - *.cob
   - *.cpy
   - *.inc
   - *.jcl

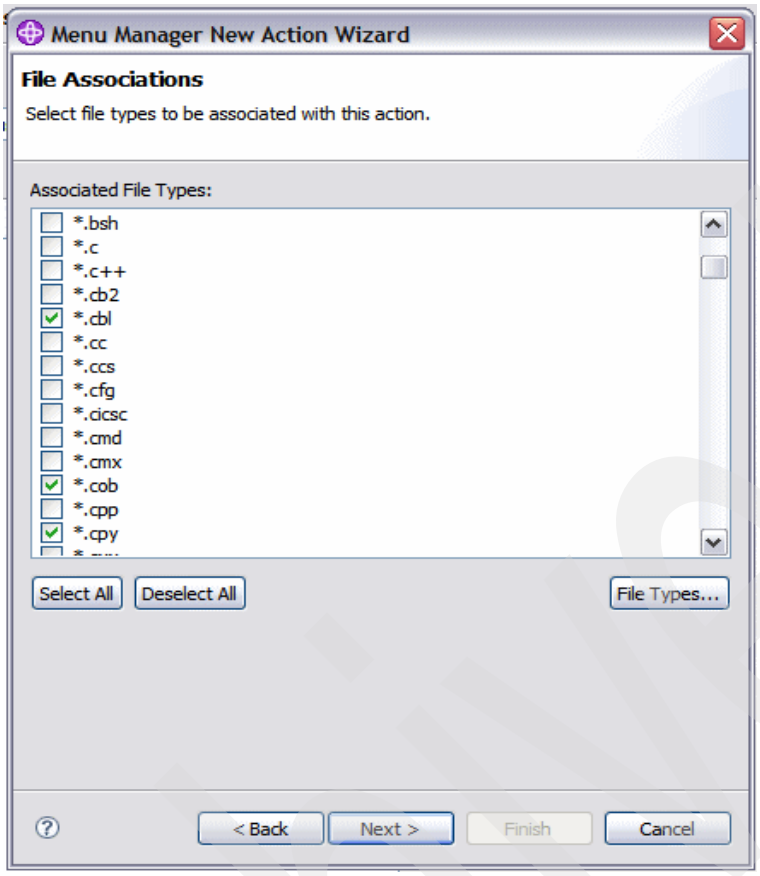
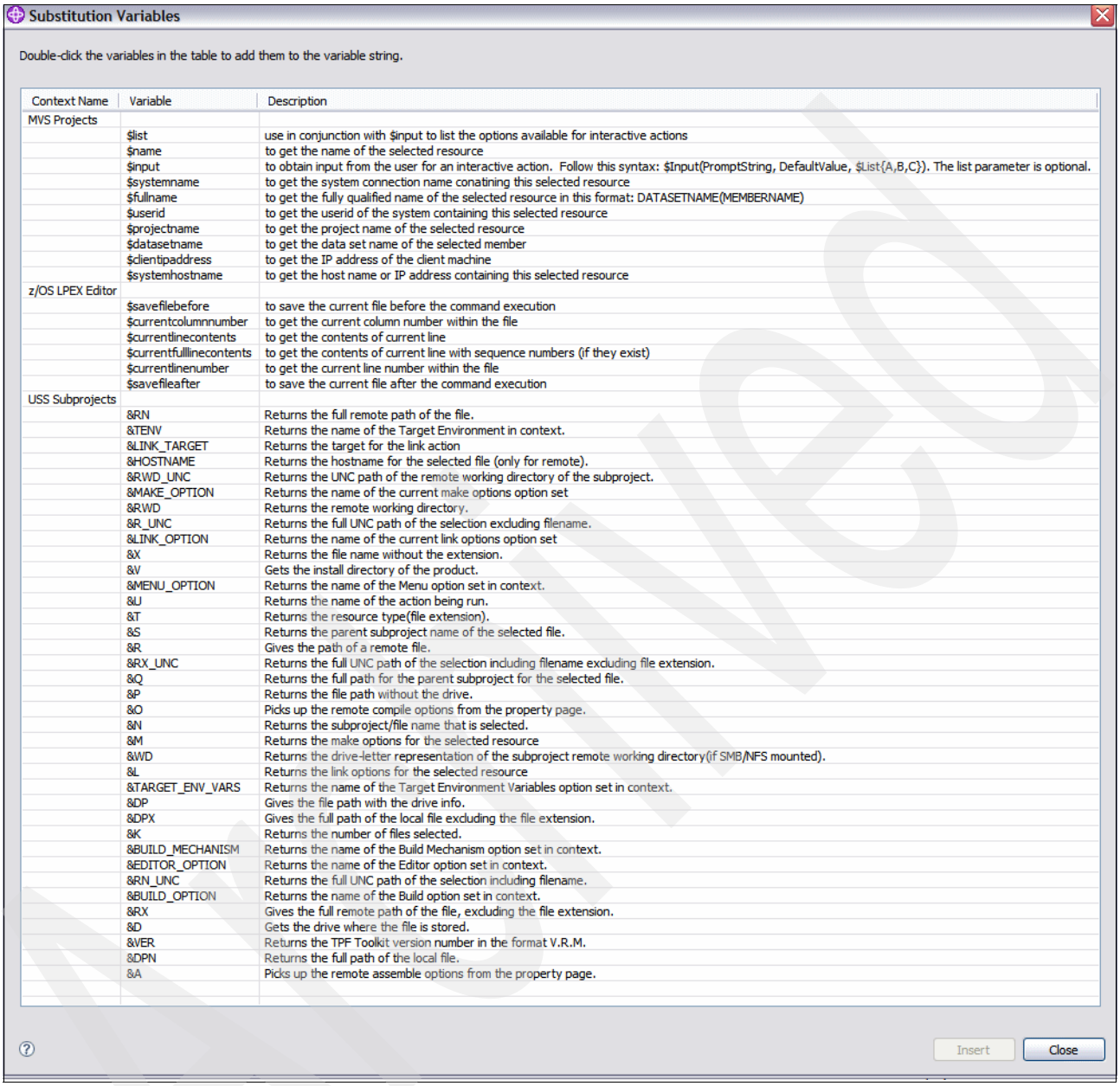


Figure 9-7 The Files Associations window

10. Click **Next**.
11. In the Run Options window, on the Basic tab, type LISTDS as a command value.
12. Click **Variable**.

Archived

13. From the context name MVS Projects, select the variable **$fullname**. You can see the list of available variables in Figure 9-8.



Context Name	Variable	Description
MVS Projects		
	$list	use in conjunction with $input to list the options available for interactive actions
	$name	to get the name of the selected resource
	$input	to obtain input from the user for an interactive action. Follow this syntax: $Input(PromptString, DefaultValue, $List{A,B,C}). The list parameter is optional.
	$systemname	to get the system connection name conatining this selected resource
	$fullname	to get the fully qualified name of the selected resource in this format: DATASETNAME(MEMBERNAME)
	$userid	to get the userid of the system containing this selected resource
	$projectname	to get the project name of the selected resource
	$datasetname	to get the data set name of the selected member
	$clientipaddress	to get the IP address of the client machine
	$systemhostname	to get the host name or IP address containing this selected resource
z/OS LPEX Editor		
	$savefilebefore	to save the current file before the command execution
	$currentcolumnnumber	to get the current column number within the file
	$currentlinecontents	to get the contents of current line
	$currentfulllinecontents	to get the contents of current line with sequence numbers (if they exist)
	$currentlinenumber	to get the current line number within the file
	$savefileafter	to save the current file after the command execution
USS Subprojects		
	&RN	Returns the full remote path of the file.
	&TENV	Returns the name of the Target Environment in context.
	&LINK_TARGET	Returns the target for the link action
	&HOSTNAME	Returns the hostname for the selected file (only for remote).
	&RWD_UNC	Returns the UNC path of the remote working directory of the subproject.
	&MAKE_OPTION	Returns the name of the current make options option set
	&RWD	Returns the remote working directory.
	&R_UNC	Returns the full UNC path of the selection excluding filename.
	&LINK_OPTION	Returns the name of the current link options option set
	&X	Returns the file name without the extension.
	&V	Gets the install directory of the product.
	&MENU_OPTION	Returns the name of the Menu option set in context.
	&U	Returns the name of the action being run.
	&T	Returns the resource type(file extension).
	&S	Returns the parent subproject name of the selected file.
	&R	Gives the path of a remote file.
	&RX_UNC	Returns the full UNC path of the selection including filename excluding file extension.
	&Q	Returns the full path for the parent subproject for the selected file.
	&P	Returns the file path without the drive.
	&O	Picks up the remote compile options from the property page.
	&N	Returns the subproject/file name that is selected.
	&M	Returns the make options for the selected resource
	&WD	Returns the drive-letter representation of the subproject remote working directory(if SMB/NFS mounted).
	&L	Returns the link options for the selected resource
	&TARGET_ENV_VARS	Returns the name of the Target Environment Variables option set in context.
	&DP	Gives the file path with the drive info.
	&DPX	Gives the full path of the local file excluding the file extension.
	&K	Returns the number of files selected.
	&BUILD_MECHANISM	Returns the name of the Build Mechanism option set in context.
	&EDITOR_OPTION	Returns the name of the Editor option set in context.
	&RN_UNC	Returns the full UNC path of the selection including filename.
	&BUILD_OPTION	Returns the name of the Build option set in context.
	&RX	Gives the full remote path of the file, excluding the file extension.
	&D	Gets the drive where the file is stored.
	&VER	Returns the TPF Toolkit version number in the format V.R.M.
	&DPN	Returns the full path of the local file.
	&A	Picks up the remote assemble options from the property page.

Figure 9-8 The list of variables that can be used

The final command should be LISTDS '$fullname.'

The same result would be obtained by the command LISTDS '$datasetname('$name)'.

14. In the Action Properties window, check **Use Existing Action** and from the drop-down menu select the action **com.ibm.tt.ui.menumanager.tsoaction**.

You can select **Show Output in dialog** in order to get the result of the command defined in a pop-up window. If you do not select it, then the output result will be shown in the view TPT Toolkit Console. See Figure 9-9 on page 239.

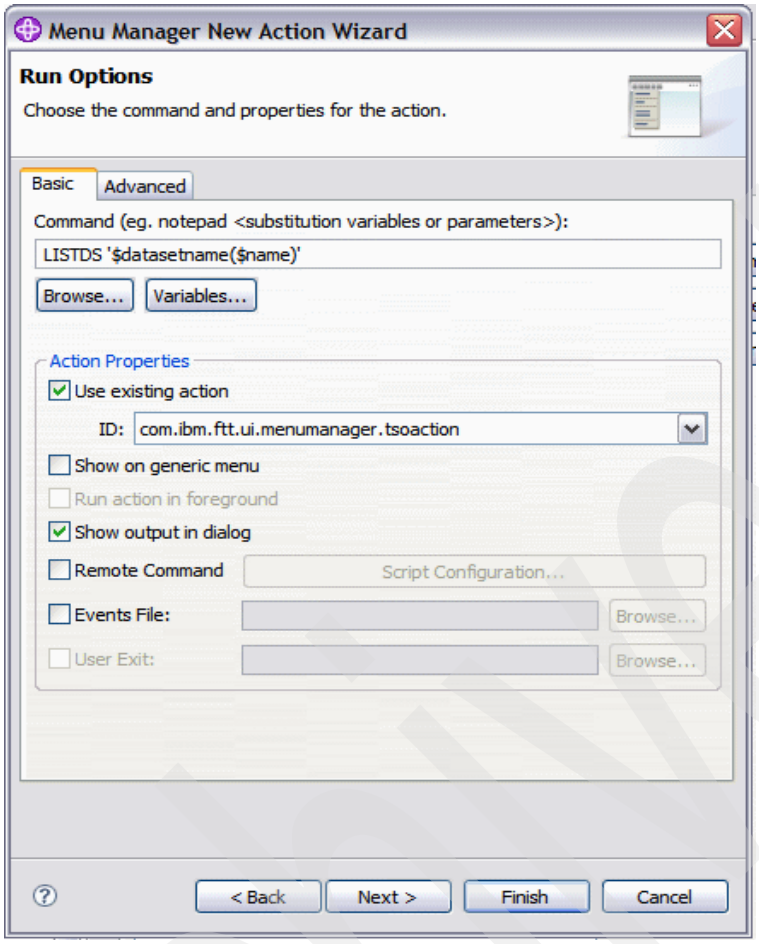


Figure 9-9 The Run Options

15. Click **Next**.

Archived

16. In the Add Actions to Menus window, select **Add to Menus** and select the **Remote Actions** check box, as shown in Figure 9-10.

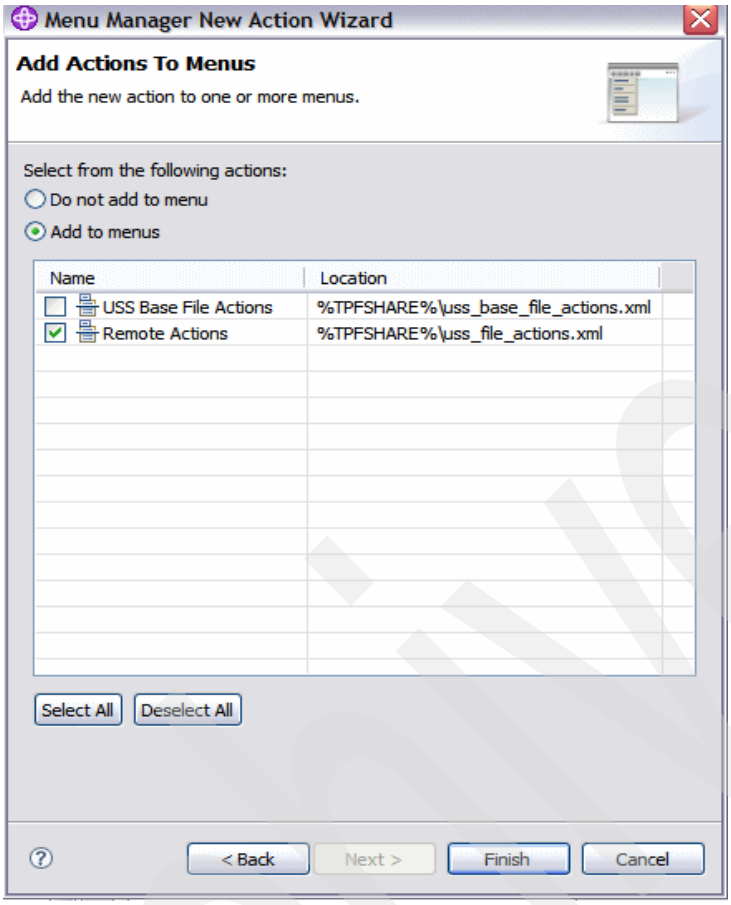


Figure 9-10 Add Actions to Menus

17. Click **Finish**.

Archived

The new action is now available from the pop-up menu for remote MVS files, as shown in Figure 9-11.

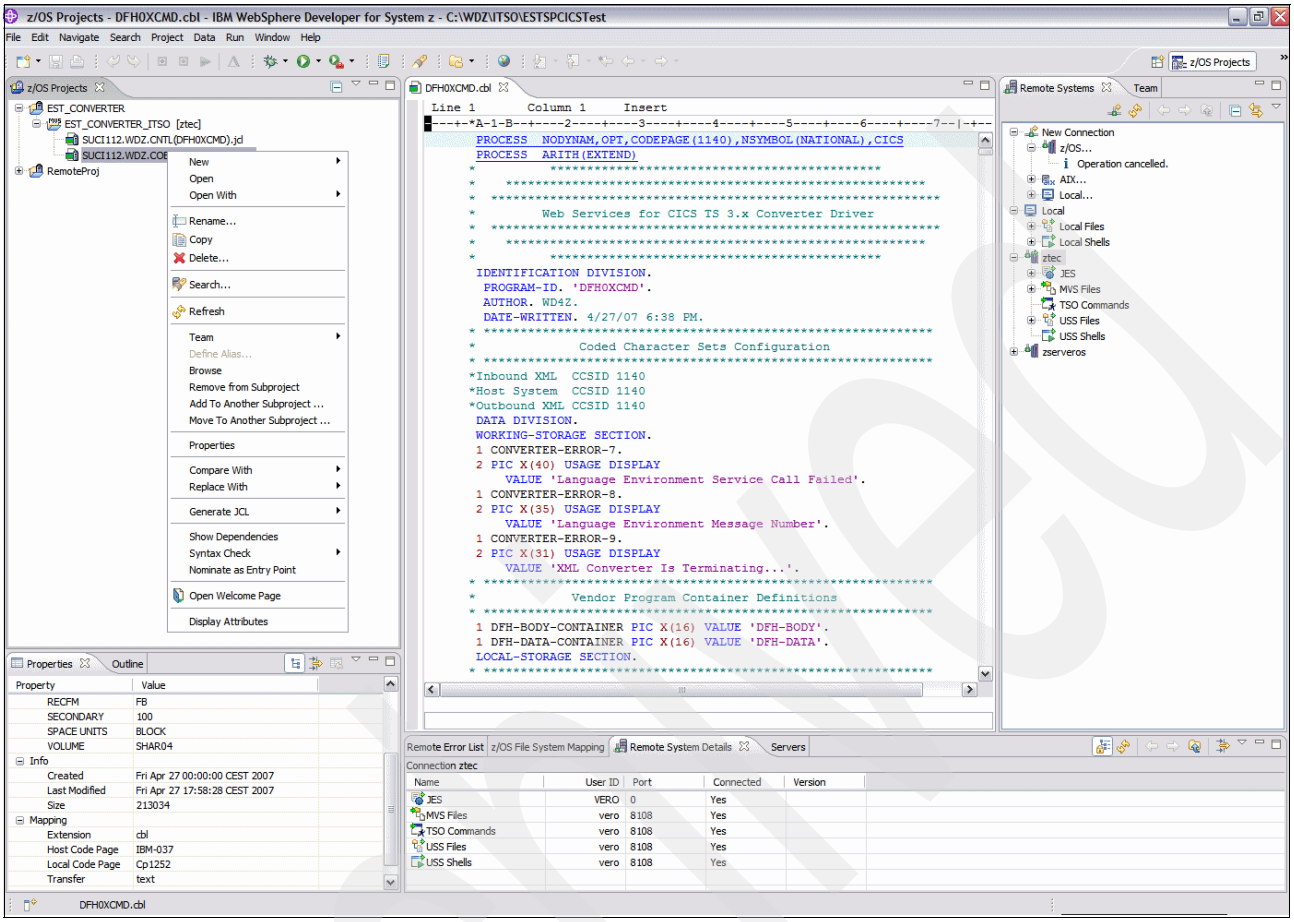


Figure 9-11 The new action available in the menu for the remote files

Archived

The new action is now ready to be executed. Click **Display Attribute**. A pop-up window appears with the output of the command. In our implementation, the pop-up window looks like the window shown in Figure 9-12.

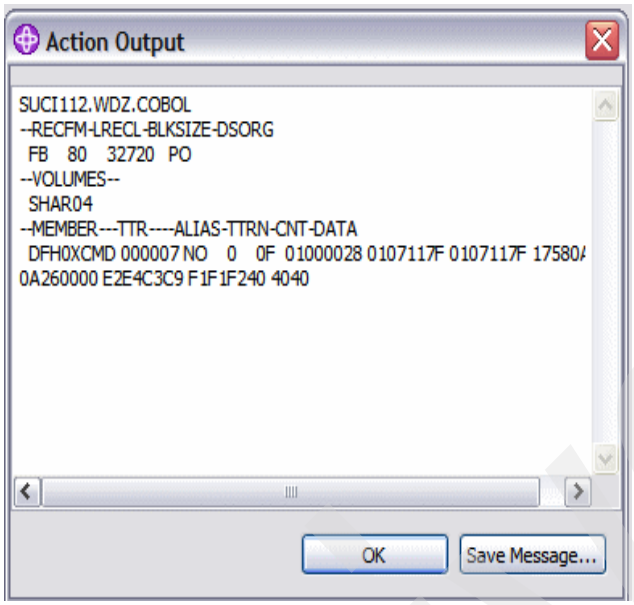


Figure 9-12 The result of the command shown in a pop-up window

You can save the output in a file by clicking the **Save Message...** button.

If you did not select **Show Output Dialog** in the Run Options window, then the result of the command is shown in the TPF Toolkit Console, as shown in Figure 9-13.

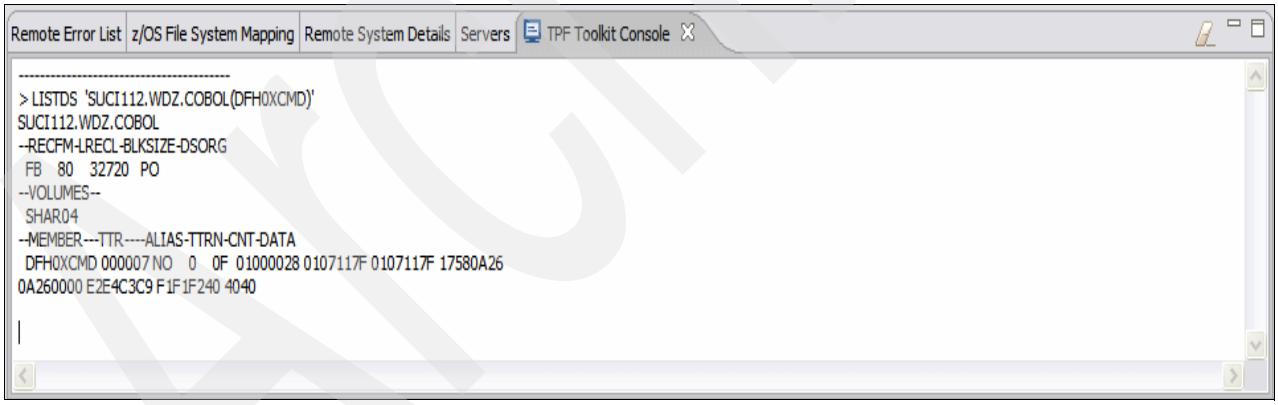


Figure 9-13 The TPF Toolkit Console

Archived

10

Core System z remote development

IBM WebSphere Developer for System z includes a tool to help developers save time by editing, compiling, and debugging host applications from their workstations.

The main features of this tool are:

- An interactive workstation-based environment to help create, maintain, and reuse applications for traditional processing or for inclusion in an SOA
- Easy access to IBM z/OS and UNIX System Services data sets
- Creation of Assembler, COBOL, PL/I, C, and C++ applications by:
  - Local and remote syntax check
  - Content assist
  - Visual BMS mapping and JCL generation capabilities
  - Visual MFS editor
  - Color coded editing
- Support for:
  - CICS
  - IMS
  - Batch
  - DB2 and DB2 stored procedures

Archived

© Copyright IBM Corp. 2009. All rights reserved.

- Debugging and testing of CICS and IMS system-based code
- Remote compile generation, building, and deployment support

Using these tools has the following advantages over traditional mainframe development:

- Faster editing: The language-sensitive editor parses the source as it is entered, giving a visual presentation of the content.
- Increased productivity: Content assist helps speed up coding and reduces typing mistakes by providing completion suggestions that include language constructs as well as user declared variables.
- Reduction of the number of compile jobs: The syntax checking facility supports eliminating syntax errors before submitting compilations to the host.
- Location of errors faster within your source files: Compile-time error messages are shown in edit sessions along side the line where the error occurred.
- Reduction of debugging effort: Significantly reduce the effort required for debugging by using the graphical debugger on the workstation while programs execute on the host.
- Capitalization on reuse: Use an existing JCL for system testing, or generate the JCL necessary for the build of your application, with the benefit of workstation organization.

Note: Refer to the SG247482-Chapter 10.zip file in Appendix B, “Additional material” on page 645 for more information.

Archived

10.1 MVS files properties

Use the MVS Files properties page to edit the properties of an MVS Files subsystem. This page contains a list of properties sets:

- Assembler Settings
- BMS Settings
- C/C++ Settings
- COBOL Settings
- Code Pages
- Application Entry Point
- JCL Job Card
- Link Options
- MFS Settings
- PL/I Settings
- Runtime Options
- Server Connection Security
- Server Launcher Settings
- Subsystem

Note: How property values cascade

Each folder in a subproject has a set of properties that are equivalent to the set of subproject properties. (For example, subprojects and folders all have a property that specifies the compiler's SYSLIB setting, even though the property may not be meaningful for a particular subproject or folder.) In addition, each file in a folder has a set of properties that are a subset of the folder properties. (Unlike file properties, folder properties may include information about linking multiple source files, for example.)

You can change the value of a subordinate property. You can indicate, for example, that the files in a specific folder be processed by a specific compiler. When you directly change the value of a subordinate property, the property is said to be distinguished. Properties values that cascade from a container (such as a folder or subproject) are inherited. For remote resources, WebSphere Developer indicates whether a property value is inherited or distinguished by marking then as follows:

- Inherited: “This value is inherited from its parent”.
- Distinguished: “This value overrides its parents' value”. Top-level resources (PDSs or sequential files in the Remote Systems view, or subprojects in the z/OS Projects view) show this message.

Changing the value of a subproject property causes the value of the equivalent property for each immediately subordinate folder to automatically change to the new value, except for folders in which the property is distinguished.

If you change the value of a folder property, the value of the equivalent property for each immediately subordinate folder or file is automatically changed to the new value.

Archived

To edit the properties, do these steps:

1. Right-click **MVS Files** under Remote Systems and select **Properties**, as shown in Figure 10-1.

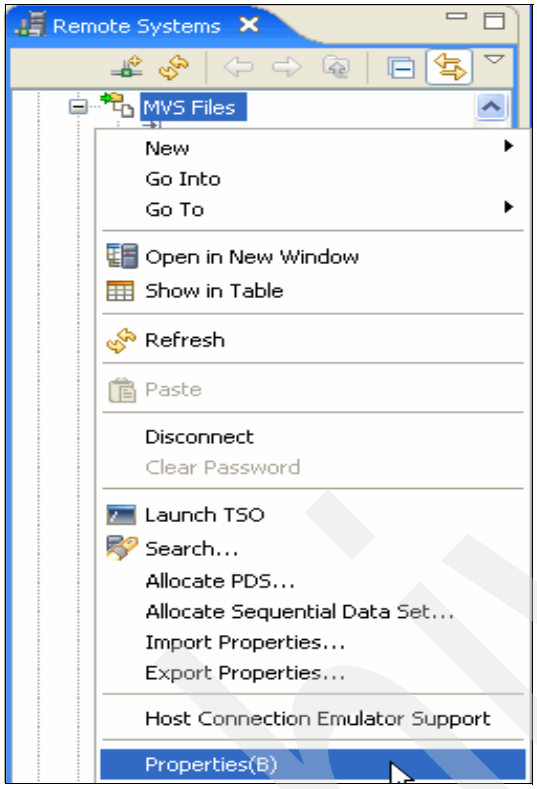


Figure 10-1 MVS Files Properties

Archived

2. Review the JCL Job Card, as shown in Figure 10-2.

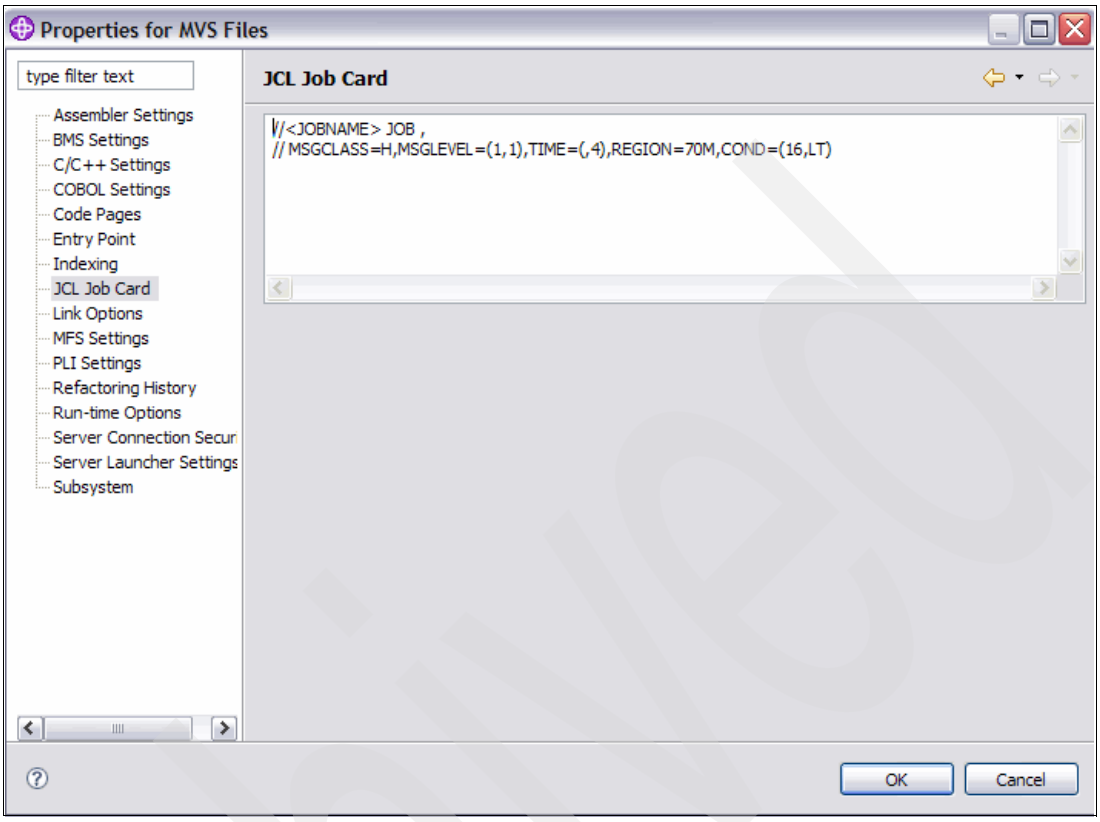


Figure 10-2 JCL Job Card

Archive

3. Review the COBOL Settings, as shown in Figure 10-3 and Figure 10-4 on page 249.

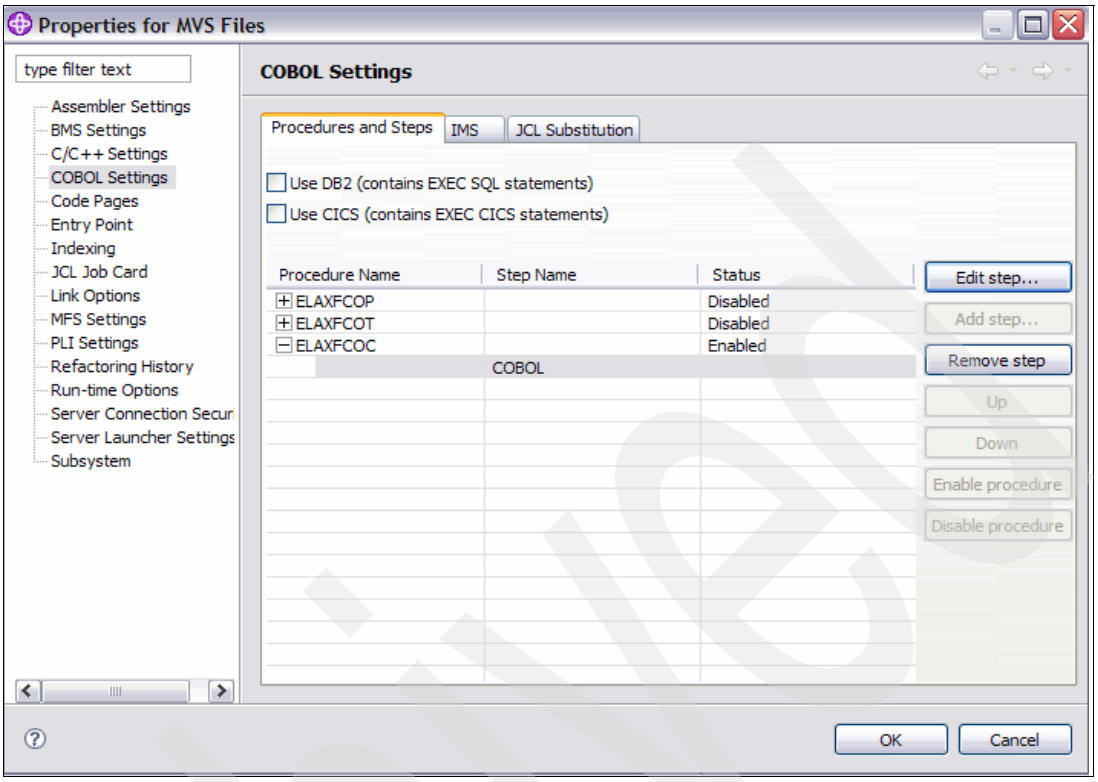


Figure 10-3 Edit COBOL compile step

Archived

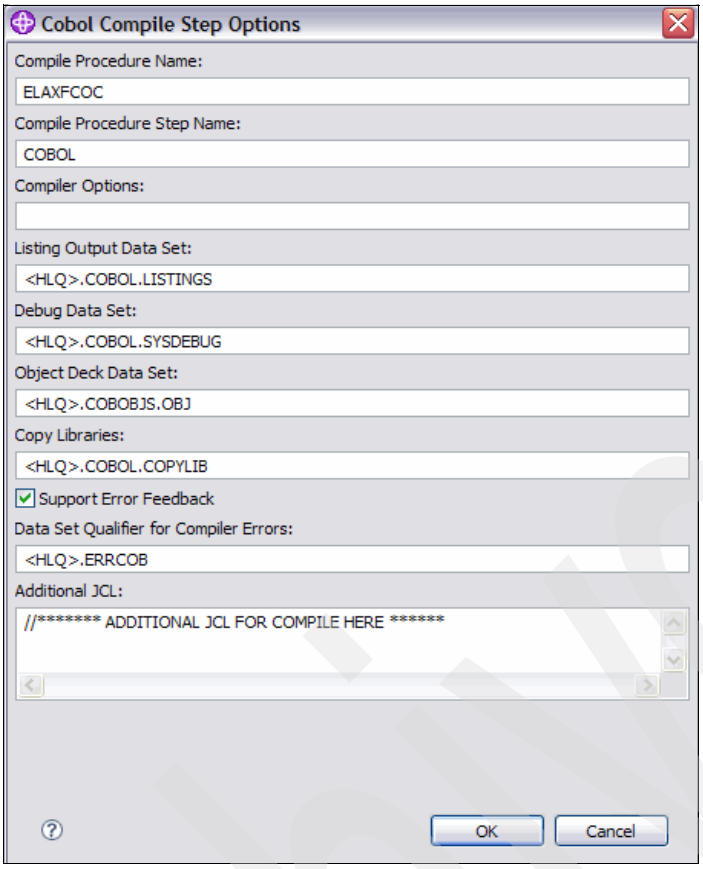


Figure 10-4 COBOL Compile Step Options

Archived

4. Review the PLI Settings as shown in Figure 10-5 and Figure 10-6 on page 251.

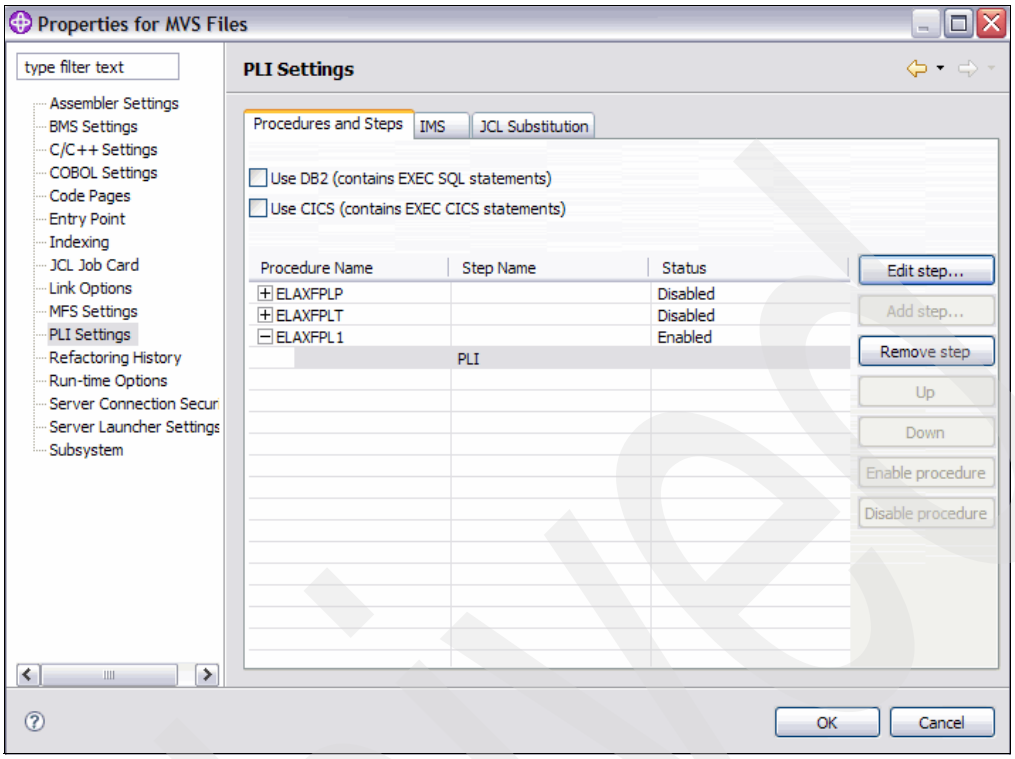


Figure 10-5 Edit PLI compile step

Archive

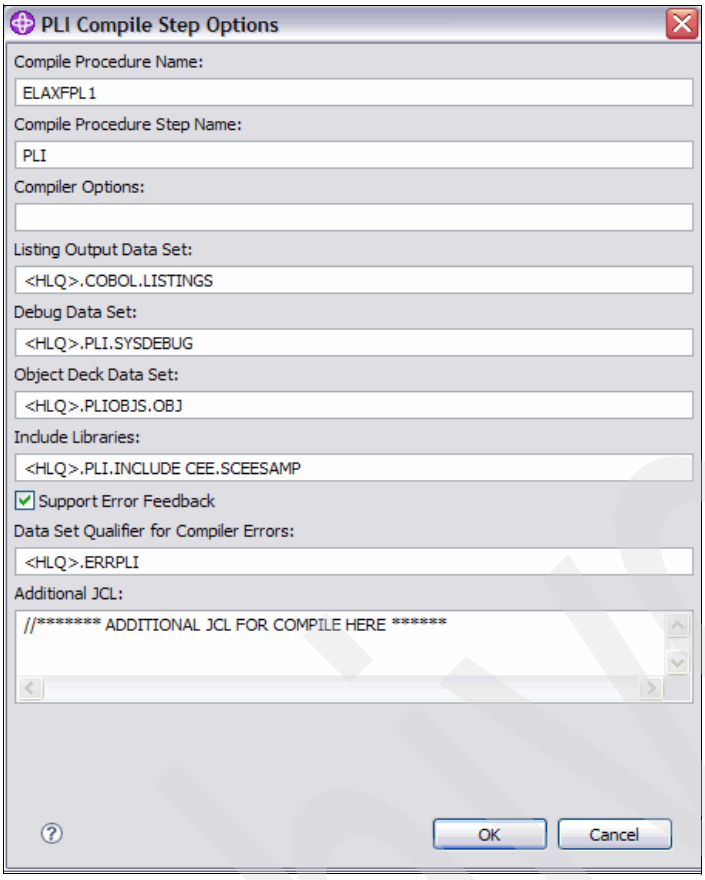


Figure 10-6 PLI Compile Step Options

10.2 ISPF editing versus LPEX editing

LPEX is an extensible editor framework that you can use to edit a variety of file types.

The z/OS LPEX Editor provides the following features:

- Auto comment, which inserts a custom comment into each line that has been modified in a source file.
- Live or selective syntax checking for High Level Assembler, C, and C++ for early identification of lexical errors.
- Syntax assist, which completes partially-typed language constructs.
- Parsed source highlighting of language constructs in distinct colors and fonts.
- Filtering the view of the text by function, instruction type, comment lines, or flow-of-control statements.
- Ability to add macros to the assembler macro table.
- High Level Assembly Language Parser (HLAsm parser).
- Online language-sensitive help for C, C++, and High Level Assembler.
- The Save As command to save documents to a remote location.
- Ability to preset breakpoints in C, C++, and Assembly source files before debugging.

Archived

- Error marker support in source files that have build errors; these markers have hover-over help.
- Line break conversions (DOS to UNIX and UNIX to DOS).
- Sabre Talk Language Parser.

10.2.1 ISPF profile within LPEX Editor

To change the profile, do these steps:

1. Select **Window** → **Preferences**. Select **ispf** on the Preferences page for the LPEX Editor, as shown in Figure 10-7.

> **Note:** There are other profiles available (XEDIT, SEU, and so on).

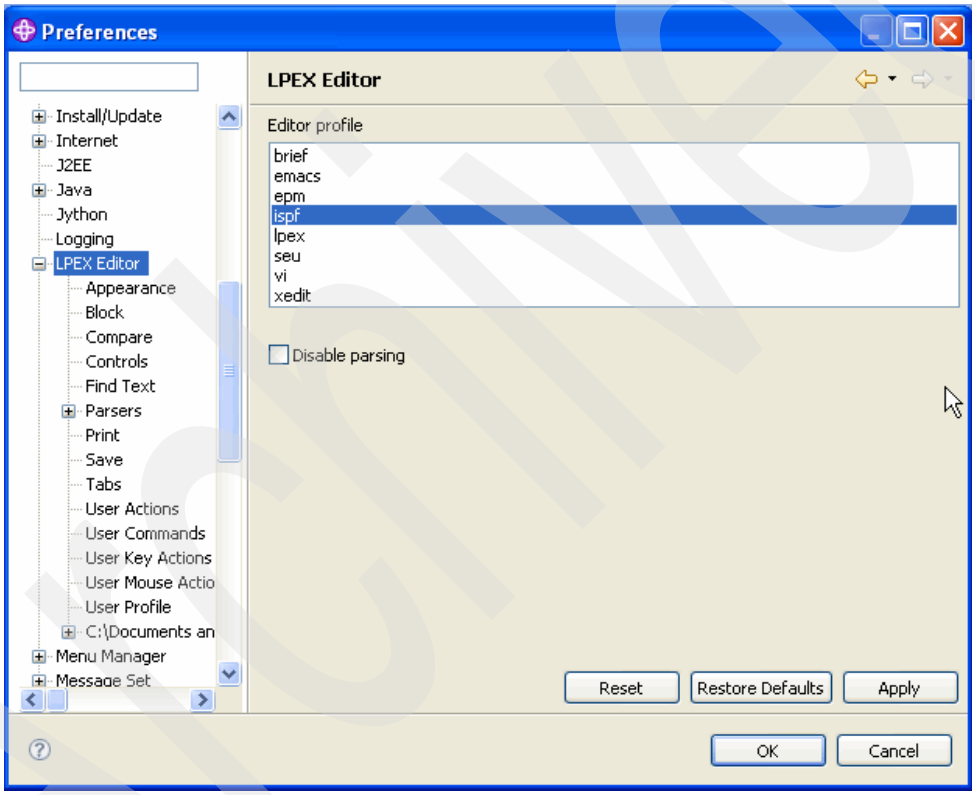


Figure 10-7 LPEX Editor preferences

Archive

2. Select **User Key Actions** to set PF Keys with a specific action. For example, you can associate F8 with the action pageDown, as shown in Figure 10-8.

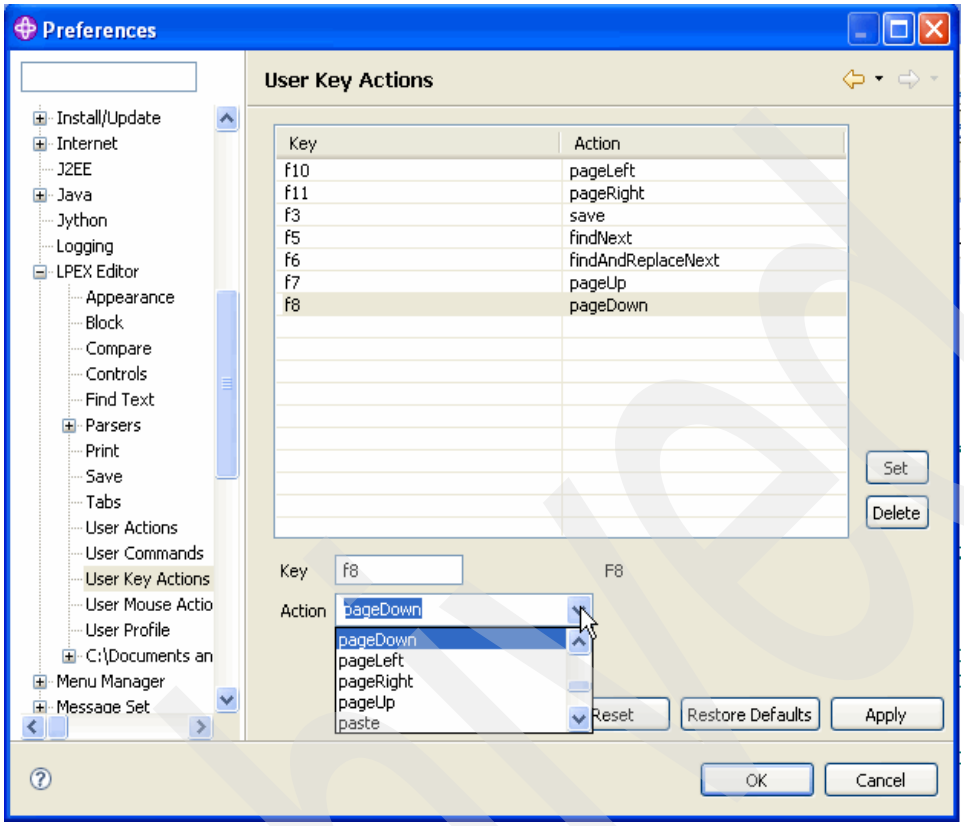


Figure 10-8 User Key Actions

Archive

10.2.2 Editing using LPEX

Go to a z/OS Project or to your data sets to open a source file. The LPEX Editor view shows the source with sequence numbers on left side and a command line on bottom side.

Prefix commands

You are able (like ISPF) to enter prefix commands. For example, type R5 in the prefix area of a source line and press Enter. The line will be repeated 5 times, as shown in Figure 10-9 and Figure 10-10 on page 255.

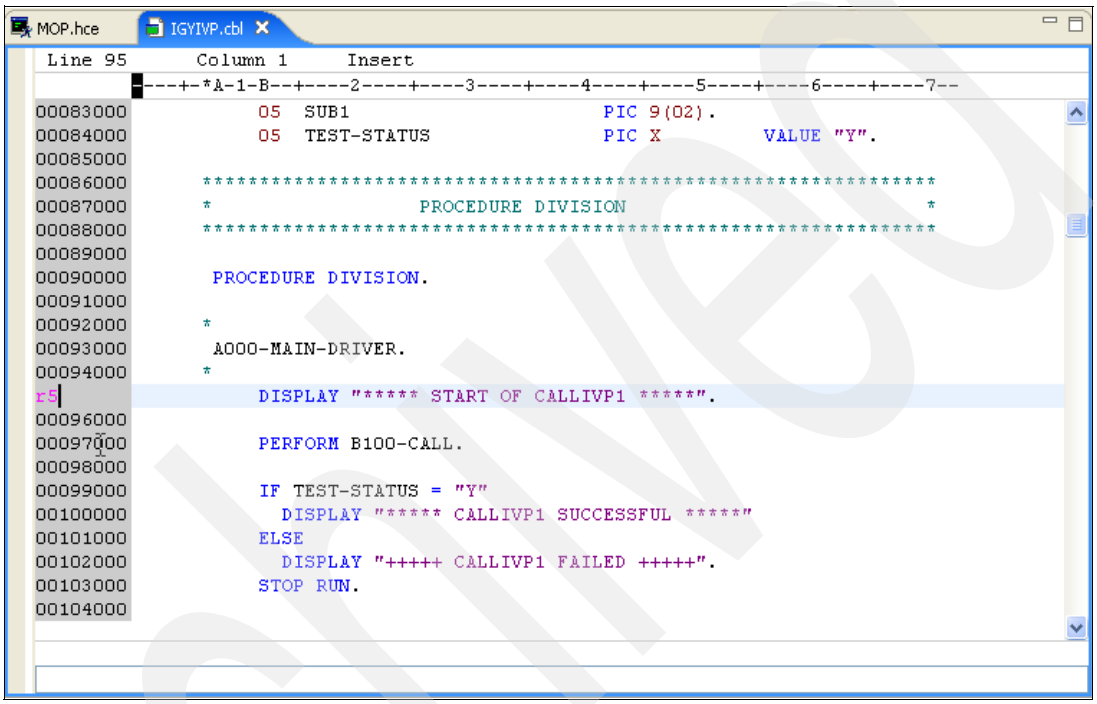


Figure 10-9 Prefix commands

Archived

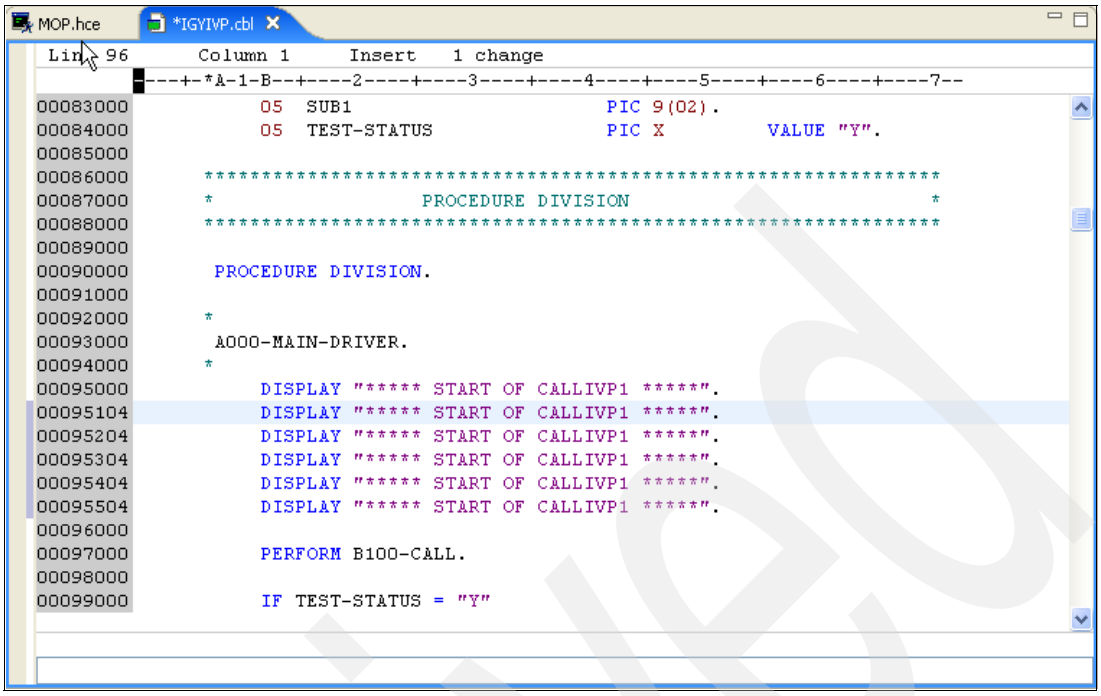


Figure 10-10 Repeat command executed

The available prefix commands are:

(Enter (in the prefix area to cause the editor to shift the specified line one character to the left.

(n Enter (n in the prefix area to cause the editor to shift the specified line n characters to the left.

((Enter ((in the prefix area of two different lines in the document to delimit a block of text that you want to shift one character to the left. Only the visible lines will be shifted.

((n Enter ((n in the prefix area of two different lines in the document to delimit a block of text that you want to shift n characters to the left. Only the visible lines will be shifted.

) Enter) in the prefix area to cause the editor to shift the specified line one character to the right.

)n Enter)n in the prefix area to cause the editor to shift the specified line n characters to the right.

)) Enter)) in the prefix area of two different lines in the document to delimit a block of text that you want to shift one character to the right. Only the visible lines will be shifted.

))n Enter))n in the prefix area of two different lines in the document to delimit a block of text that you want to shift n characters to the right. Only the visible lines will be shifted.

/ Enter / in the prefix area to make the specified line the current line.

< Enter < in the prefix area to cause the editor to shift the specified line one character to the left. If there is not at least one blank at the start of the line, nothing is done.

Archived

<n	Enter <n in the prefix area to cause the editor to shift the specified line n characters to the left. If there are less than n spaces at the start of the line, only the spaces are deleted.
<<	Enter << in the prefix area of two different lines in the document to delimit a block of text that you want to shift one character to the left. Only the visible lines will be shifted. Lines that do not have at least one space at the start of the line will be unaffected.
<<n	Enter <<n in the prefix area of two different lines in the document to delimit a block of text that you want to shift n characters to the left. Only the visible lines will be shifted. Lines that have less than n spaces at the start of the line will only have those spaces deleted.
>	Enter > in the prefix area to cause the editor to shift the specified line one character to the right. If the line cannot be shifted without truncating a non blank character, nothing is done.
>n	Enter >n in the prefix area to cause the editor to shift the specified line n characters to the right. If the line cannot be shifted n characters without truncating a non blank character, the line will only be shifted as far to the right as it can be without truncating a non blank character.
>>	Enter >> in the prefix area of two different lines in the document to delimit a block of text that you want to shift one character to the right. Only the visible lines will be shifted. Lines that cannot be shifted without truncating a non blank character will be unaffected.
>>n	Enter >>n in the prefix area of two different lines in the document to delimit a block of text that you want to shift n characters to the right. Only the visible lines will be shifted. Lines that cannot be shifted without truncating a non blank character will only be shifted as far to the right as they can be without truncating a non blank character.
A	Enter A in the prefix area to cause the editor to copy or move lines after the specified line. The source lines and the type of operation (move or copy) are determined by the first occurrence of one of the following commands: C, Cn, CC, M, Mn, or MM.
An	Enter An in the prefix area to cause the editor to copy or move lines after the specified line. The source lines and the type of operation (move or copy) are determined by the first occurrence of one of the following commands: C, Cn, CC, M, Mn, or MM. n copies of the source will be inserted after the specified line. n is a positive integer.
B	Enter B in the prefix area to cause the editor to copy or move lines before the specified line. The source lines and type of operation (move or copy) are determined by the first occurrence of one of the following commands: C, Cn, CC, M, Mn, or MM.
Bn	Enter Bn in the prefix area to cause the editor to copy or move lines before the specified line. The source lines and type of operation (move or copy) are determined by the first occurrence of one of the following commands: C, Cn, CC, M, Mn, or MM. n copies of the source will be inserted before the specified line. n is a positive integer.
C	Enter C in the prefix area to indicate that the specified line is to be the source for a copy operation. The target location for the copy is determined by the first occurrence of one of the following commands: A, An, B, Bn, O, On, or OO.

Cn	Enter Cn in the prefix area to indicate that the specified line and the following n - 1 visible lines are to be the source for a copy operation. n is a positive integer. The target location for the copy is determined by the first occurrence of one of the following commands: A, An, B, Bn, O, On, or OO.
CC	Enter CC in the prefix area of two different lines in the document. The CC commands will delimit the source for a copy operation. Only visible lines are included in the source. The target location for the copy is determined by the first occurrence of one of the following commands: A, An, B, Bn, O, On, or OO.
D	Enter D in the prefix area to cause the editor to delete the specified line.
Dn	Enter Dn in the prefix area to cause the editor to delete the specified line and the following n - 1 visible lines.
DD	Enter DD in the prefix area of two different lines in the document to delimit a block of text that you want deleted. Only the visible lines will be deleted.
F	Enter F in the prefix area of an excluded block header to show the first line in the excluded block. Lines can be excluded with the X, Xn, and XX commands.
Fn	Enter Fn in the prefix area of an excluded block header to show the first n lines in the excluded block. Lines can be excluded with the X, Xn, and XX commands.
I	Enter I in the prefix area to cause the editor to insert a new line after the specified line.
In	Enter In in the prefix area to cause the editor to insert n new lines after the specified line. n is a positive integer.
L	Enter L in the prefix area of an excluded block header to show the last line in the excluded block. Lines can be excluded with the X, Xn, and XX commands.
Ln	Enter Ln in the prefix area of an excluded block header to show the last n lines in the excluded block. Lines can be excluded with the X, Xn, and XX commands.
LC	Enter LC in the prefix area to cause the editor to change the specified line to lower case.
LCn	Enter LCn in the prefix area to cause the editor to change the specified line and the following n - 1 visible lines to lower case.
LCC	Enter LCC in the prefix area of two different lines in the document to delimit a block of text that you want to change to lower case. Only the visible lines will be changed.
M	Enter M in the prefix area to indicate that the specified line is to be the source for a move operation. The target location for the move is determined by the first occurrence of one of the following commands: A, An, B, Bn, O, On, or OO.
Mn	Enter Mn in the prefix area to indicate that the specified line and the following n - 1 visible lines are to be the source for a move operation. n is a positive integer. The target location for the move is determined by the first occurrence of one of the following commands: A, An, B, Bn, O, On, or OO.

Archived

MM	Enter M in the prefix area to indicate that the specified line is to be the source for a move operation. The target location for the move is determined by the first occurrence of one of the following commands: A, An, B, Bn, O, On, or OO.
O	Enter O in the prefix area to cause the editor to overlay the specified line. The source lines and the type of operation (move or copy) are determined by the first occurrence of one of the following commands: C, Cn, CC, M, Mn, or MM.
On	Enter On in the prefix area to cause the editor to overlay the specified line and the following n - 1 visible lines. The source lines and the type of operation (move or copy) are determined by the first occurrence of one of the following commands: C, Cn, CC, M, Mn, or MM.
OO	Enter OO in the prefix area of two different lines in the document. The OO commands will delimit the overlay target for a move or copy operation. Only the visible lines will be overlaid. The source lines and the type of operation (move or copy) are determined by the first occurrence of one of the following commands: C, Cn, CC, M, Mn, or MM.
R	Enter R in the prefix area to cause the editor to duplicate the specified line.
Rn	Enter Rn in the prefix area to cause the editor to duplicate the specified line n times.
RR	Enter RR in the prefix area of two different lines in the document. The RR commands will delimit a block of text that you want duplicated. Only the visible lines will be duplicated.
RRn	Enter RRn in the prefix area of two different lines in the document. The RRn commands will delimit a block of text that you want duplicated n times. Only the visible lines will be duplicated.
UC	Enter UC in the prefix area to cause the editor to change the specified line to upper case.
UCn	Enter UCn in the prefix area to cause the editor to change the specified line and the following n - 1 visible lines to upper case.
UCC	Enter UCC in the prefix area of two different lines in the document to delimit a block of text that you want to change to upper case. Only the visible lines will be changed.
X	Enter X in the prefix area to cause the editor to exclude the specified line. The line can be reshown with the L, Ln, F, or Fn commands.
Xn	Enter Xn in the prefix area to cause the editor to exclude the specified line and the following n - 1 visible lines. The line can be reshown with the L, Ln, F, or Fn commands.
XX	Enter XX in the prefix area of two different lines in the document to delimit a block of text that you want to exclude. The line can be reshown with the L, Ln, F, or Fn commands.

Editor command line

You are able to issue commands at the editor command line. For example, you can enter the command UNDO to the action described in “Prefix commands” on page 254, as shown in Figure 10-11 on page 259 and Figure 10-12 on page 259.

Archive

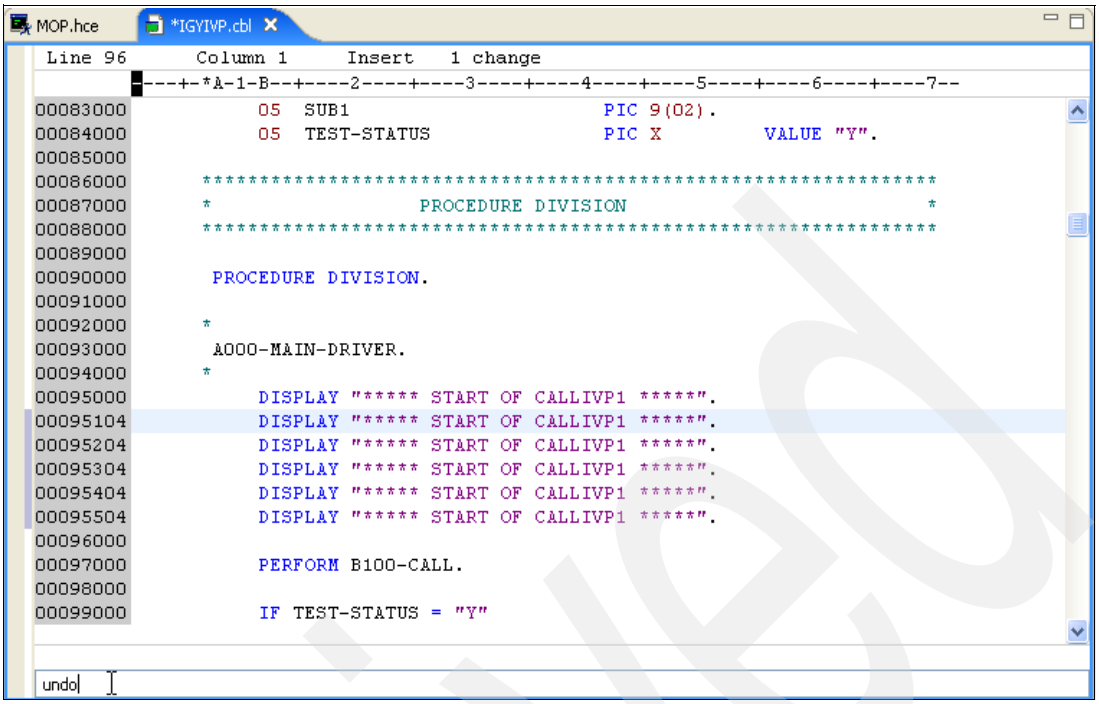


Figure 10-11 Command line

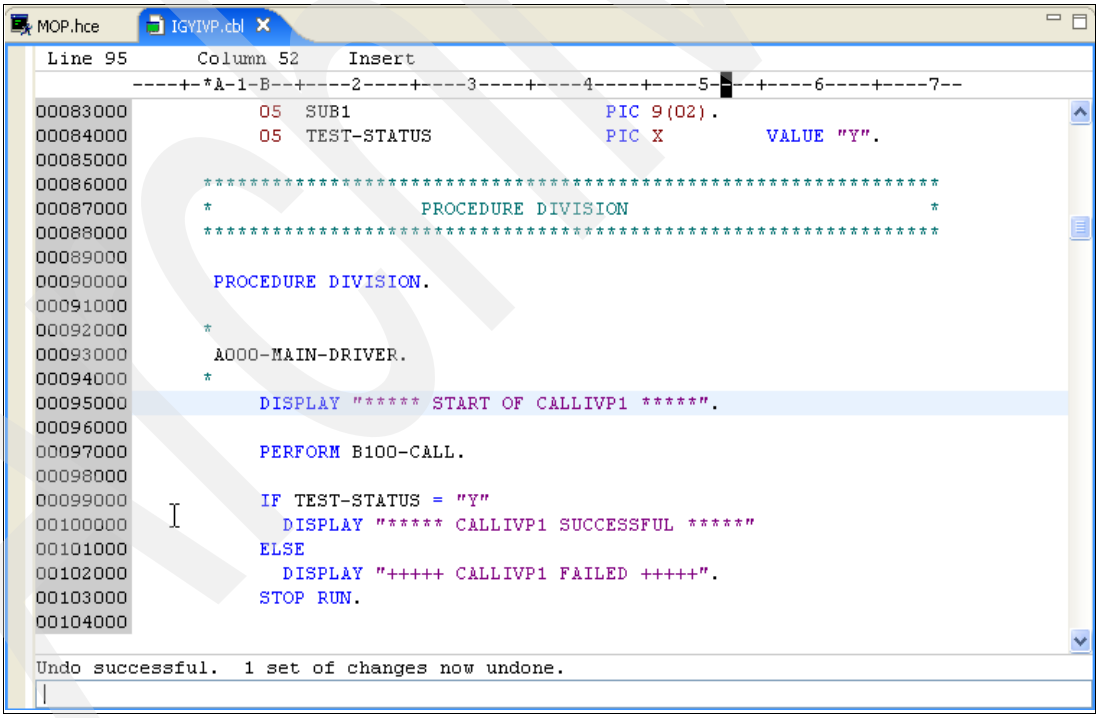


Figure 10-12 Command executed

Archived

The available commands are:

action	Runs an editor action.
add	Adds one or more blank lines.
block	Sets or manipulates a block of text.
calc	Performs simple calculations.
compare	Examines two documents for differences in their content.
delete	Deletes one or more lines.
deleteText	Deletes text at the current cursor position.
expandAll	Brings hidden lines back into view.
findText	Finds and optionally replaces text in the document.
get	Imports a file into the document.
help	Displays help information about a requested editor topic.
input	Gets user input to use with an editor command.
insert	Inserts a new line.
insertShow	Inserts a new show element into the document view.
insertText	Inserts text at the current cursor position.
load	Loads or reloads a file.
locate	Moves the cursor to a specified element, line, mark, sequence number, or sequence text.
parse	Triggers the document parser.
print	Prints the document.
processPrefix	Processes prefix commands.
profile	Exports or imports the editor default settings.
query	Returns the setting of an editor parameter.
replaceText	Replaces text at the current cursor position.
resequence	Resequences the document sequence numbers.
reset	Clears the editor default settings or error log.
save	Saves the contents of the document to a file.
saveAsHtml	Saves the contents of the document to an HTML file.
screenShow	Forces document views to be refreshed.
set	Assigns a value to an editor parameter.
sort	Sorts text lines in the document.
trim	Trims the trailing blanks in the document.
undo	Used to undo or redo one or more sets of document changes.
updateProfile	Updates selected editor parameters as they apply to a document view.

Archived

New command names linked to actions can be added as you like. For example, new commands F and FIND to do the same action as **`findText`**. These new commands can be added in the LPEX Editor preferences.Select **Window** → **Preferences** → **User Commands** to add new commands with a specific action. For example, Select the Class name **com.ibm.lpex.samples.FindTextContextCommand** and enter Name F. Click **Set**, as shown in Figure 10-13 and Figure 10-14.

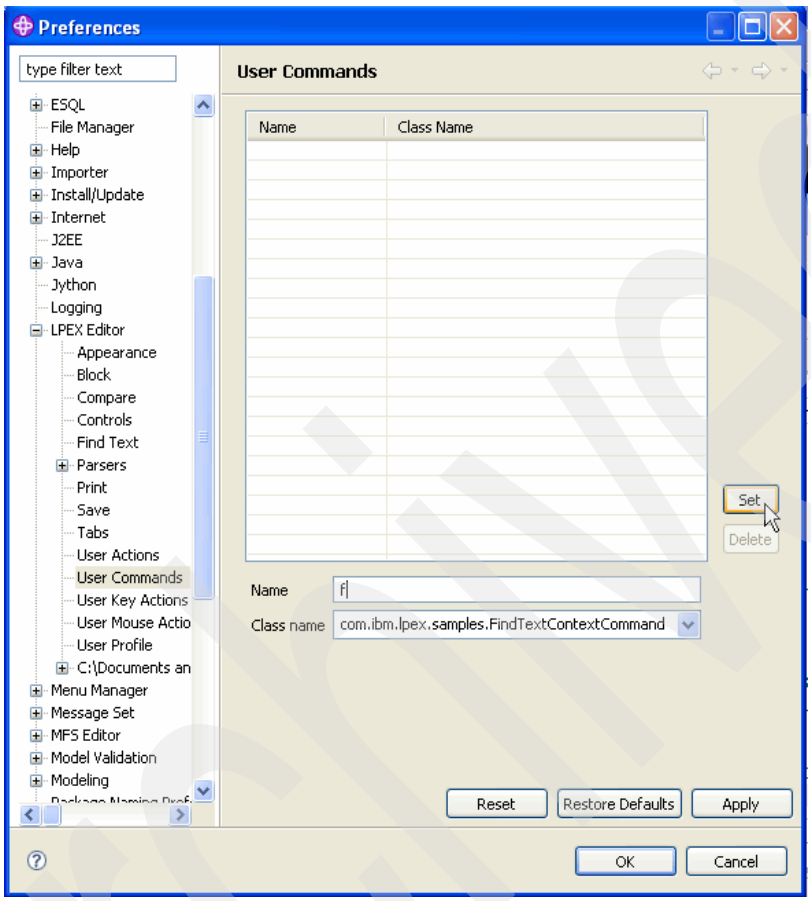


Figure 10-13 User Commands

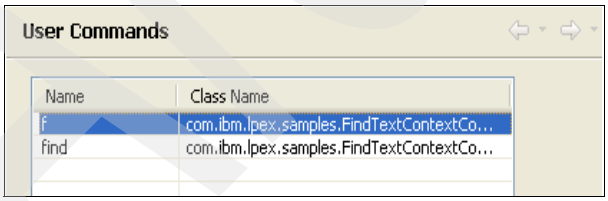


Figure 10-14 User Commands added

You can use this new command at the LPEX Editor command line. To find all occurrences of the text sub1, enter the command **`f all sub1`**. For more information about the **`findText`** command, go to the Help menu, select **Search**, and type in the command, as shown in Figure 10-15 and Figure 10-16.

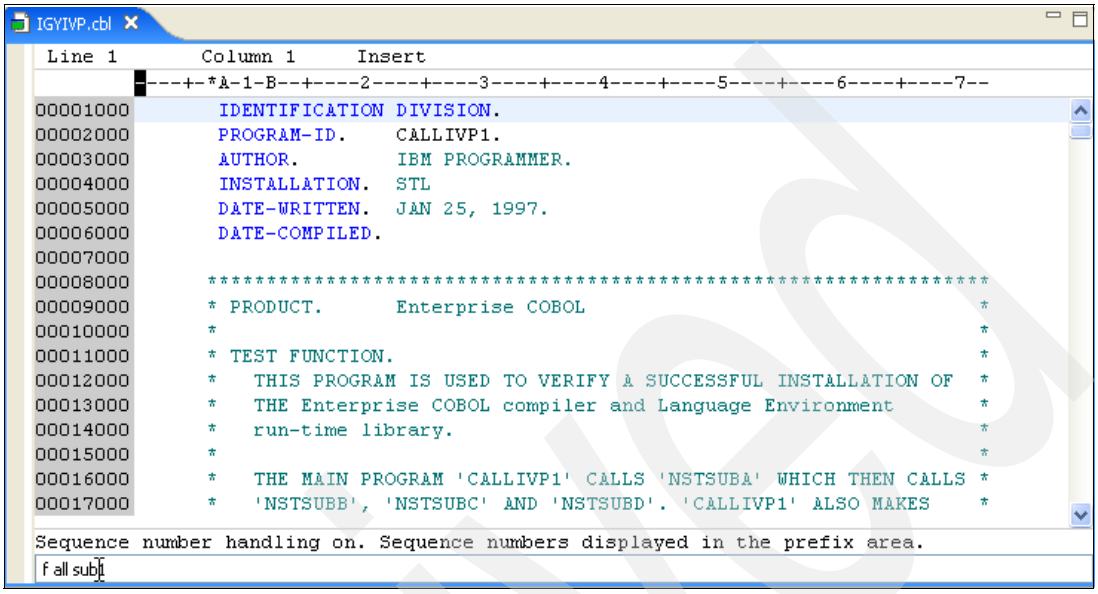


Figure 10-15 Find All command

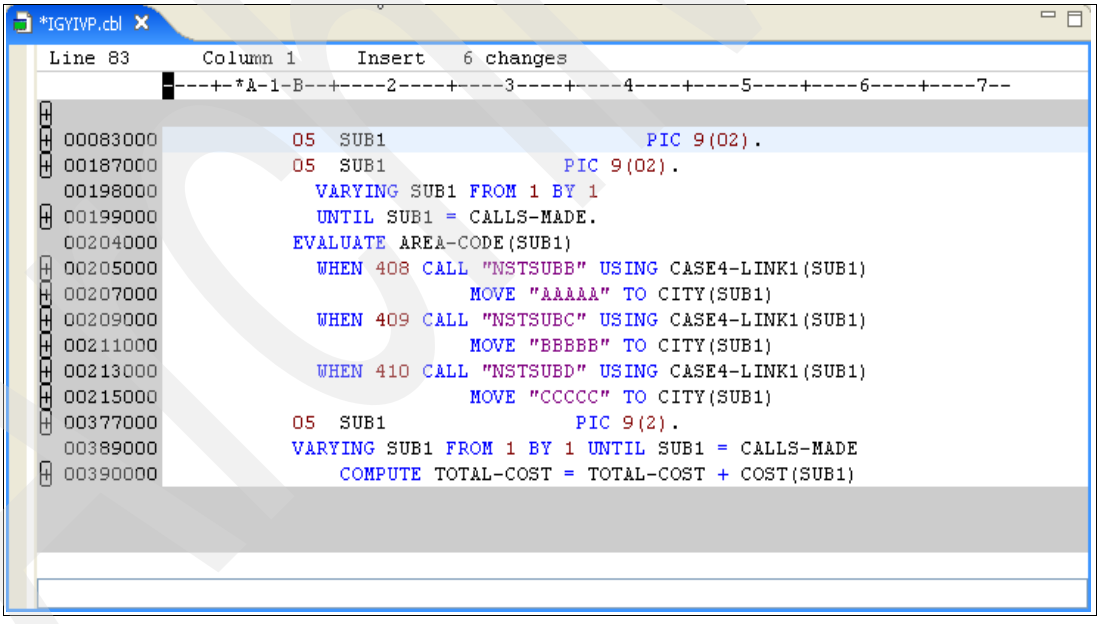


Figure 10-16 Find All Result

Using keys and mouse events

Keys and Mouse Events can be used with the LPEX Editor. This mode of editing does not exist in ISPF except for keys usable with the Terminal Emulator. For example, Ctrl-F starts the Find/Replace wizard. Click **All** to find all occurrences of sub1. To show all again, type Ctrl-W (keys) or right-click and select **Show all** (mouse event), as shown in Figure 10-17 through Figure 10-19 on page 264.

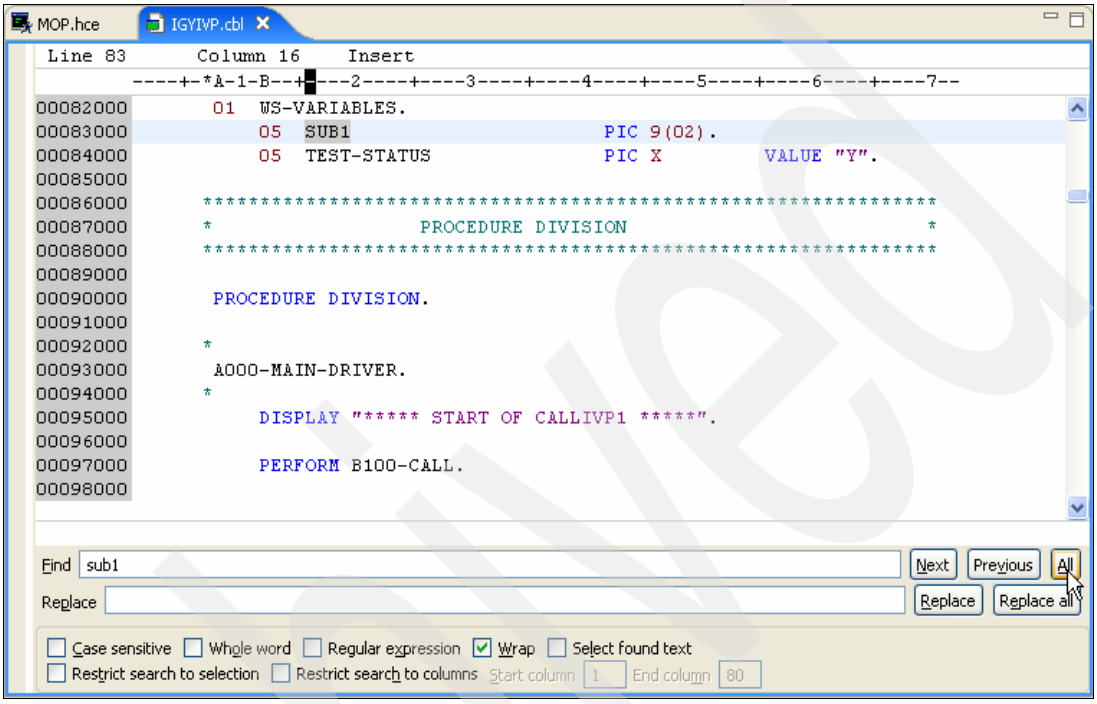


Figure 10-17 Find/Replace Wizard

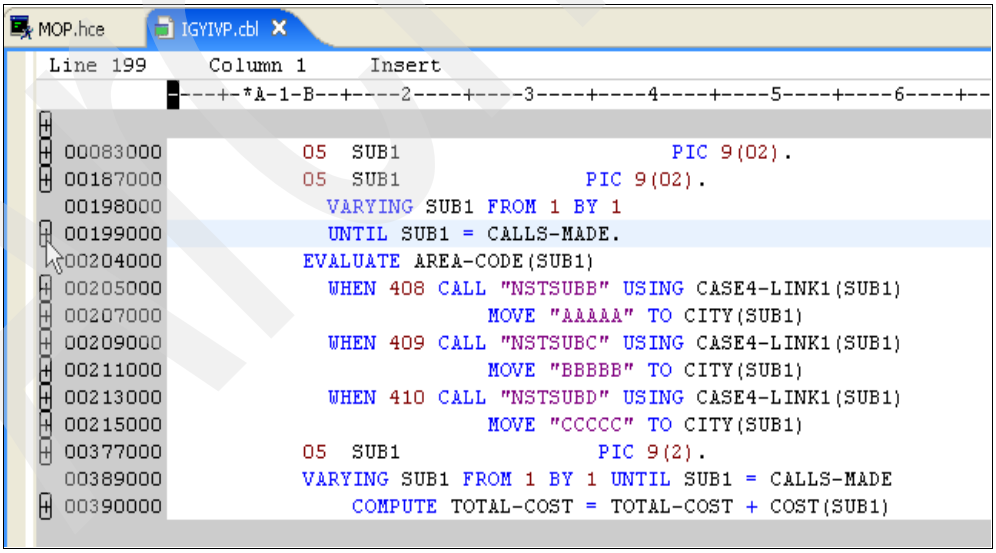


Figure 10-18 Find All Result

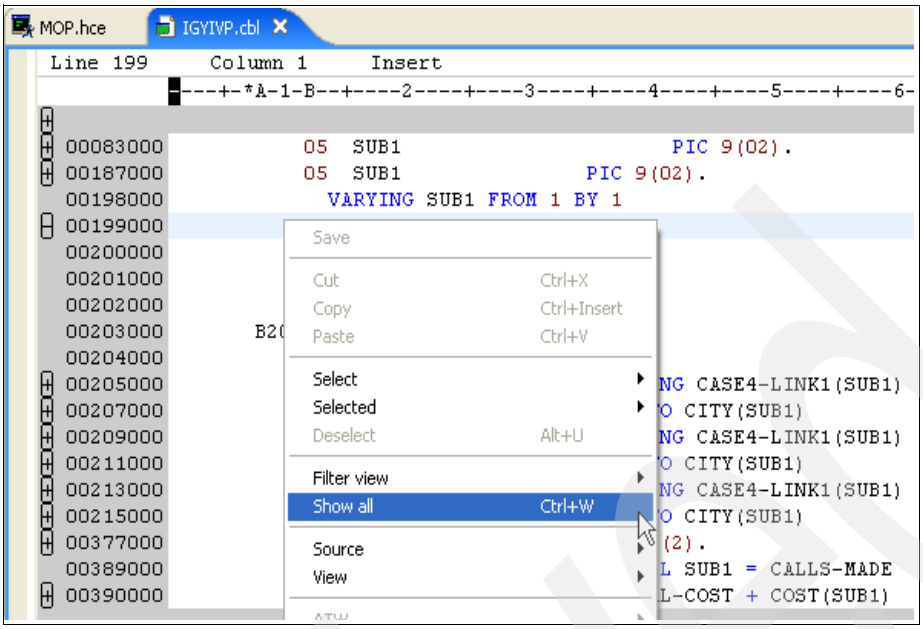


Figure 10-19 Keys and Mouse Events

The keys are (an explanation of the actions can be found by selecting **Help** → **Search**):

Alt+B	blockMarkCharacter
Alt+Backspace	undo
Alt+C	blockCopy
Alt+D	blockDelete
Alt+Enter	newLine
Alt+F7	blockShiftLeft
Alt+F8	blockShiftRight
Alt+I	blockLowerCase
Alt+J	join
Alt+K	blockUpperCase
Alt+L	blockMarkElement
Alt+M	blockMove
Alt+Numpad Enter	newLine
Alt+Page Down	findBlockEnd
Alt+Page Up	findBlockStart
Alt+Q	findQuickMark
Alt+R	blockMarkRectangle
Alt+S	split
Alt+U	blockUnmark
Alt+Z	blockOverlay
Backspace	backSpace
Ctrl+A	blockMarkAll

Archived

Ctrl+Backspace	deleteLine
Ctrl+C	copy
Ctrl+D	duplicateLine
Ctrl+Delete	truncate
Ctrl+Down	scrollDown
Ctrl+End	bottom
Ctrl+Enter	openLine
Ctrl+F	find
Ctrl+Home	top
Ctrl+Insert	copy
Ctrl+J	findLastChange
Ctrl+L	locateLine
Ctrl+Left	prevWord
Ctrl+M	match
Ctrl+N	findNext
Ctrl+Numpad Enter	openLine
Ctrl+P	print
Ctrl+Page Down	pageRight
Ctrl+Page Up	pageLeft
Ctrl+Q	setQuickMark
Ctrl+Right	nextWord
Ctrl+S	save
Ctrl+Shift+End	blockMarkBottom
Ctrl+Shift+F	findAndReplace
Ctrl+Shift+Home	blockMarkTop
Ctrl+Shift+L	locateSequenceLine
Ctrl+Shift+Left	blockMarkPrevWord
Ctrl+Shift+M	findMatch
Ctrl+Shift+N	compareNext
Ctrl+Shift+P	comparePrevious
Ctrl+Shift+Page Down	blockMarkPageRight
Ctrl+Shift+Page Up	blockMarkPageLeft
Ctrl+Shift+R	compareRefresh
Ctrl+Shift+Right	blockMarkNextWord
Ctrl+Shift+U	comparePrevious
Ctrl+Shift+Z	redo
Ctrl+T	scrollTop
Ctrl+U	findUp

Archived

Ctrl+Up	scrollUp
Ctrl+V	paste
Ctrl+W	showAll
Ctrl+X	cut
Ctrl+Y	redo
Ctrl+Z	undo
Delete	delete
Down	down
End	end
Enter	splitLine
Esc	commandLine
F1	help
Home	home
Insert	toggleInsert
Left	left
Numpad Enter	splitLine
Page Down	pageDown
Page Up	pageUp
Right	right
Shift+Backspace	backSpace
Shift+Delete	cut
Shift+Down	blockMarkDown
Shift+End	blockMarkEnd
Shift+Enter	newLine
Shift+F10	popupAtCursor
Shift+Home	blockMarkHome
Shift+Insert	paste
Shift+Left	blockMarkLeft
Shift+Numpad Enter	newLine
Shift+Page Down	blockMarkPageDown
Shift+Page Up	blockMarkPageUp
Shift+Right	blockMarkRight
Shift+Tab	prevTabStop
Shift+Up	blockMarkUp
Tab	insertToTab
Up	up

Archived

Filter view

There are several filters to show only a specific category of lines of codes. Right-click and select **Filter view** → **Comments** to see only comments lines, as shown in Figure 10-20.

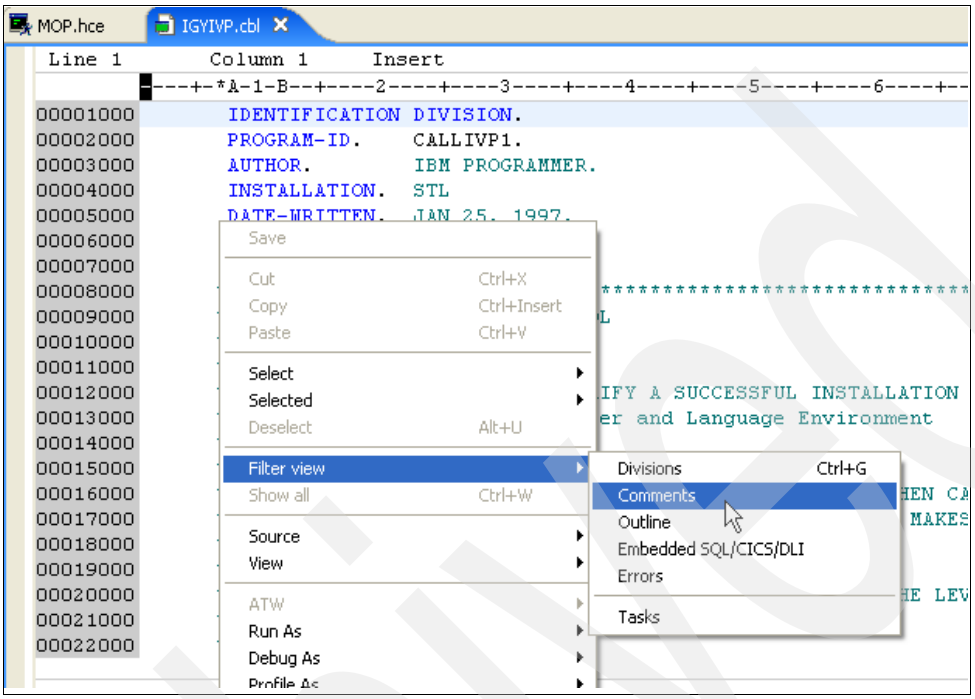


Figure 10-20 Filter view

Archived

Multiple views

It is possible to see several vertical or horizontal views of the same source, which permits you to focus on different parts of the source in the same view. Editing is permitted in all views. Right-click and select **View** → **Open new view** (or press Ctrl-2) to see an additional view of the source, as shown in Figure 10-21 through Figure 10-24 on page 269.

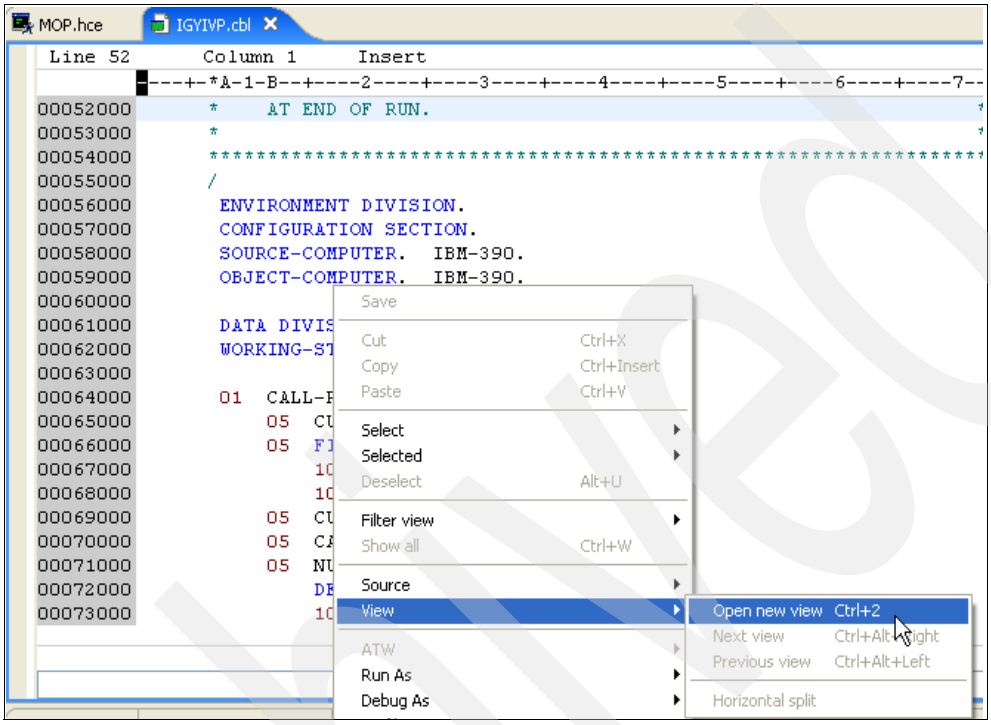


Figure 10-21 Multiple views

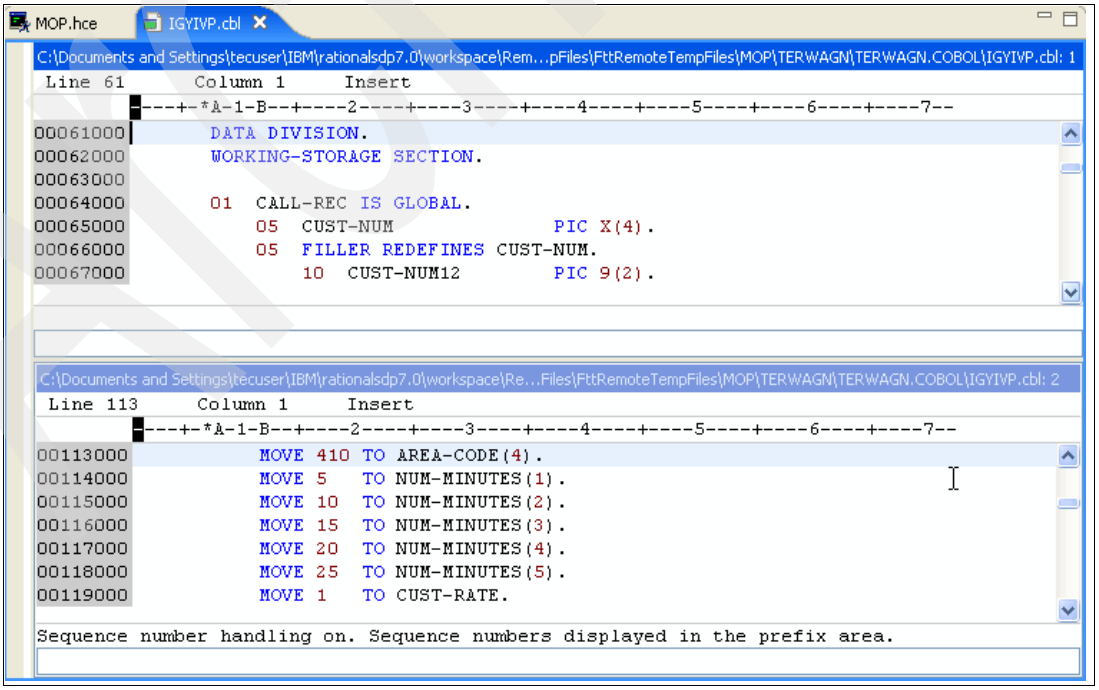


Figure 10-22 Split views

Hex edit line

Right-click a line of code and select **Source** → **Hex edit line** (see Figure 10-23 and Figure 10-24).

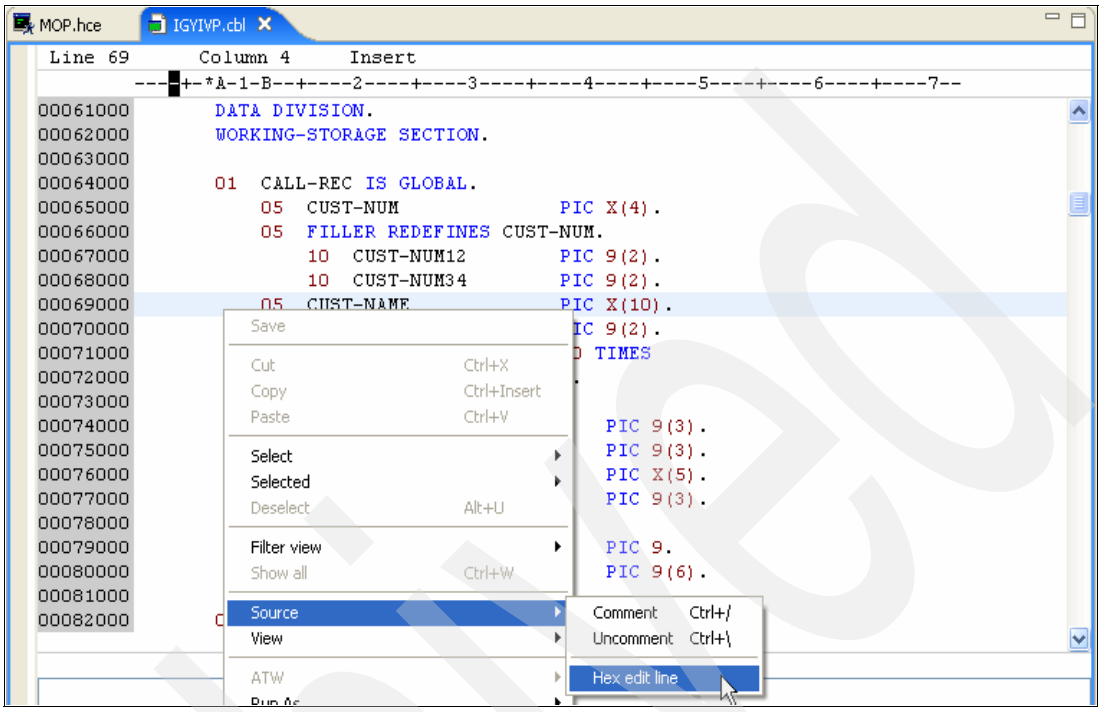


Figure 10-23 Hex edit line

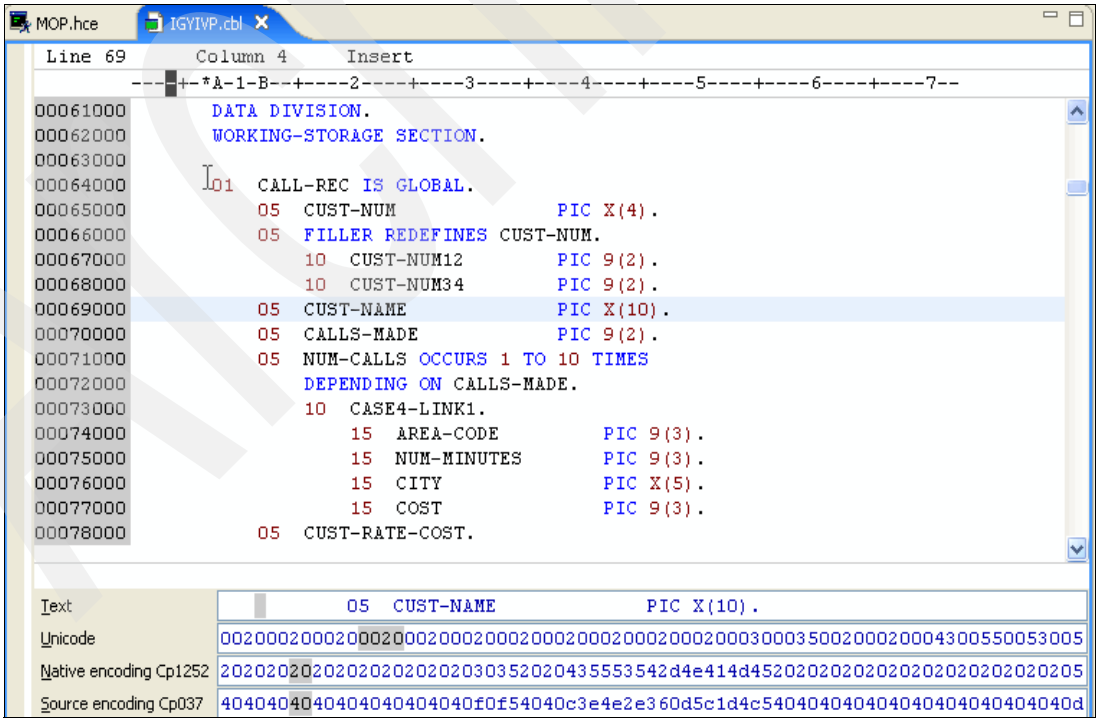


Figure 10-24 Hexadecimal view

Outline view

The workbench Outline view is customized to assist you as you edit JCL, COBOL, PL/I, and REXX files in the z/OS LPEX Editor, as shown in Figure 10-25.

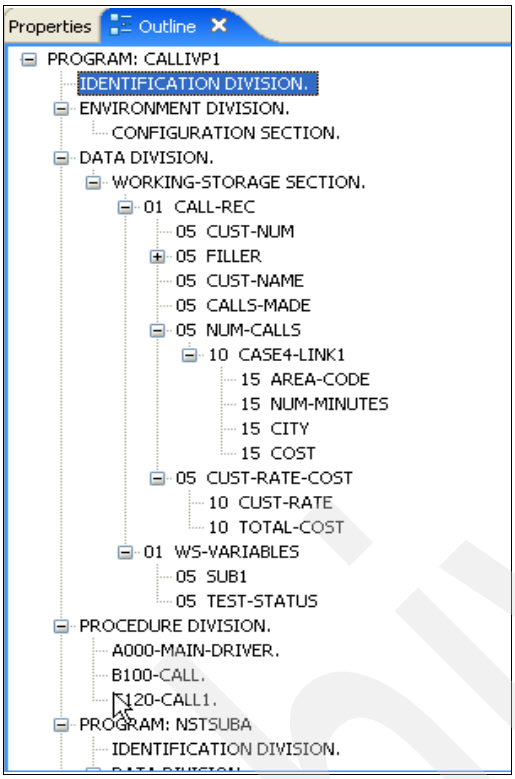


Figure 10-25 Outline view

10.2.3 Advanced functions

Here we discuss some advanced functions.

Replace with local history

A local edit history of a file is maintained when you create or modify a file. Each time you edit and save the file, a copy is saved so that you can replace the current file with a previous edit or even restore a deleted file. You can also compare the contents of all the local edits. Each edit in the local history is uniquely represented by the date and time the file was saved. Right-click a source file and select **Replace with** → **Local History**, as shown in Figure 10-26 on page 271 and Figure 10-27 on page 271.

Archived

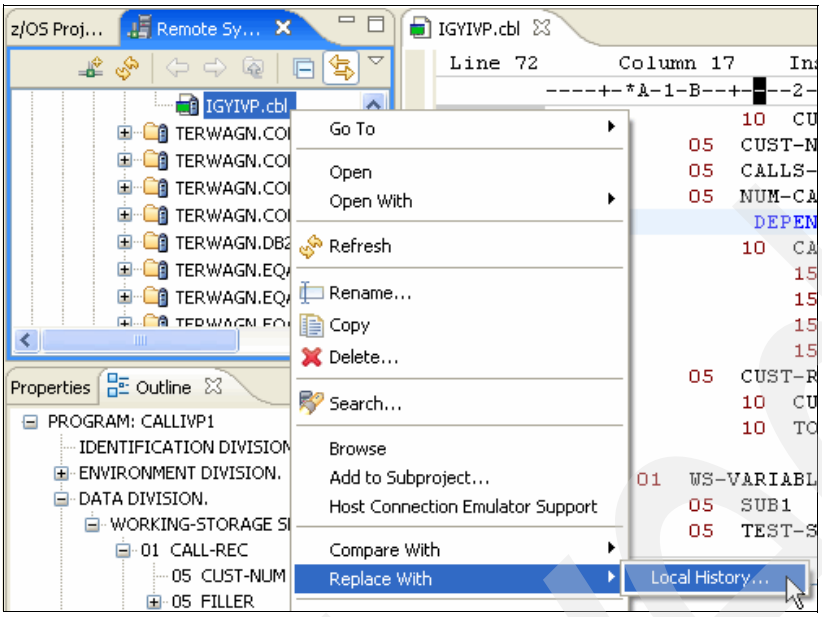


Figure 10-26 Replace With Local History

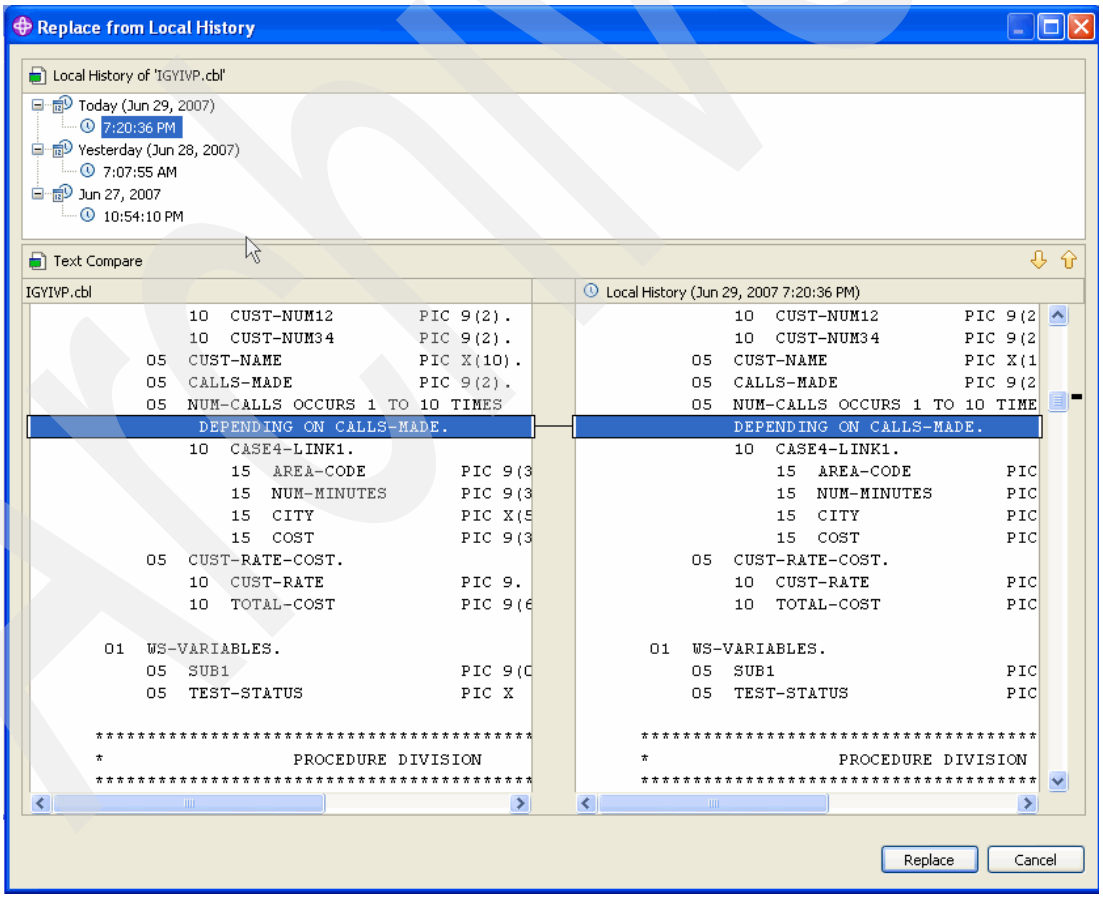


Figure 10-27 Local History

Compare files

In one of the navigation views, select two files and select **Compare With** → **Each Other**, as shown in Figure 10-28 and Figure 10-29.

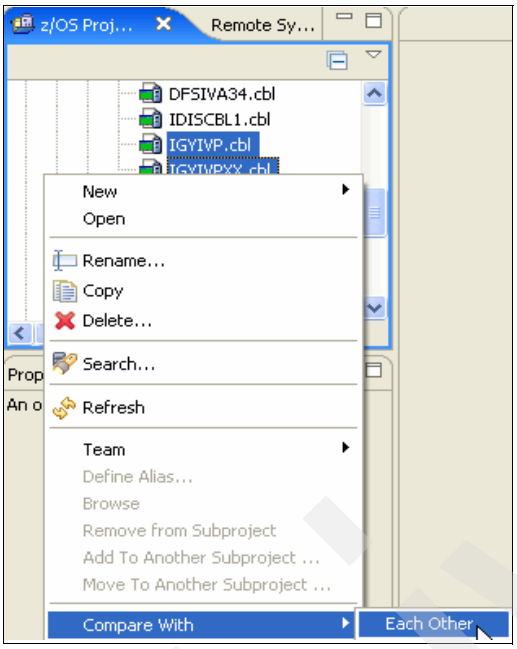


Figure 10-28 Compare two files

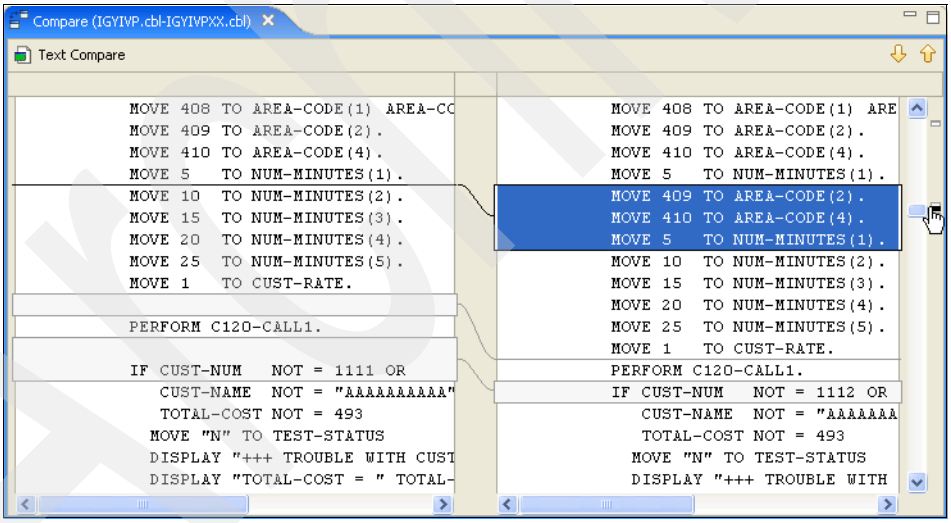


Figure 10-29 Compare view

Remote search

You can search through a z/OS system for files or files that contain a search string. Select **Search** → **Remote**, and then click the **Remote z/OS Search** tab. In the Search string field, enter CICS (for example). Select the search scope by clicking **Browse** and one or several data sets. Click **Search**. See the Search Result and open the source edit by double-clicking each result line, as shown in Figure 10-30 on page 273 through Figure 10-35 on page 276.

Archived

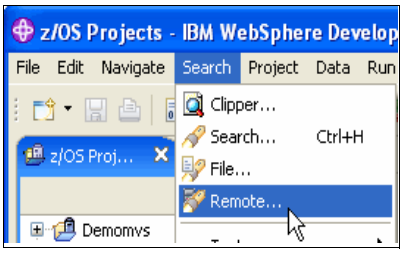


Figure 10-30 Go to remote search

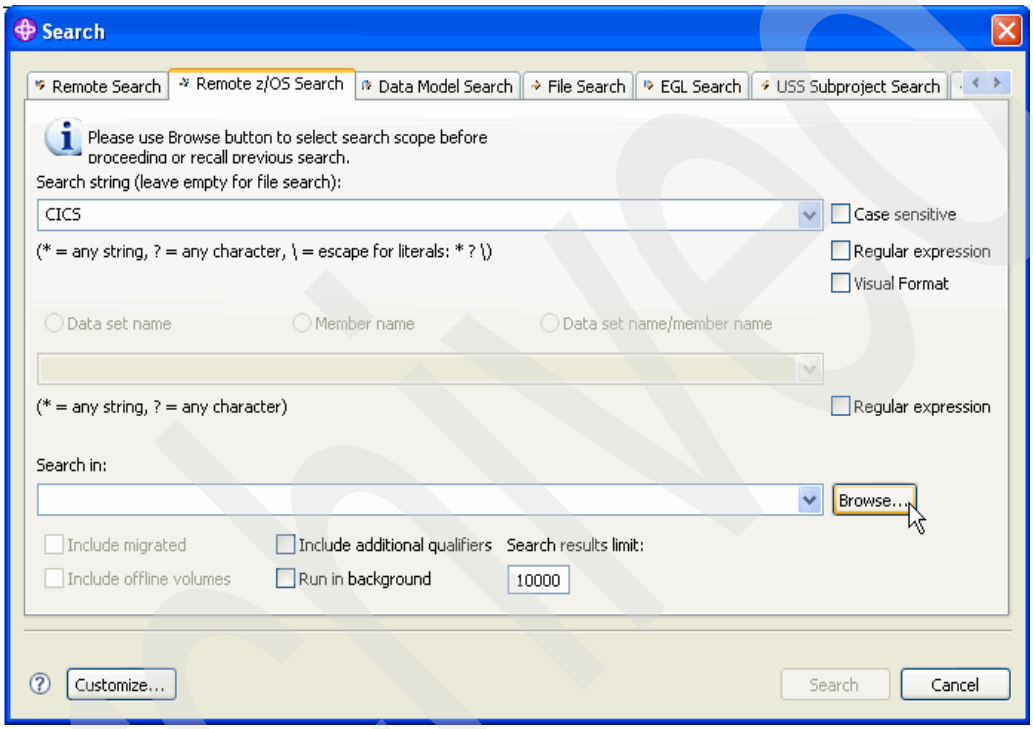


Figure 10-31 Remote z/OS Search

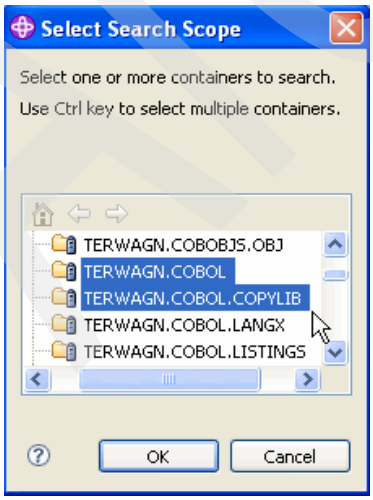


Figure 10-32 Select Search Scope

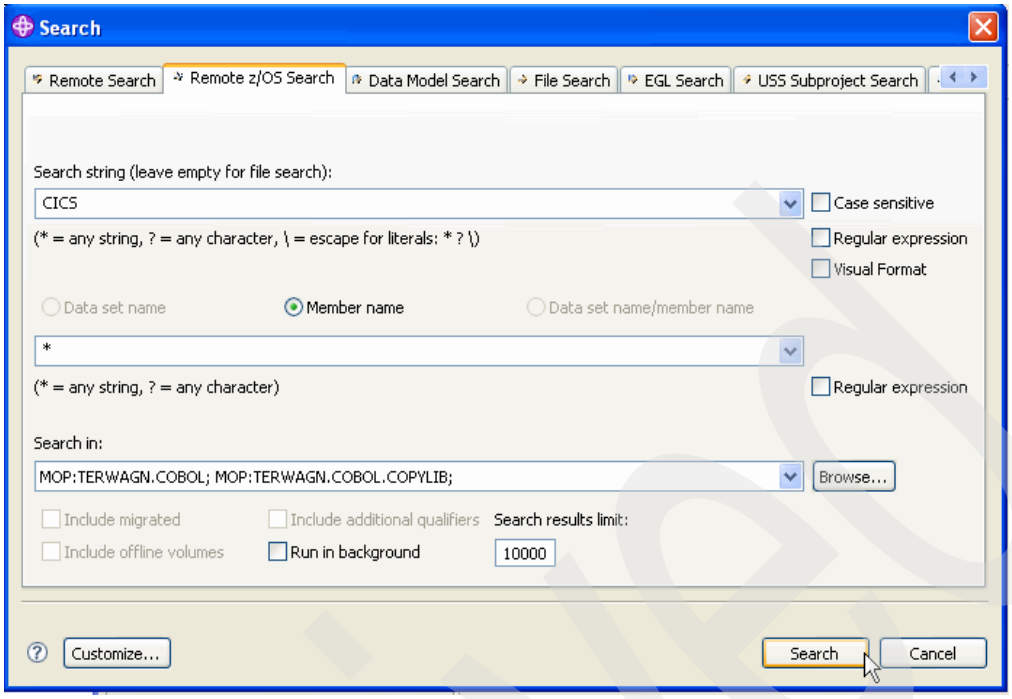


Figure 10-33 Starting Remote z/OS Search

Archive

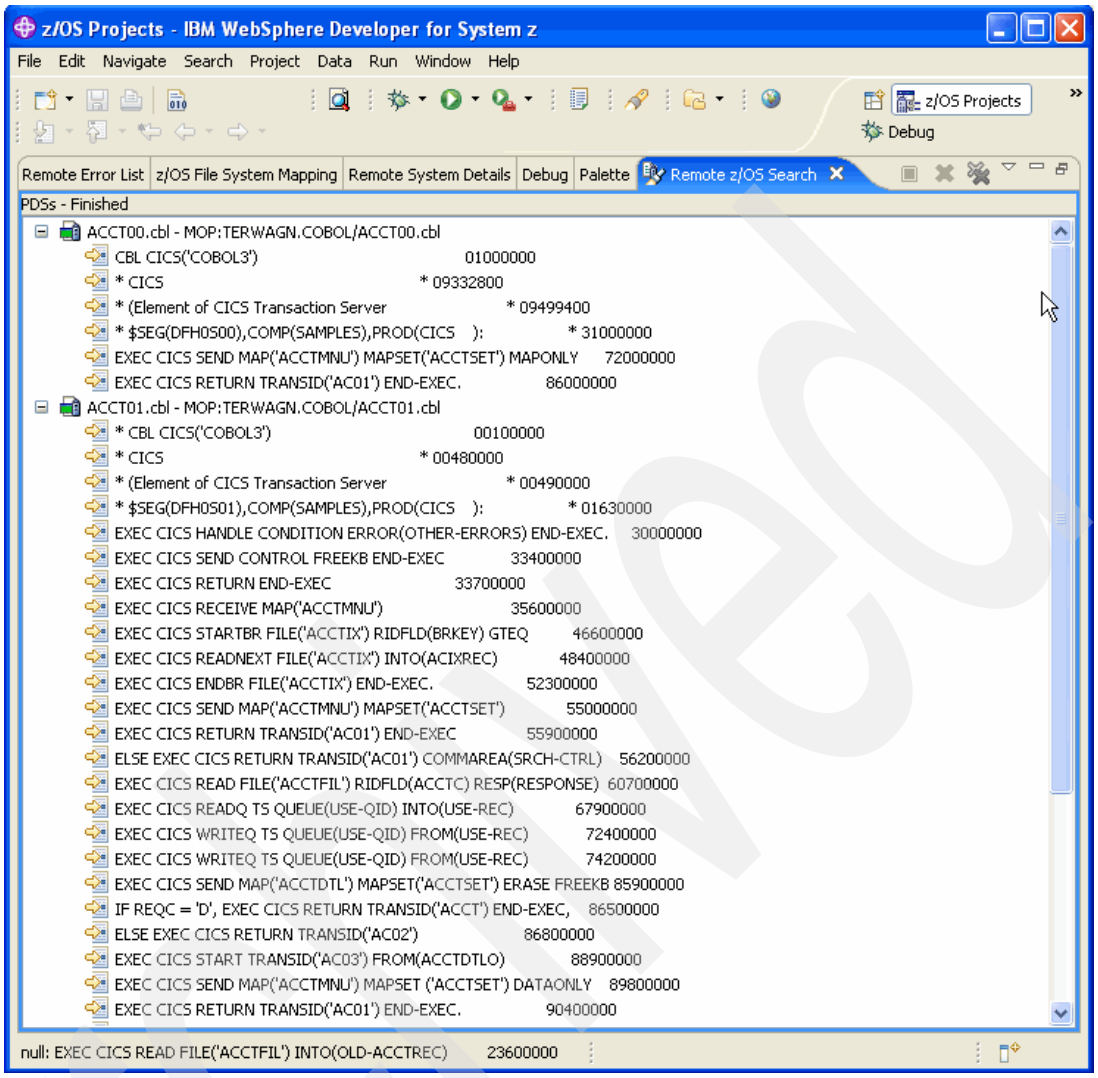


Figure 10-34 Remote z/OS Search result

Archived

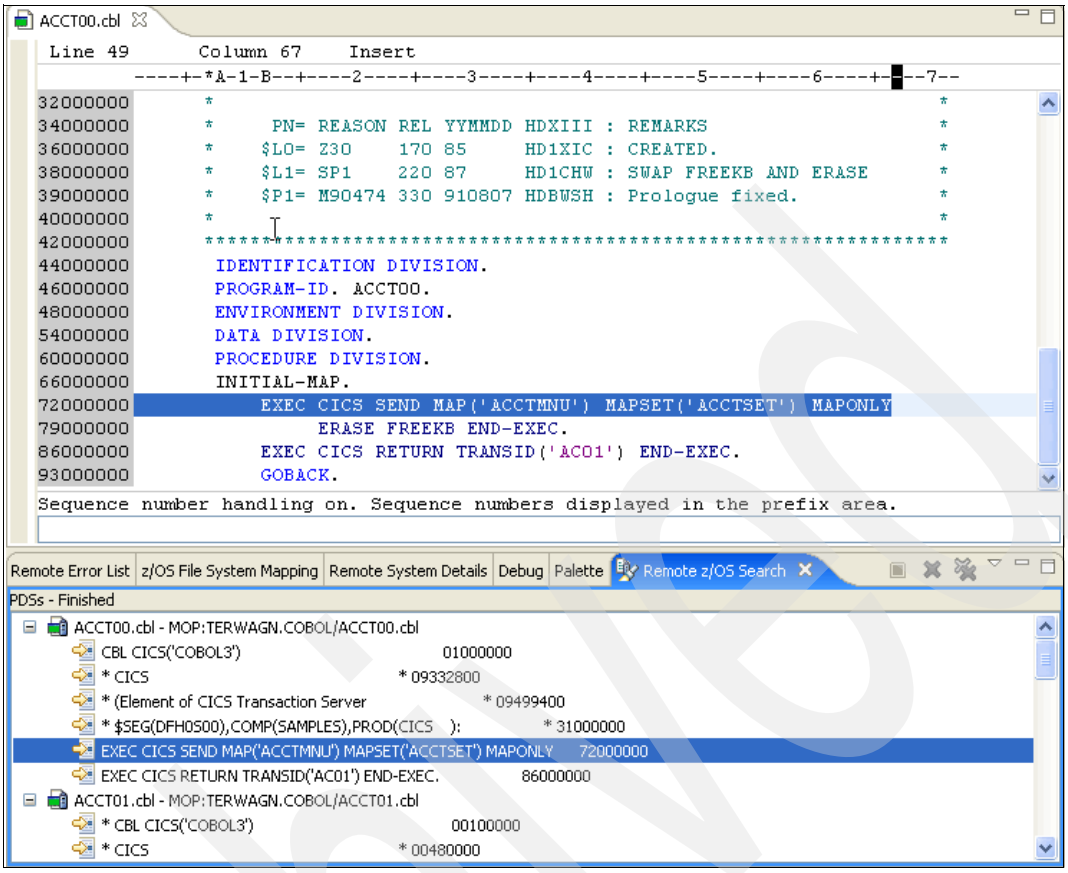


Figure 10-35 Edit source from Remote z/OS Search result

10.3 Developing and debugging a batch program

Note: The sample batch programs used in this chapter are delivered with the COBOL and PL/I compilers. These sample programs may already be installed on your system. If you need them, we provide them in the Web material associated with this book; refer to Appendix B, “Additional material” on page 645 for more information.

This section describes how to develop, compile, and debug two batch programs (one COBOL and one PL/I) using WD/z. Seeing that there are several methods to manage your development projects, we decided to choose one that permits you to quickly start by creating a new project and as many subprojects as programs.

Do these steps:

1. Using the z/OS Projects perspective, right-click in z/OS Projects, click **New**, and select **z/OS Project...**, as shown in Figure 10-36 on page 277.

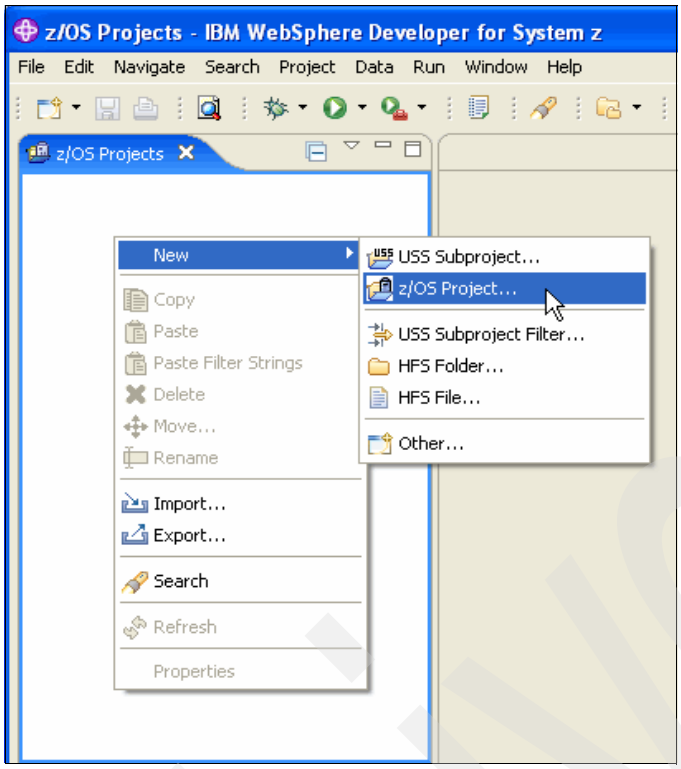


Figure 10-36 Batch programs: New z/OS Project

2. Type a Project Name of your choice and check **Create an MVS subproject**, as shown in Figure 10-37.

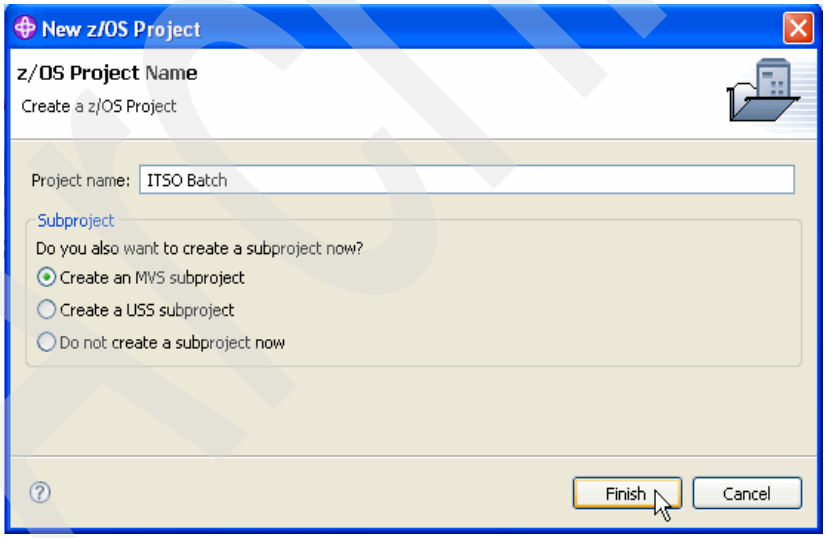


Figure 10-37 Batch programs: Create z/OS Project

Archived

3. Enter the name of the Batch Cobol subproject. Click **Finish**, as shown in Figure 10-38.

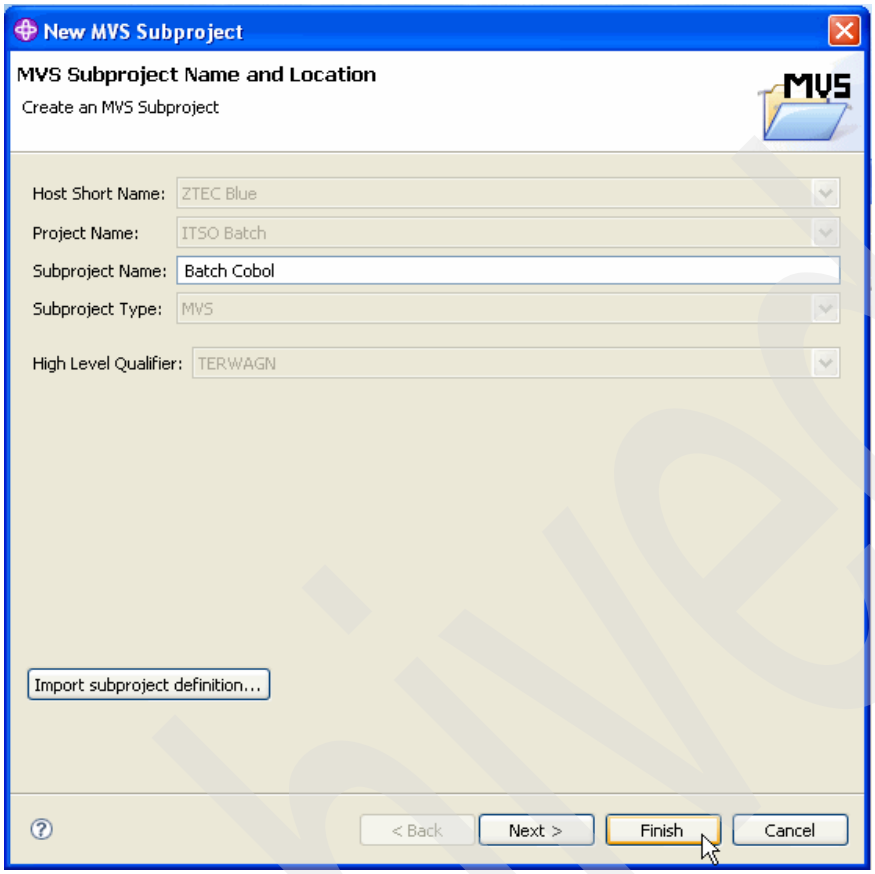


Figure 10-38 Batch programs: Create z/OS Subproject

4. Now we add the COBOL program to the Batch Cobol subproject.
   a. Open the Connection view at the MVS Files and PDS where the Batch Cobol subproject's source resides.
   b. Select the source (**IGYIVP**), right-click it, and click **Add selected resources to subproject**, as shown in Figure 10-39 on page 279.

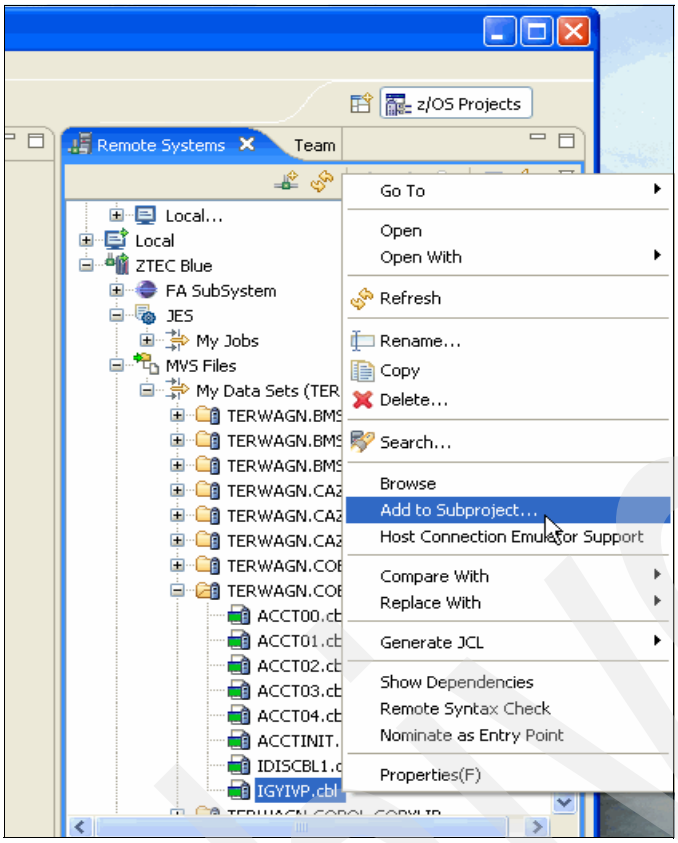


Figure 10-39 Adding of the Cobol Batch program to the subproject

c. Choose the right subproject (for example, Cobol Batch) and click **Finish**, as shown in Figure 10-40.

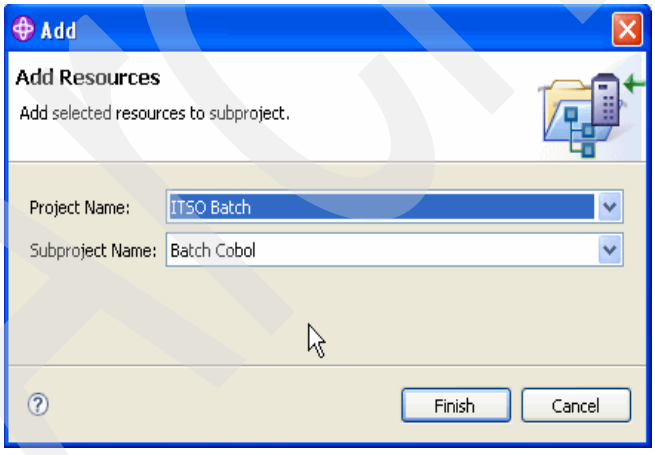


Figure 10-40 Add resource to subproject

Archived

5. Do the same for the PL/I program. In the project view, select the project, right-click it, and select **New** → **MVS subproject**. Open the Connection view to add the PL/I program to this new subproject, as shown in Figure 10-41 through Figure 10-44 on page 282.

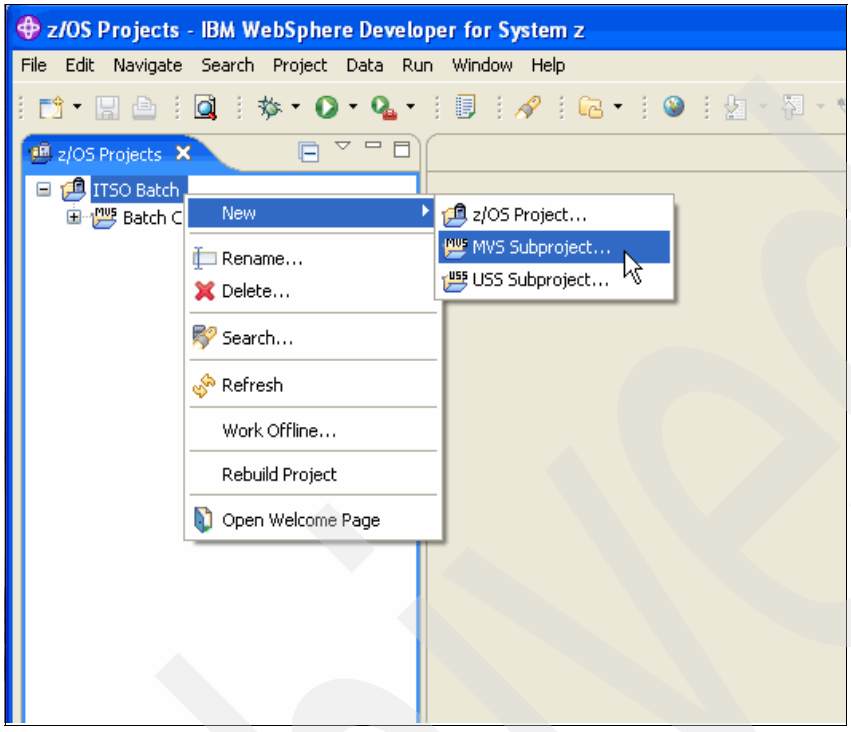


Figure 10-41 Adding a new batch PL/I subproject

Archived

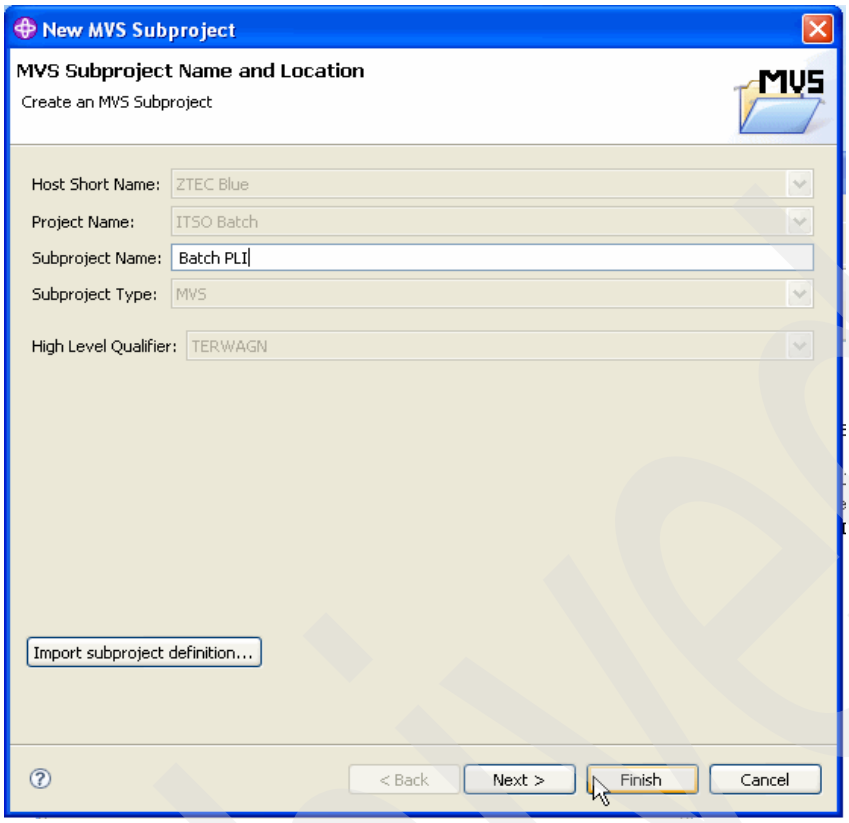


Figure 10-42 Create the PL/I subproject

Archived

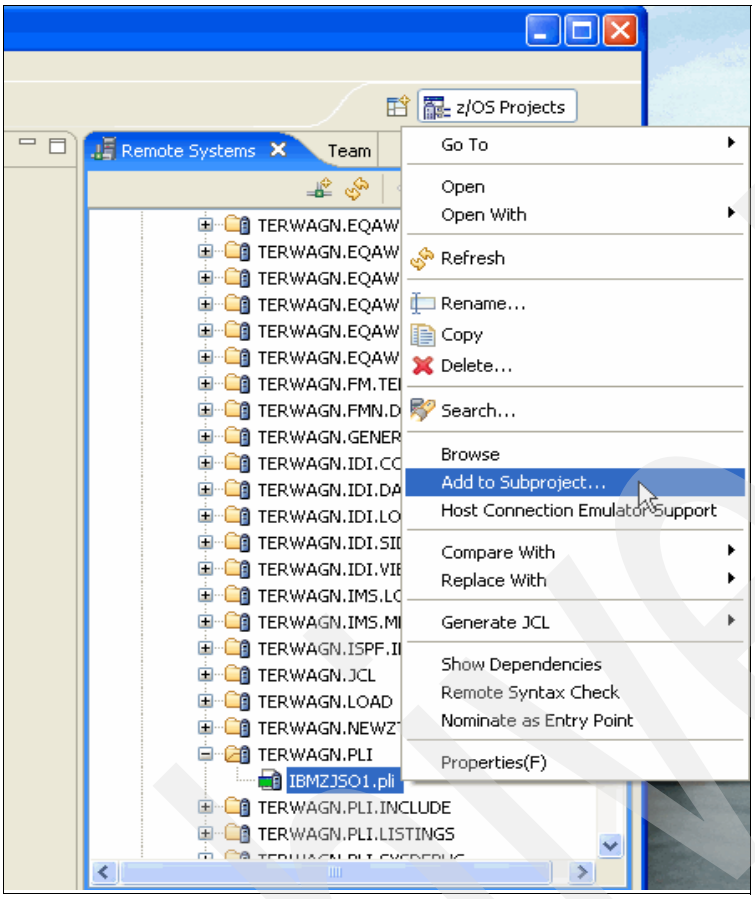


Figure 10-43 Adding of the PL/I batch program (IBMZJSO1) to the subproject

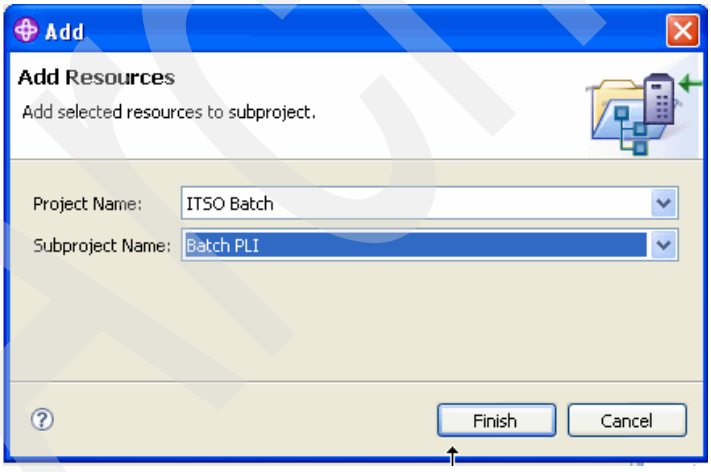


Figure 10-44 Adding a resource to the subproject

6. Now it is time to build the project to produce two load modules on the target system. It will generate the build JCLs and submit it to the mainframe.

   In the Project view, right-click the project and select **Rebuild Project**, as shown in Figure 10-45 on page 283.

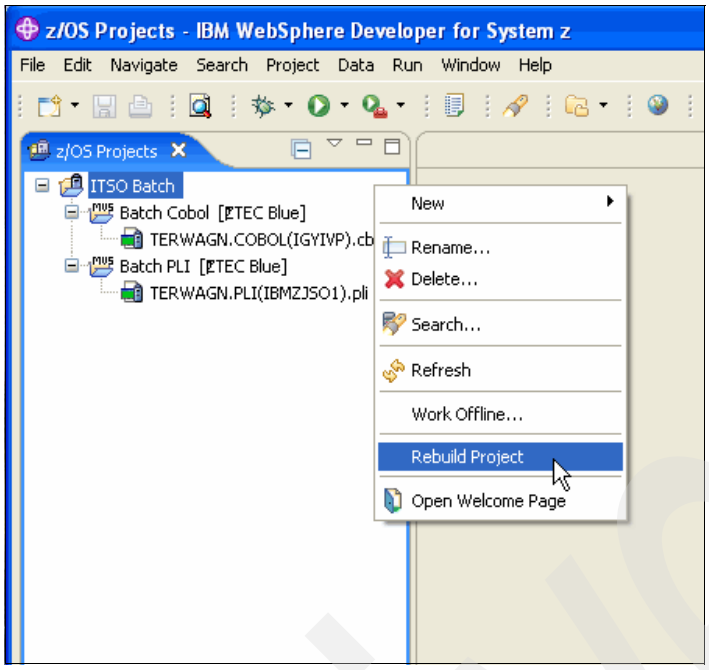


Figure 10-45 Rebuild the batch project

7. A Progress Information window appears, as shown in Figure 10-46.

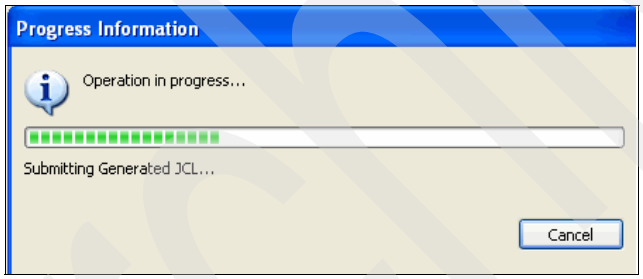


Figure 10-46 Rebuild project - Progress Information

Archived

8. There are two possible results:

   – No error: You can see, for each subproject, all the objects generated by the build procedure (listing, sysdebug, object, and load module), as shown in Figure 10-47.

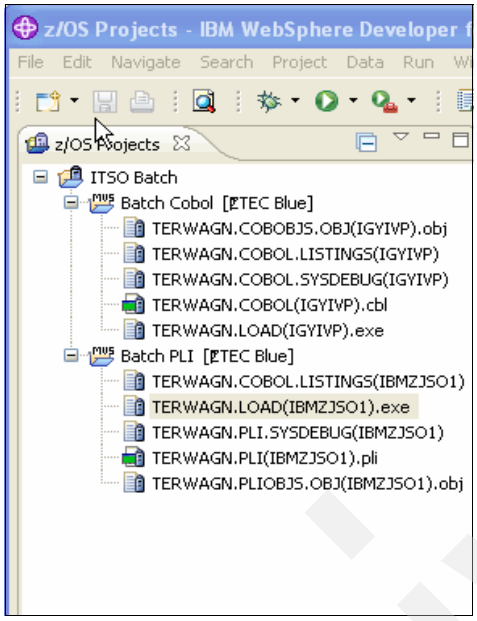


Figure 10-47 View of the Batch project after rebuilt

   – At least one error: In this case, you can see the messages from the compilers in the Remote error list. Double-clicking each error message opens the related source and points directly to the line with the error, as shown in Figure 10-48 through Figure 10-50 on page 285.

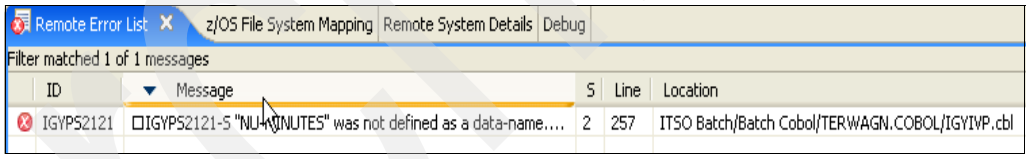


Figure 10-48 Remote error list

Archived

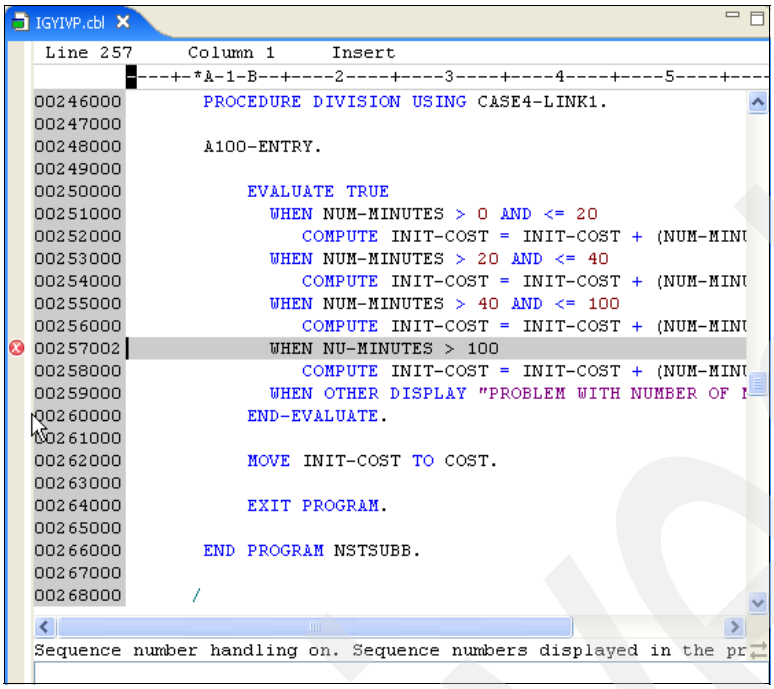


Figure 10-49 Error shown in the source

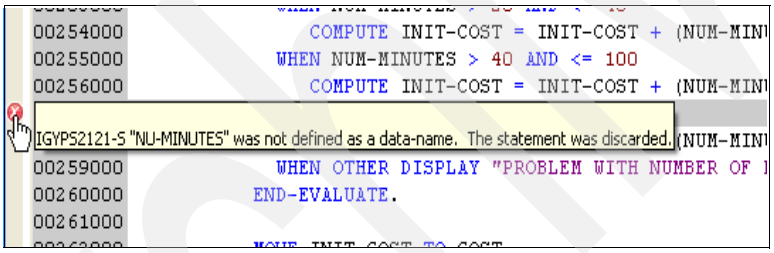


Figure 10-50 Error message shown with hovering cursor

Archive

9. At this point, you are able to run or debug the programs. Right-click the executable programs (suffixed by .exe) and select **Debug Application**. It will generate and submit a JCL. This JCL includes your IP address to be used by IBM Debug Tool when starting a debug session, as shown in Figure 10-51.

Important: You need the IBM Debug Tool for z/OS installed on the mainframe if you wish to debug using WD/z. IBM Debug Tool for z/OS helps you examine, monitor, and control the execution of programs written in Language Environment® assembler, C, C++, COBOL, or PL/I on z/OS and OS/390®. You can debug a program using a 3270 interface (Main Frame Interface (MFI)), a graphical interface (WD/z) or no interface (batch mode using a command and a log file).

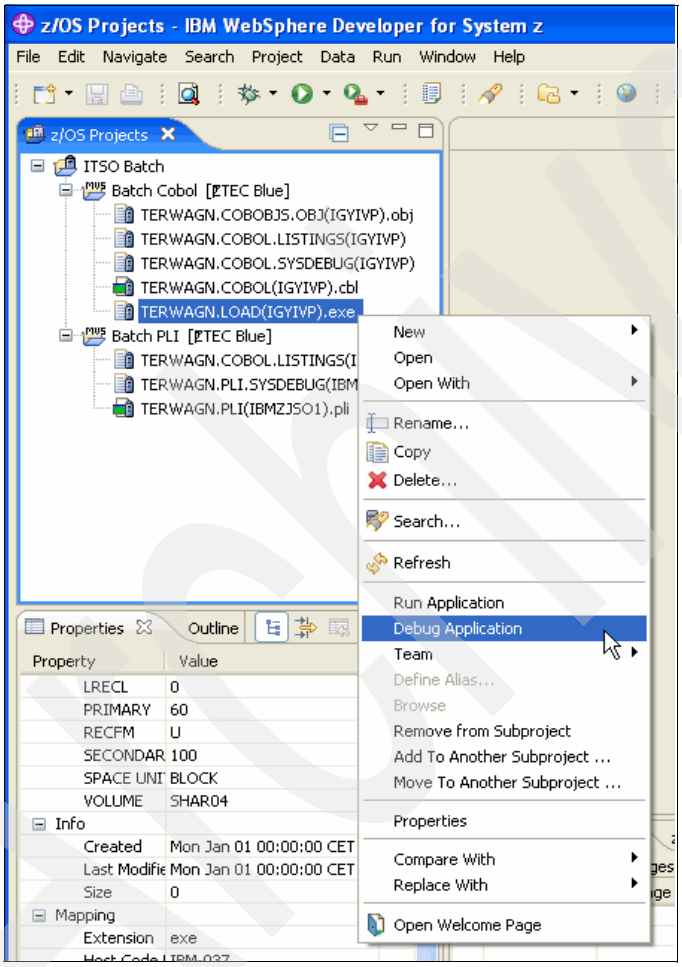


Figure 10-51 Starting debug program

10. When the job starts to run in debug mode, IBM Debug Tool for z/OS will contact WD/z using the IP address it has received from the JCL, as shown in Example 10-1.

Example 10-1 IP address included in the generated JCL

```
//GO    EXEC   PROC=ELAXFGO,GO=IGYIVP,
//           LOADDSN=TERWAGN.LOAD,
//         PARM.RUN=('/TEST(,,,TCPIP&&9.164.169.44%8001:*)')
```

When WD/z is contacted by Debug Tool, it switches to the Debug perspective. You could see the following pop-up window asking if you accept this switch. Click **Yes**. You can check the box **Remember my decision** if you wish, as shown in Figure 10-52 and Figure 10-53.

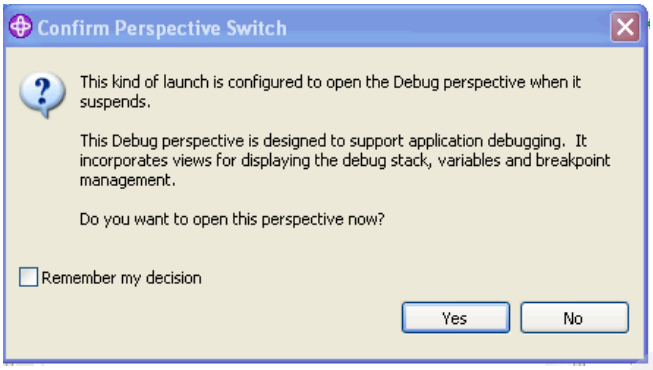


Figure 10-52 Confirm Perspective Switch

Tip: If you do not get the debug perspective, try to verify if the Debug daemon is listening. Manually switch to the Debug perspective and check the daemon's status (see Figure 10-53).

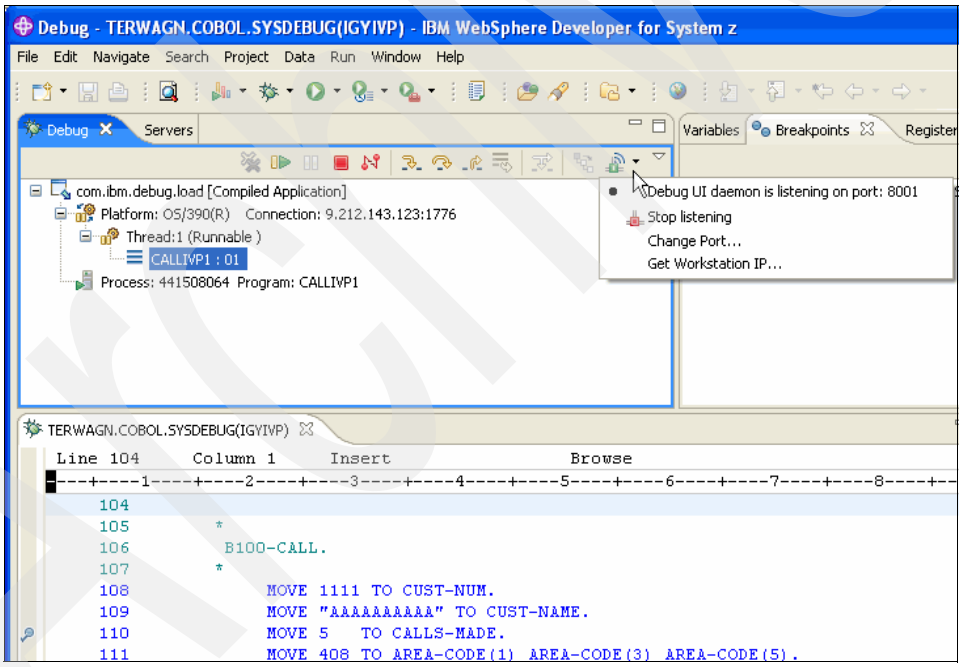


Figure 10-53 Verifying the Debug daemon status

11. The Debug perspective opens with several views.The most important are the **So**urce view, the Variables/Monitor view, and the Debug Console view, as shown in Figure 10-54.

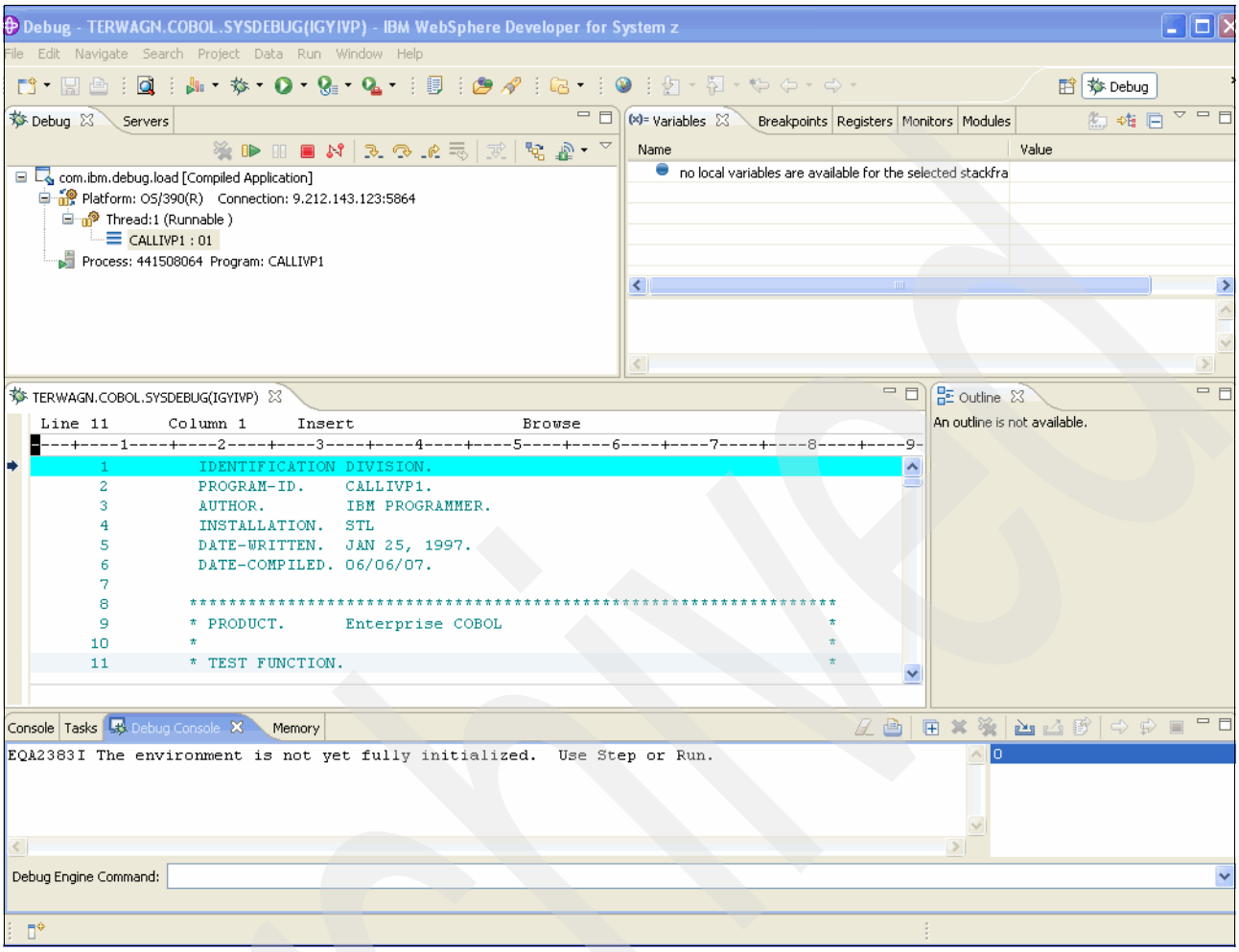


Figure 10-54 Debug perspective

12. We are going to add a breakpoint. In the source view, scroll down the program, select a line of code, right-click it, and select **Add breakpoint**. A bullet is added in front of the line chosen. You can right-click this bullet to edit or remove it. Another way to add or remove a breakpoint is double-clicking in front of the line (in the grey column), as shown in Figure 10-55 on page 289 through Figure 10-57 on page 289.

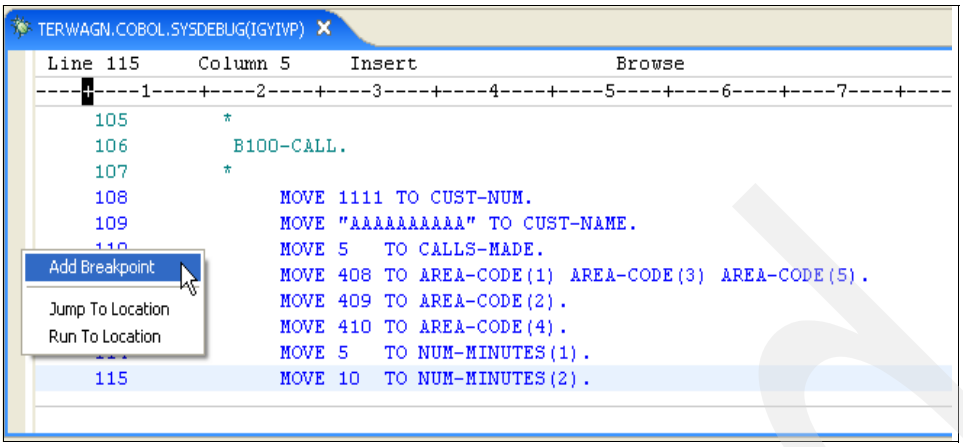


Figure 10-55 Adding a breakpoint

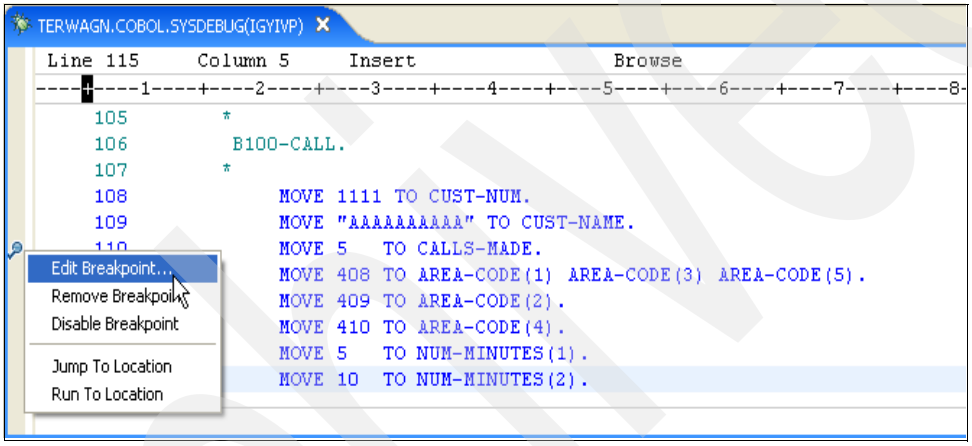


Figure 10-56 Edit a breakpoint

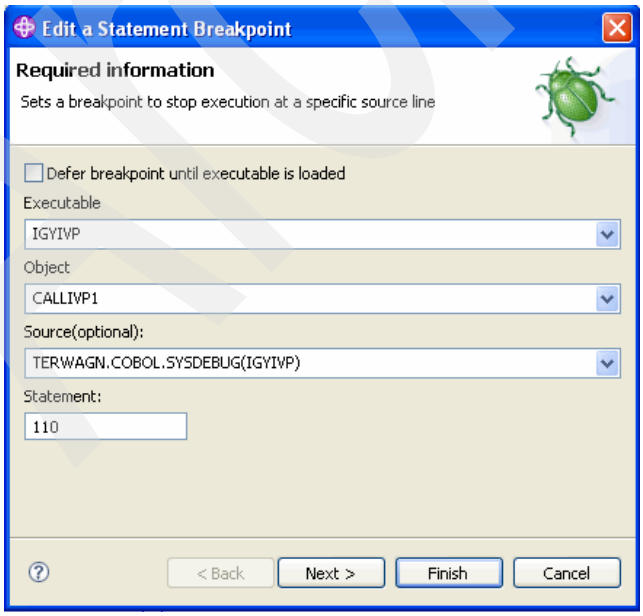


Figure 10-57 Edit a statement breakpoint

13. We will animate the debug session by using the functions Run, Step, Step into, and Step over. There are buttons associated with this actions. This buttons are in the Debug view, as shown in Figure 10-58. These actions are available by using function keys, as shown in Figure 10-59.

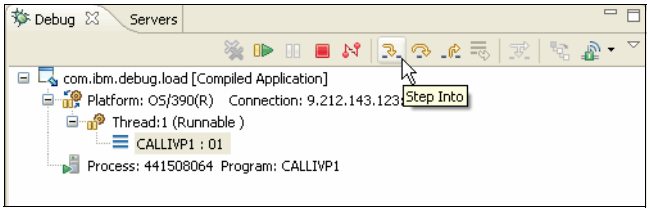


Figure 10-58 Debugging buttons in the Debug view

Resume	F8	Debugging
Run to Line	Ctrl+R	Debugging
Step Into	F5	Debugging
Step Into Selection	Ctrl+F5	Debugging
Step Over	F6	Debugging
Step Return	F7	Debugging
Monitor Expression	Ctrl+Shift+M	Debugging Compiled Languages
Show Stopping Thread	Ctrl+F9	Debugging Compiled Languages

Figure 10-59 Functions keys associated with Debug actions

Tip: You can change these associated keys in the Preferences window. To access it, select **Window** → **Preferences**. Type Keys in the search box.

14. Now we are going to see several ways to view, monitor, and modify variables contents:
    - Moving the mouse over the variables allows you to see the variable contents, as shown in Figure 10-60.

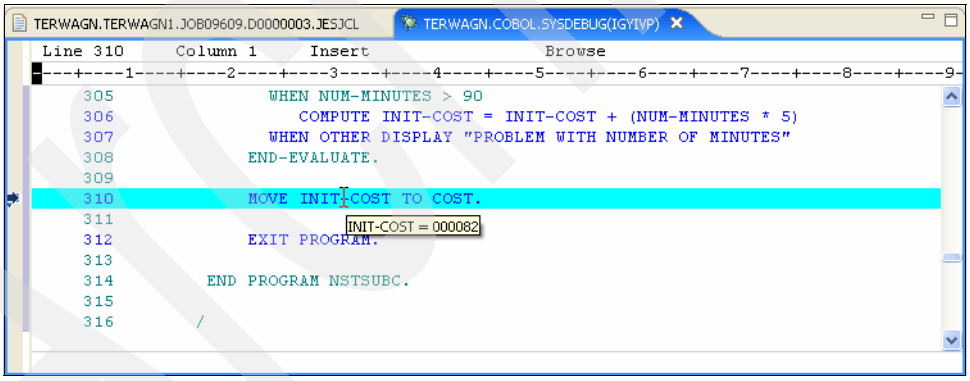


Figure 10-60 View of variable

    - Automonitor: When the automonitor is on, the variables of the current line are automatically shown in the monitor view. You can turn on the automonitor by doing these steps:
        i. Go to the Debug console view and enter the command **`set automonitor on`**, as shown in Figure 10-61 on page 291 and Figure 10-62 on page 291.

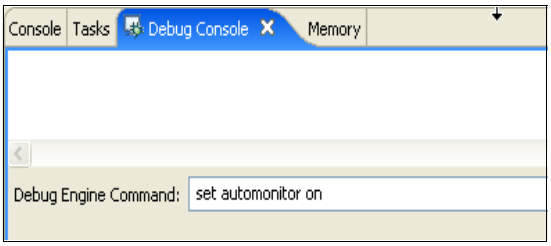


Figure 10-61 Debug console: enter the command

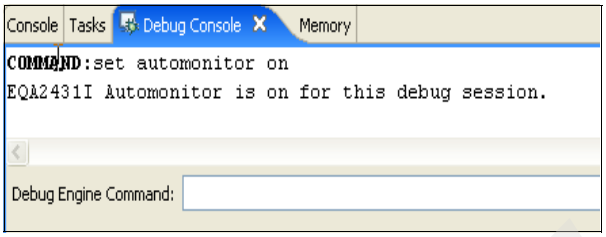


Figure 10-62 Debug console: result of the command

ii. Go to the Monitor view to see the actual monitored variables, as shown in Figure 10-63.

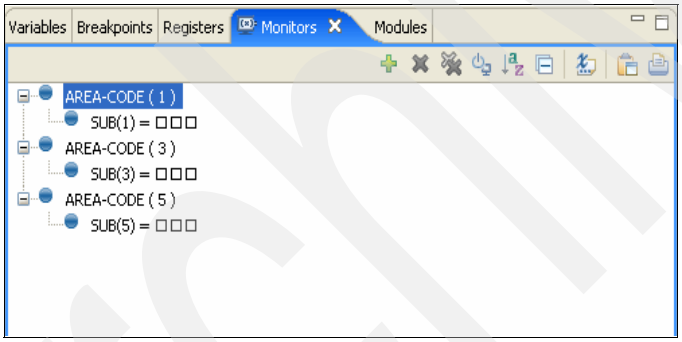


Figure 10-63 Monitor view: automonitored variables

Archived

d. Go to the Source view, move the cursor to a variable, right-click it, and select **Monitor Expression**. The variable will be added in the Monitors view. You can modify the content of the variable by double-clicking the value, as shown in Figure 10-64 through Figure 10-66.

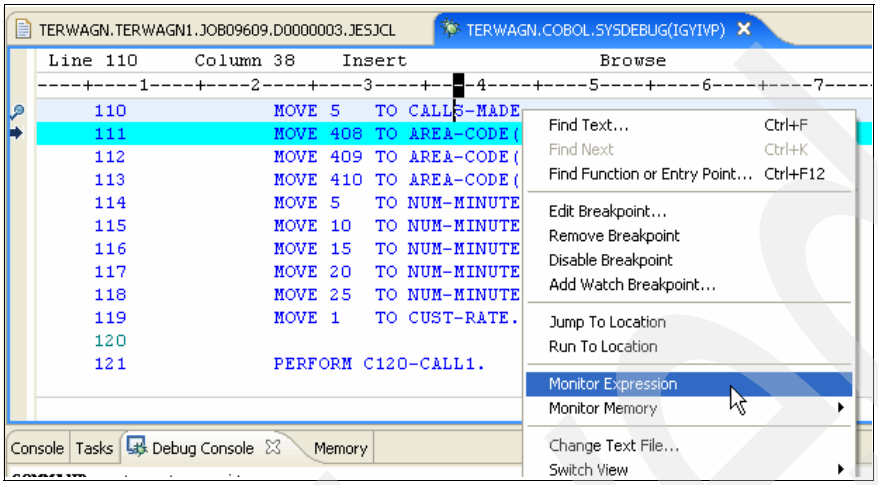


Figure 10-64 Monitor Expression

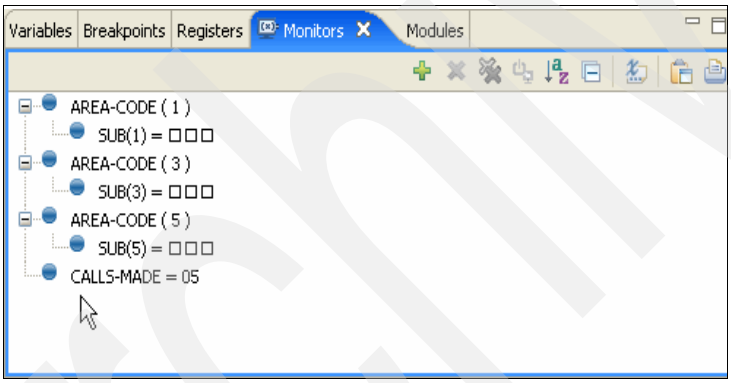


Figure 10-65 Monitors view

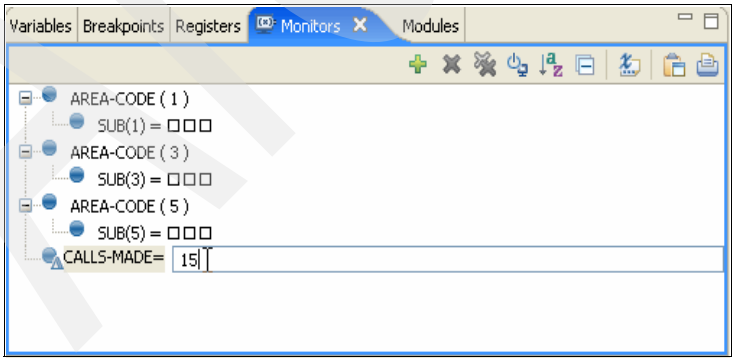


Figure 10-66 Modify variable content

e. Go to the Variables view to see the full storage of your program, as shown in Figure 10-67.

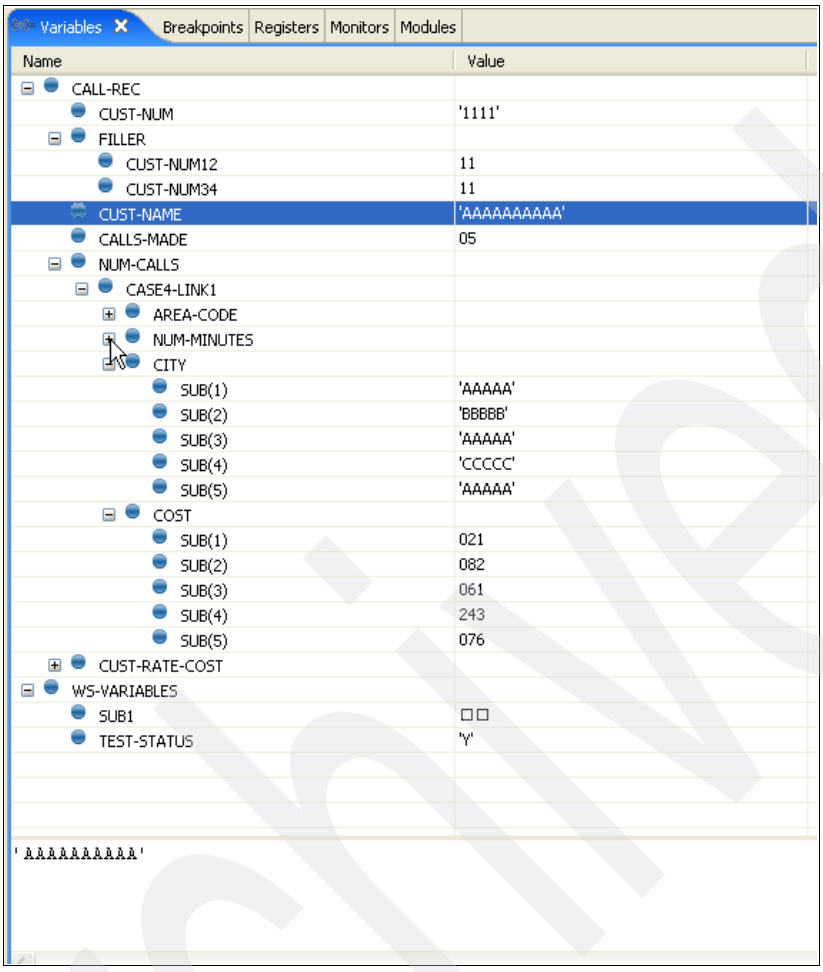


Figure 10-67 Variables view

Archived

f. Go to the Source view, move the cursor to a variable, right-click it, and select **Monitor Memory**. Select the type of display you wish. A Memory view is added at the bottom of the perspective, as shown in Figure 10-68 and Figure 10-69.

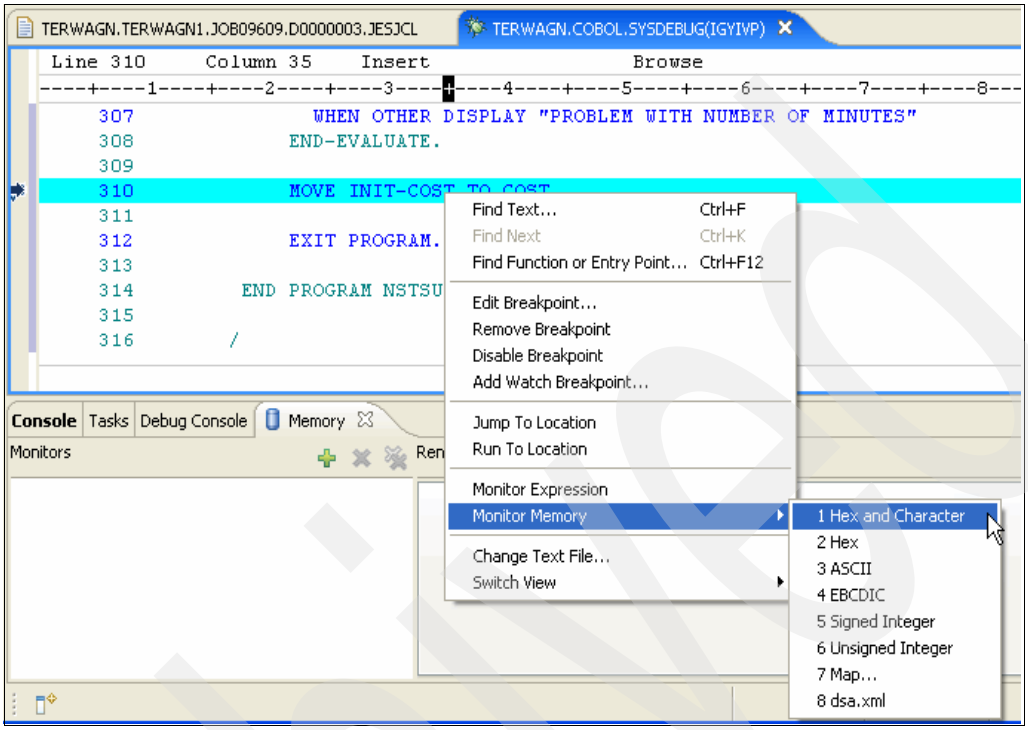


Figure 10-68 Monitor Memory

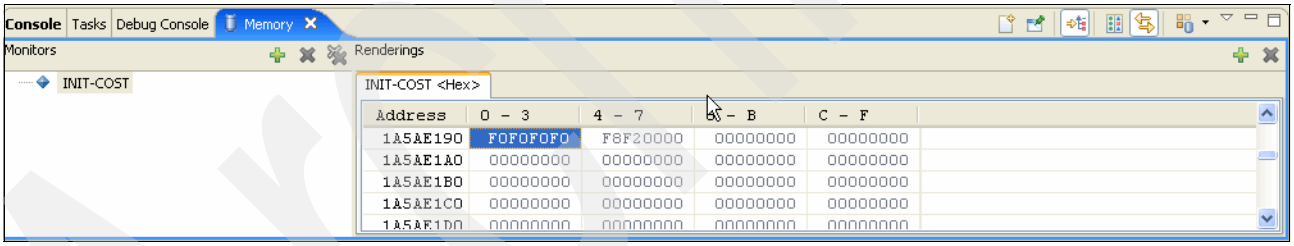


Figure 10-69 Memory view

15. You are now ready to develop and debug batch programs using WD/z.

10.4 Developing and debugging a CICS application

Note: The sample CICS application used in this section is delivered with CICS. This sample program may already be installed on your system. If you need it, we provide it in the Web material associated with this book; refer to Appendix B, “Additional material” on page 645 for more information.

This section describes how to develop, compile, and debug a couple of CICS programs using WD/z. Because there are several methods to manage your development projects, we decided to choose one that allows you to quickly start by creating a new project and as many subprojects as programs.

Do these steps:

1. Using the z/OS Projects perspective, right-click in the **z/OS Projects** pane, click **New**, and select **z/OS Project...**, as shown in Figure 10-70.

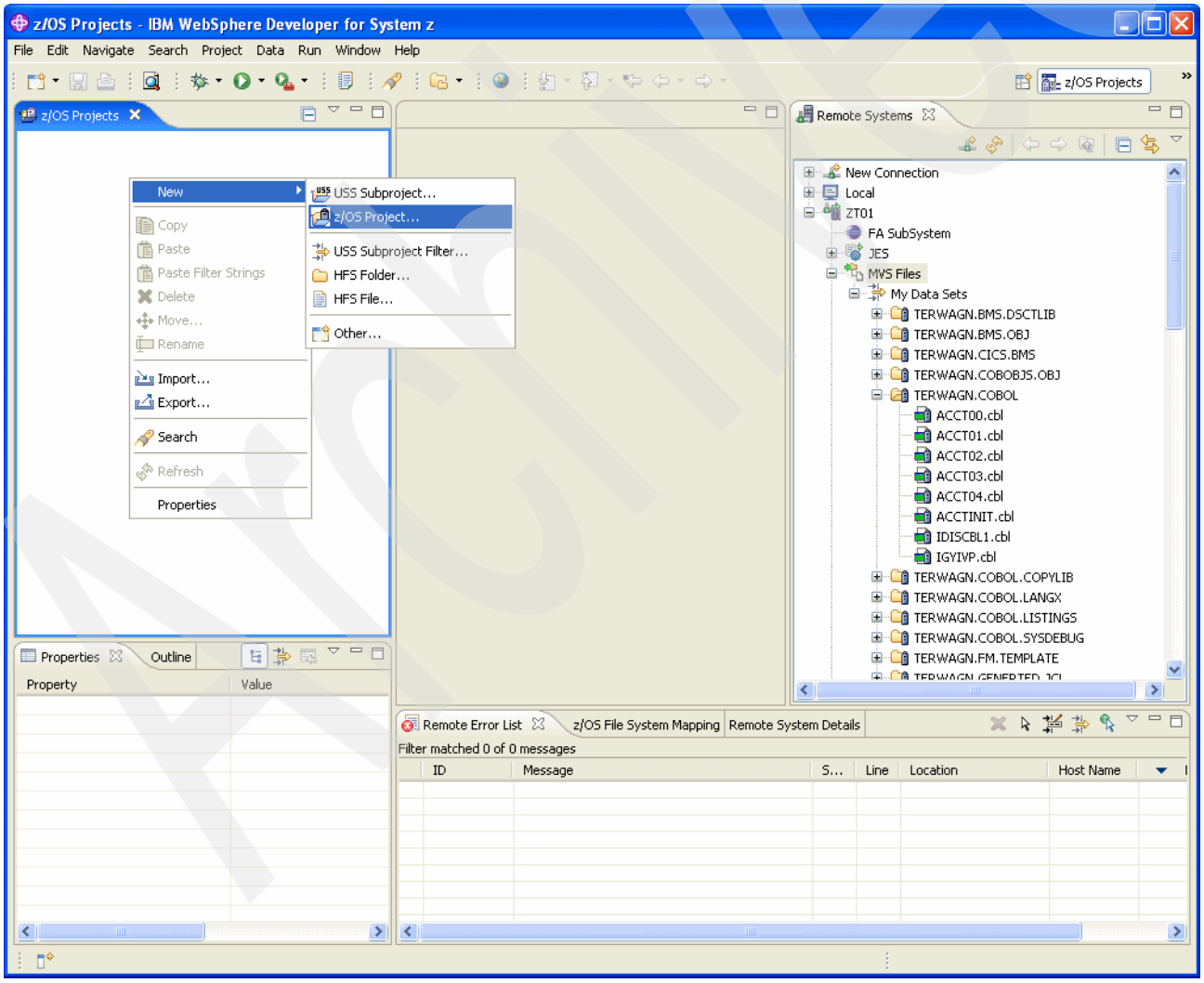


Figure 10-70 CICS application: New z/OS project

2. Type a Project Name of your choice and check **Create an MVS subproject**, as shown in Figure 10-71.

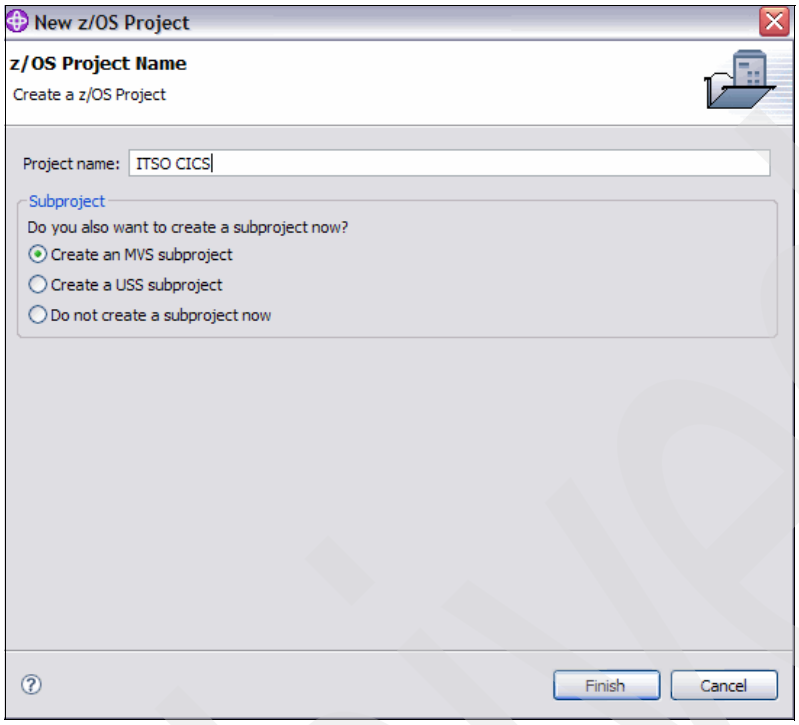


Figure 10-71 CICS application: Create z/OS project

3. Enter the name of the subproject. We suggest using the name of the program. For example, the first CICS program is ACCT00. Click **Finish**, as shown in Figure 10-72 on page 297.

Archive

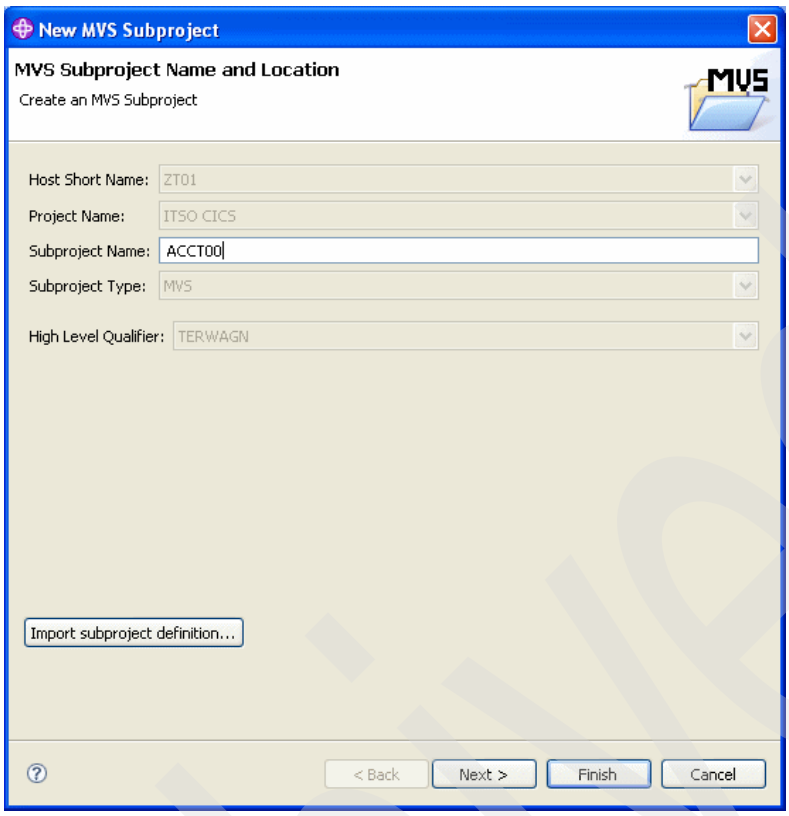


Figure 10-72 CICS application: Create z/OS subproject

4. Right-click the subprojetct just created and review the properties by selecting **Properties** in the menu, as shown in Figure 10-73.

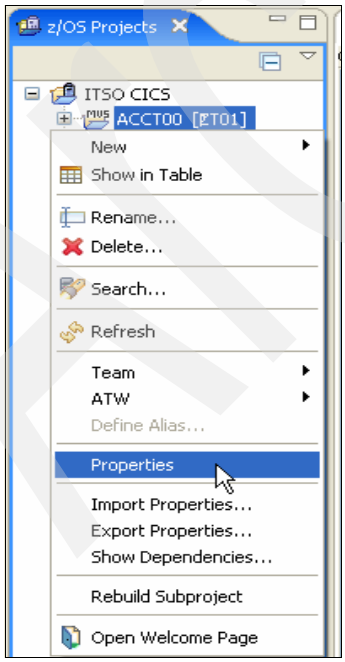


Figure 10-73 Subproject properties

5. Normally, if you correctly set up the properties associated with the MVS Files under your connection, you should get the correct parameters except for the ones specific to the CICS application. Do these steps to set up the properties:

   a. Verify the COBOL settings if the **Use CICS** box is checked, as shown in Figure 10-74.

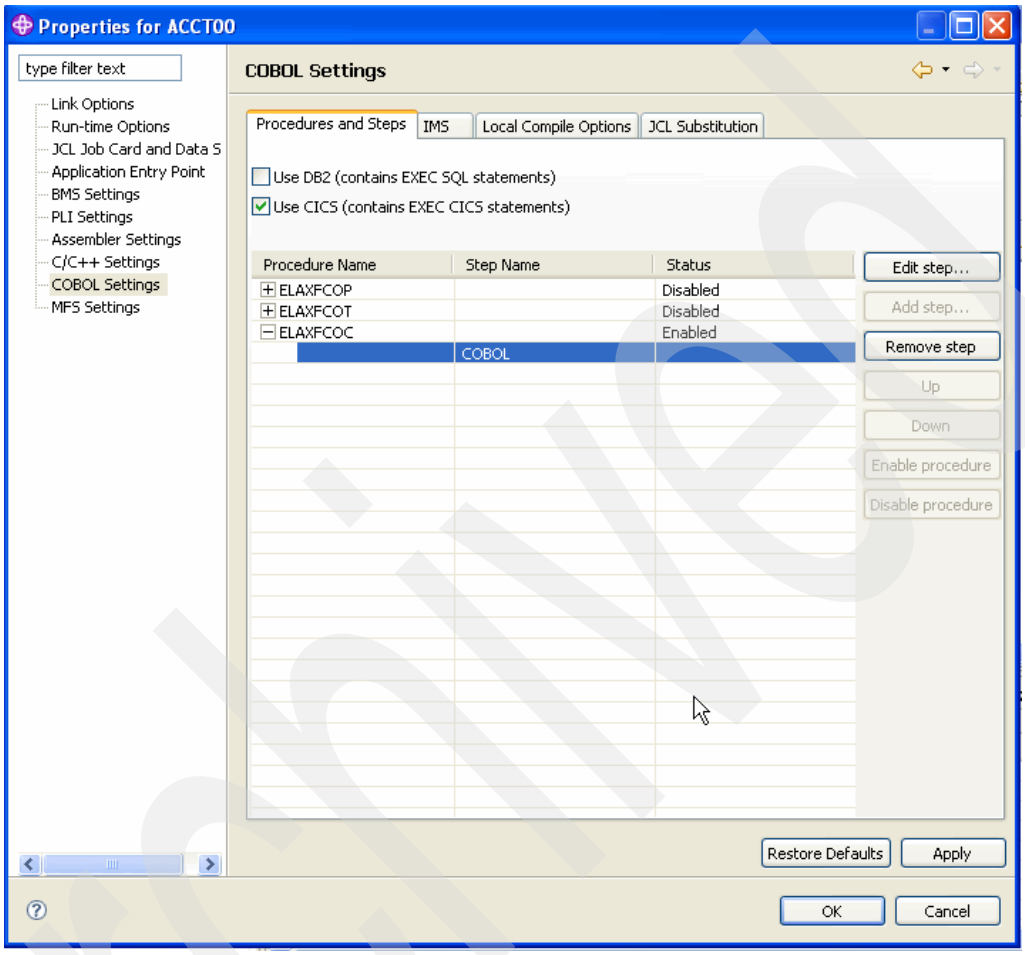


Figure 10-74 COBOL setting for a CICS program

   b. Click the **Edit Step** button for the enabled procedure (probably the third one named ELAXFCOC*)*. Verify the value for Copy libraries, where you should have the library where the standard CICS copybooks are available on your system, as shown in Figure 10-75 on page 299.

Archive

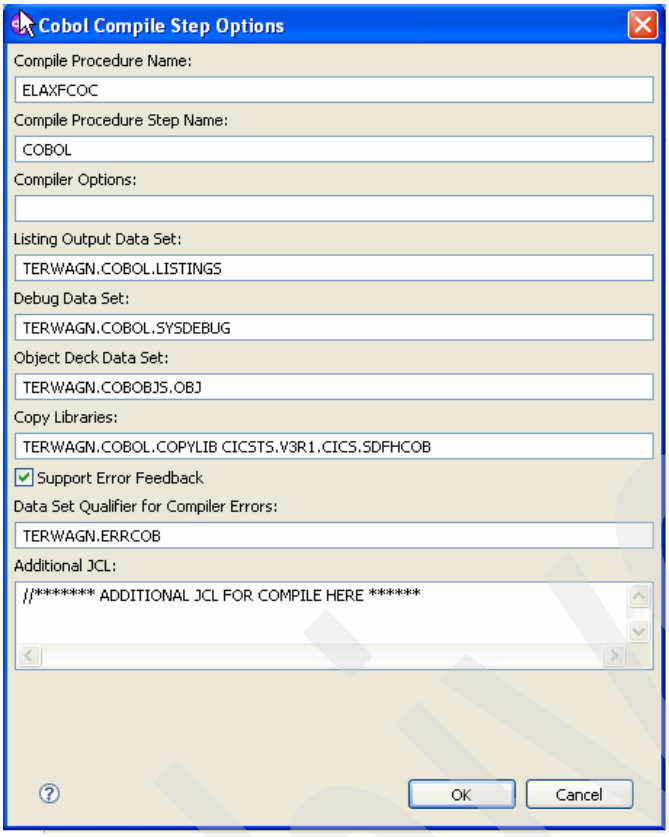


Figure 10-75 CICS COBOL Compile Step Options

c. As we should do the same setup for each subproject, we export the properties to an XML file. Right-click the subproject and select **Export properties...**, as shown in Figure 10-76.

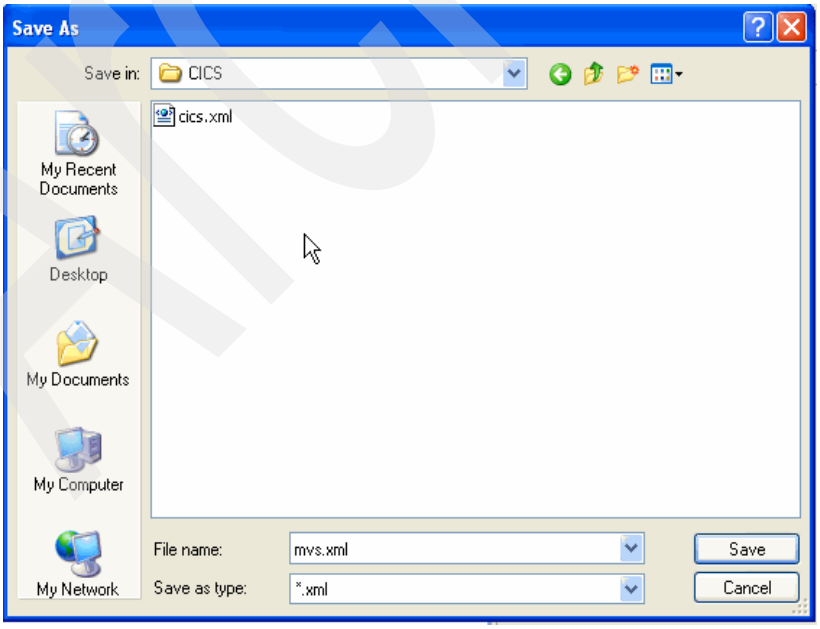


Figure 10-76 Exporting the properties for the CICS subproject

6. Create subprojects for each of the remaining CICS programs (ACCT01, ACCT02, ACCT03, and ACCT04). Do the following steps for each program:
   a. Right-click the project and click **Create MVS Subproject**.
   b. Name the subproject (we suggest using the name of the program).
   c. Click the **Import subproject definition...** button and select the properties file exported from the first subproject.
7. Now, we add the programs to the subprojects. Do the following steps for each program:
   a. Open the Connection view for the MVS Files and PDS where the CICS COBOL sources are located.
   b. Select the first source (**ACCT00**), right-click it, and select **Add selected resources to subproject**, as shown in Figure 10-77.

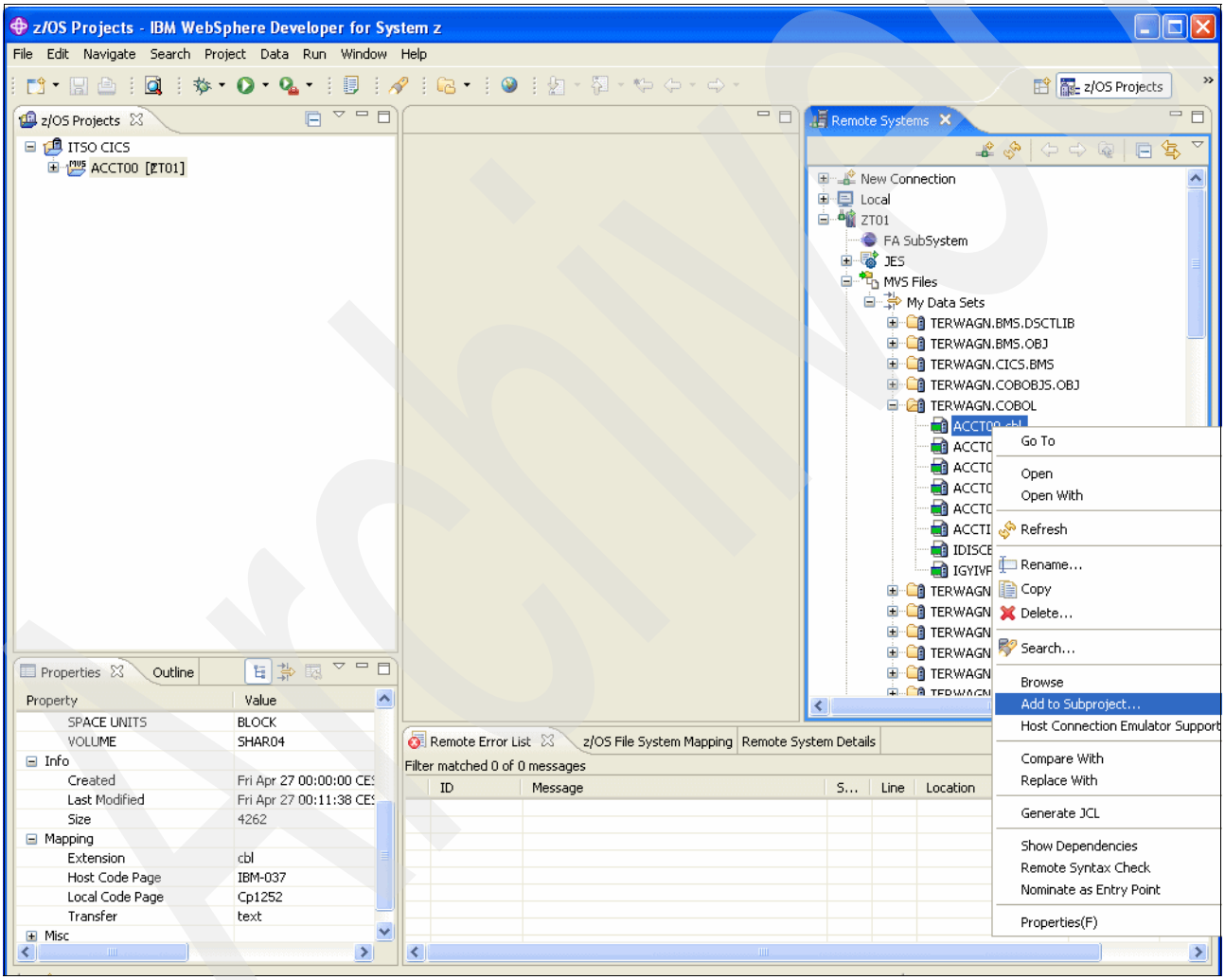


Figure 10-77 Adding the CICS COBOL source to the subproject

   c. Choose the corresponding subproject (for example, ACCT00 for the program ACCT00) and click **Finish**, as shown in Figure 10-78 on page 301.

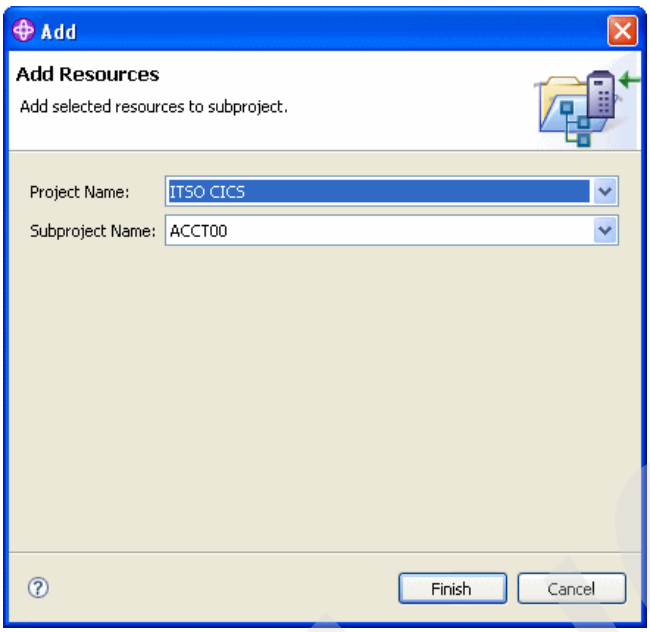


Figure 10-78 Add resource to the subproject

8. Create a new subproject for MAP BMS. Right-click the project and select **Create MVS Subproject**. Name the subproject and click **Finish**, as shown in Figure 10-79.

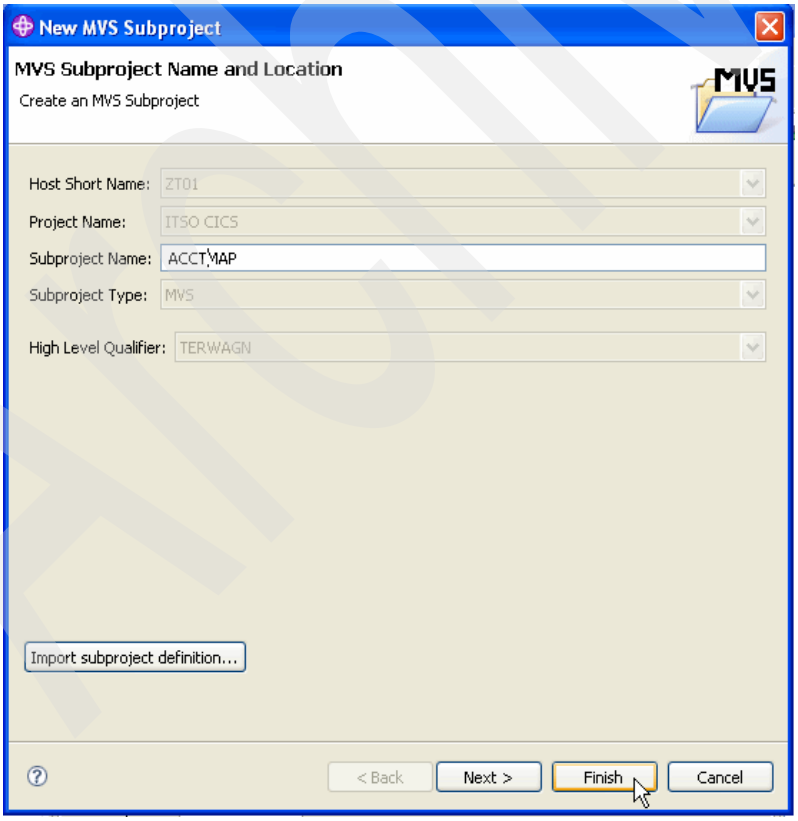


Figure 10-79 Create MVS subproject for MAP BMS

9. Add the file ACCTSET to this subproject, as shown in Figure 10-80 on page 302.

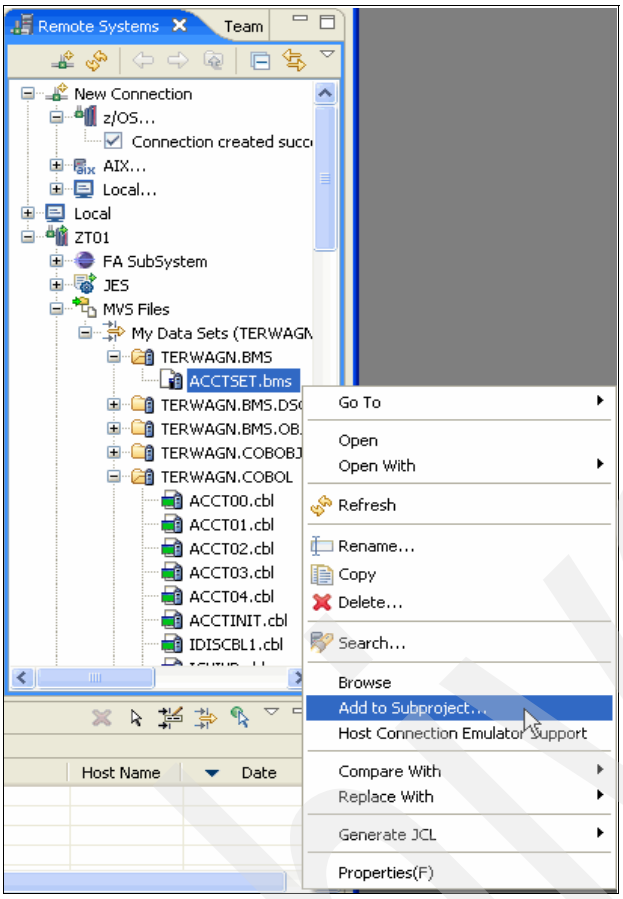


Figure 10-80 Add MAP BMS to subproject

10. When done, right-click the subproject and select **Properties**, as shown in Figure 10-81 on page 303.

Archived

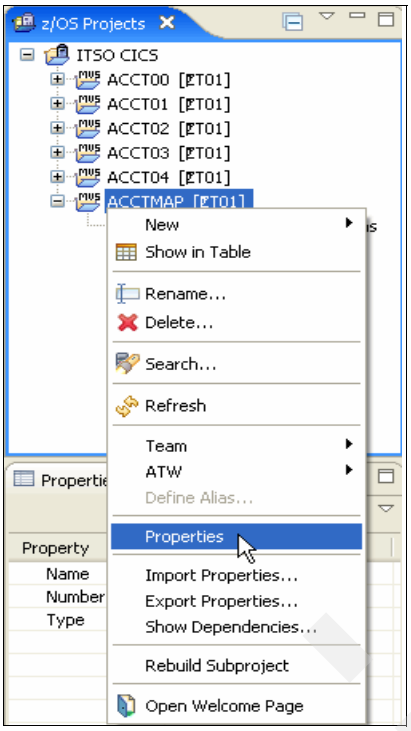


Figure 10-81 Properties of MVS subproject

11. Select **BMS Settings**, expand the procedure **ELAXFBMS**, select the step **ASMMAP**, and click the **Edit Step** button, as shown in Figure 10-82.

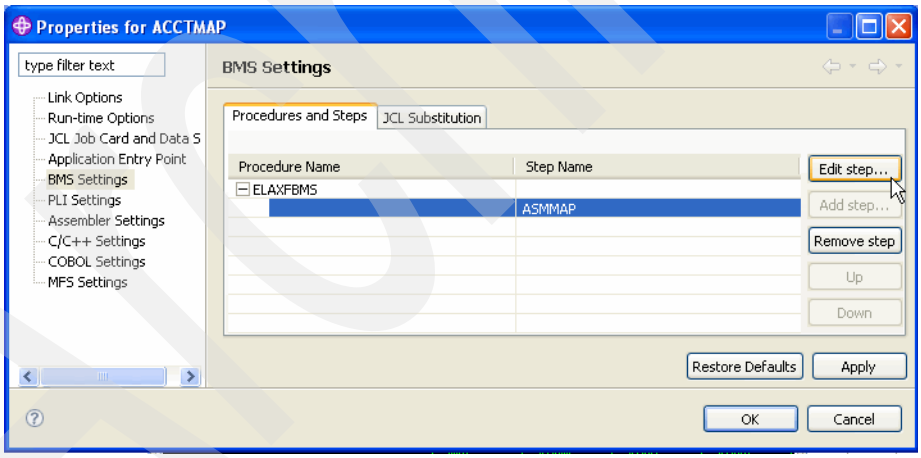


Figure 10-82 Edit ASMMAP step

12. Verify that the BMS Map Object Library (type your object library) and the BMS DSCTLIB Library are not empty and correct the situation if they are. Click **OK**.

Archived

13. We generate a JCL to assemble and link the map. Right-click the MAP source in the subproject and select **Generate JCL** → **For Assemble Link**, as shown in Figure 10-83.

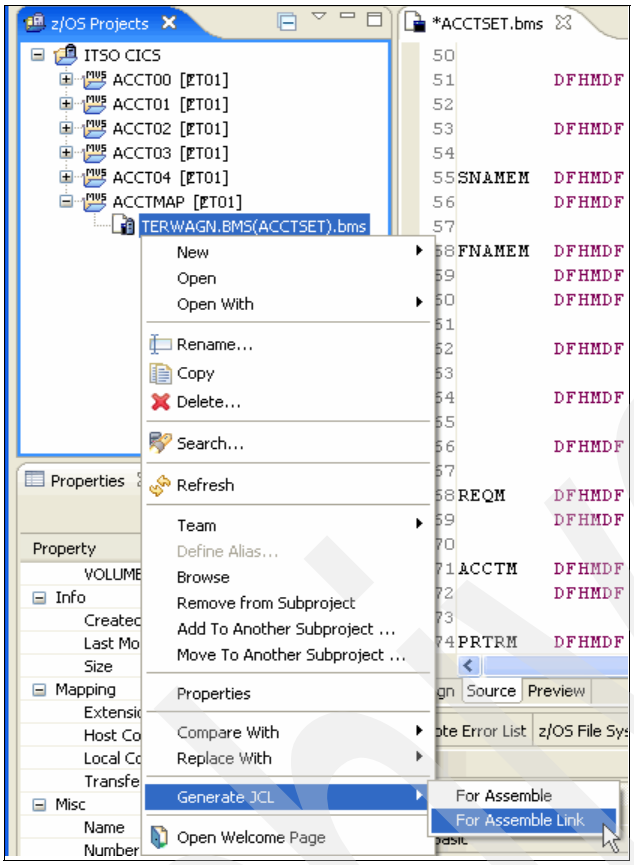


Figure 10-83 Generate JCL to Assemble Link the MAP

14. A pop-up window asks for the job name, the PDS name to store the JCL, and the JCL name. Use the values proposed by the wizard. Click **OK**. When you see the JCL Generation Message, click **OK**. The JCL is added to your subproject, as shown in Figure 10-84 through Figure 10-86 on page 305.

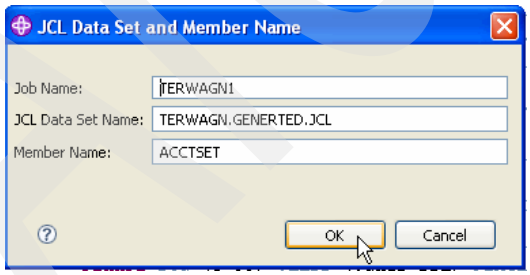


Figure 10-84 JCL generation pop-up window

Archived

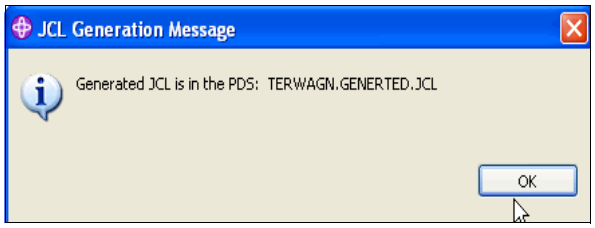


Figure 10-85 JCL Generation Message

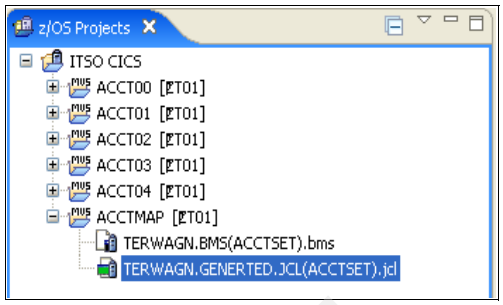


Figure 10-86 JCL added in subproject

15. Right-click the JCL and select **Submit**. The Job Submission Confirmation pop-up window appears, as shown in Figure 10-87 and Figure 10-88 on page 306.

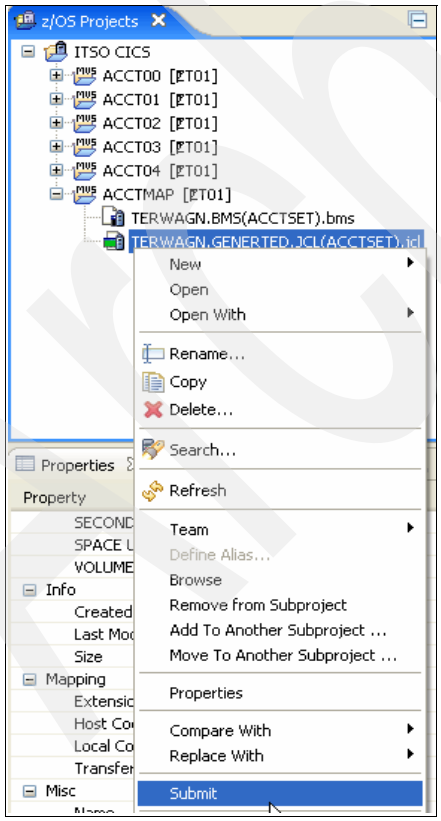


Figure 10-87 Submit JCL

Archived

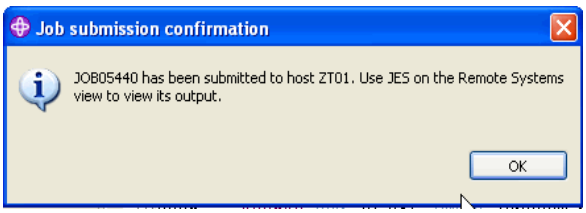


Figure 10-88 Job submission confirmation

16. Open the Remote Systems view and click **My Jobs**. Do not forget to refresh the job list by right-clicking **My jobs** and selecting **Refresh**. You should see the submitted job and its sysouts. Double-click the first one (JES2:JESMSGLG) to see the result, as shown in Figure 10-89 and Figure 10-90.

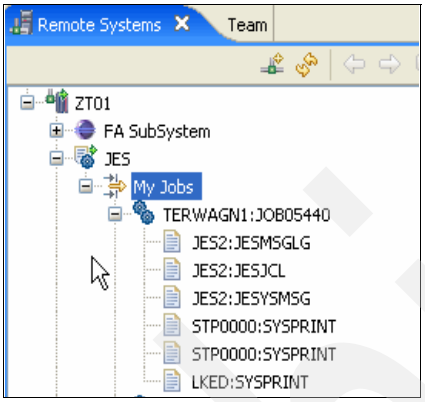


Figure 10-89 Job monitoring My Jobs

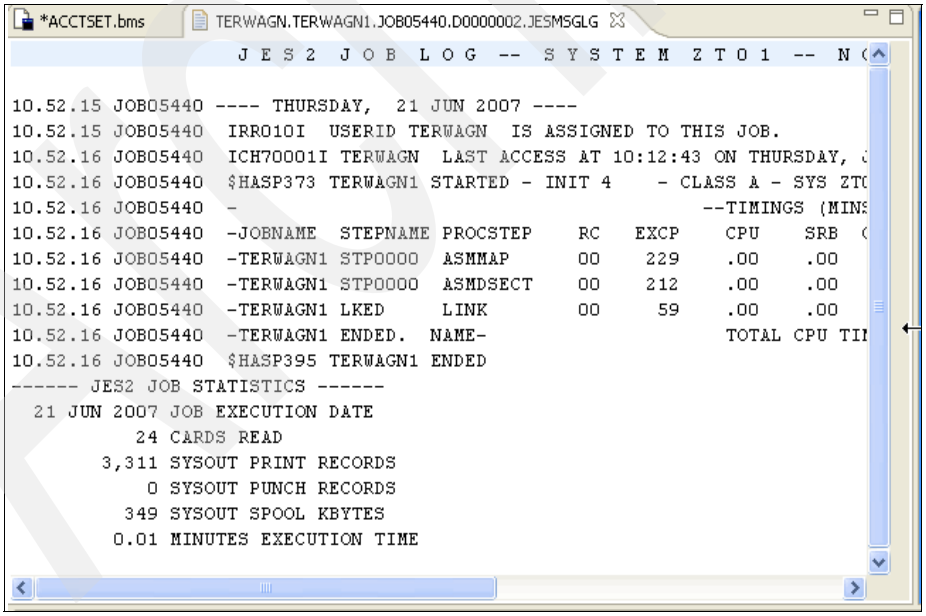


Figure 10-90 JES messages

Tip: You can also right-click **My jobs** and select **Show in table**. The job list appears in the Remote Systems Details view. This view shows more information about the jobs (especially the Return Code).

17. Open the Project view, right-click the CICS project, and select **Rebuild Project**. The application automatically generates the JCL to compile and link the programs. The application submits the JCL to the mainframe (JES) and parses the errors from the compiler, as shown in Figure 10-91.

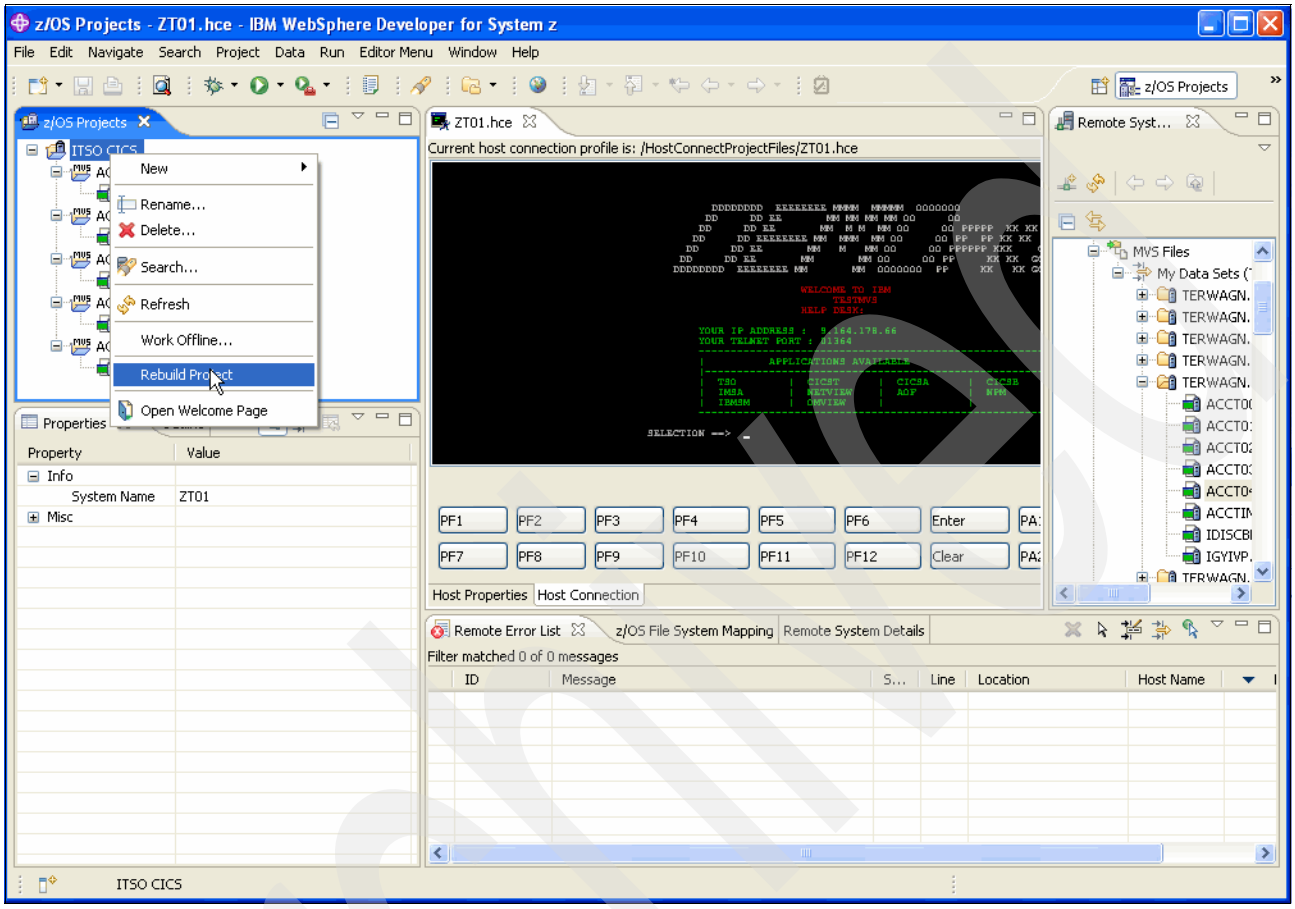


Figure 10-91 Rebuild CICS Project

18. You should see, under each subproject, several new members from different PDS, that is, objects, load modules, listings, and so on, as shown in Figure 10-92.

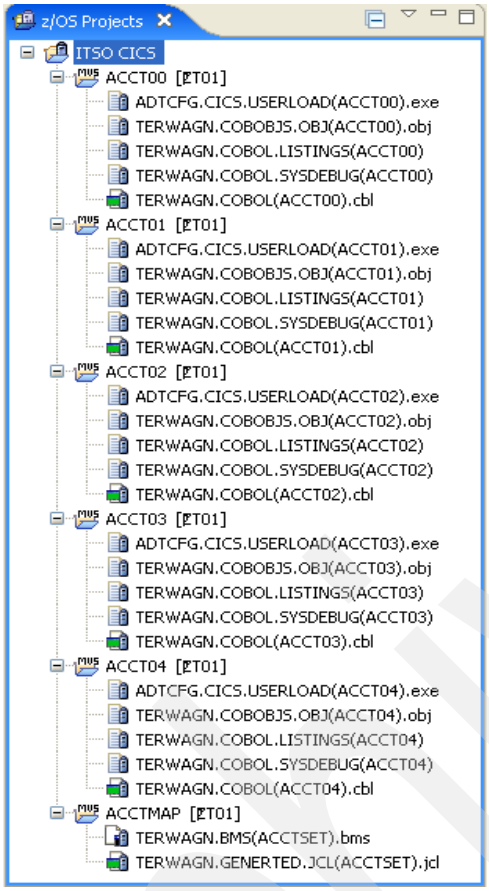


Figure 10-92 CICS Project built

19. We are now ready to debug the CICS transaction. We need a CICS terminal. In the Remote Systems view, right-click your system and select **Host Connection Emulator Support**, as shown in Figure 10-93 on page 309.

Archived

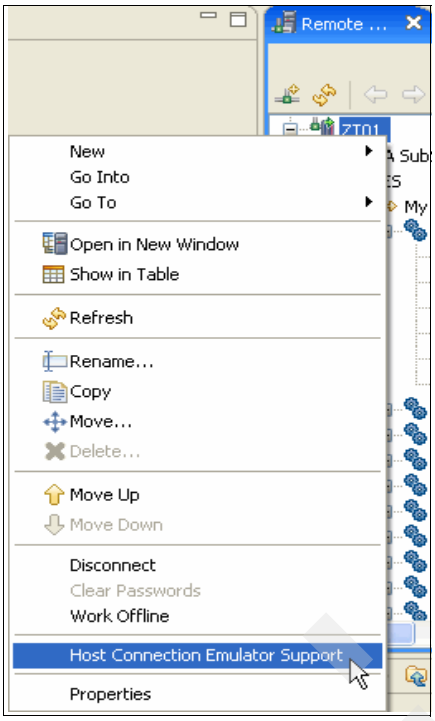


Figure 10-93 Starting a Host Connection Emulator

Archived

20. A terminal view opens. Start your CICS application and log on to CICS, as shown in Figure 10-94 and Figure 10-95 on page 311.

> **Note:** If needed, you can change the properties of the terminal emulation by clicking the tab (see the bottom of the current view).

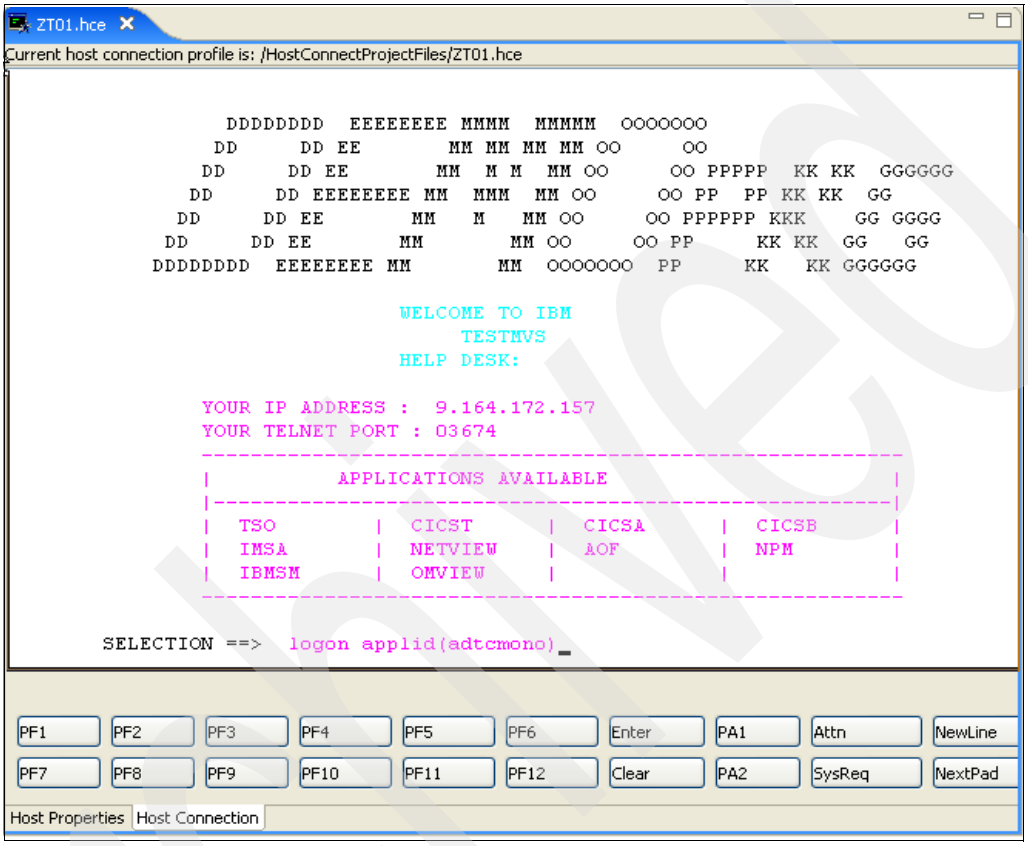


Figure 10-94 Terminal emulation

Archived

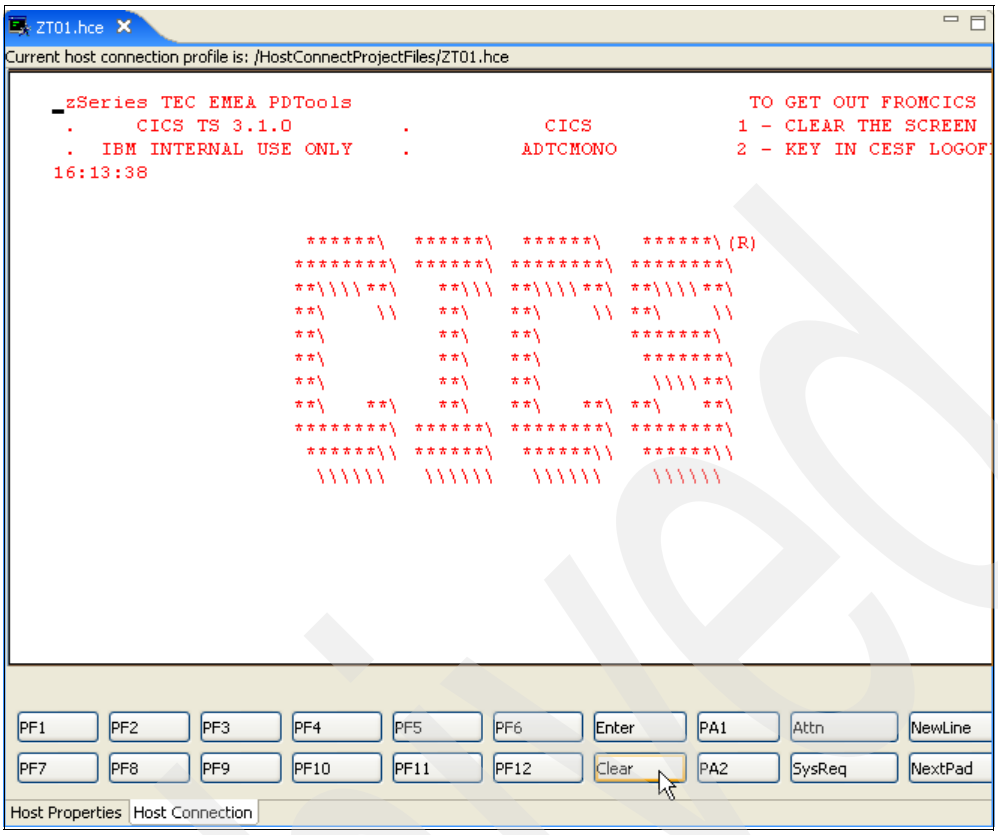


Figure 10-95 Log on to CICS

21. To start debugging CICS transaction, we have to define a Debug Profile in CICS. There are two features available to manage this profile: CADP (a CICS feature) and DTCN (a Debug Tool feature). Depending on your local installation, CADP or DTCN is available. For more information, refer to the *Debug Tool for z/OS Debug Tool Utilities and Advanced Functions for z/OS User's Guide*, SC19-1196.

Archived

The following instructions are based on the CADP transaction:

a. Type CADP and press Enter to open the Debugging Profile list, as shown in Figure 10-96.

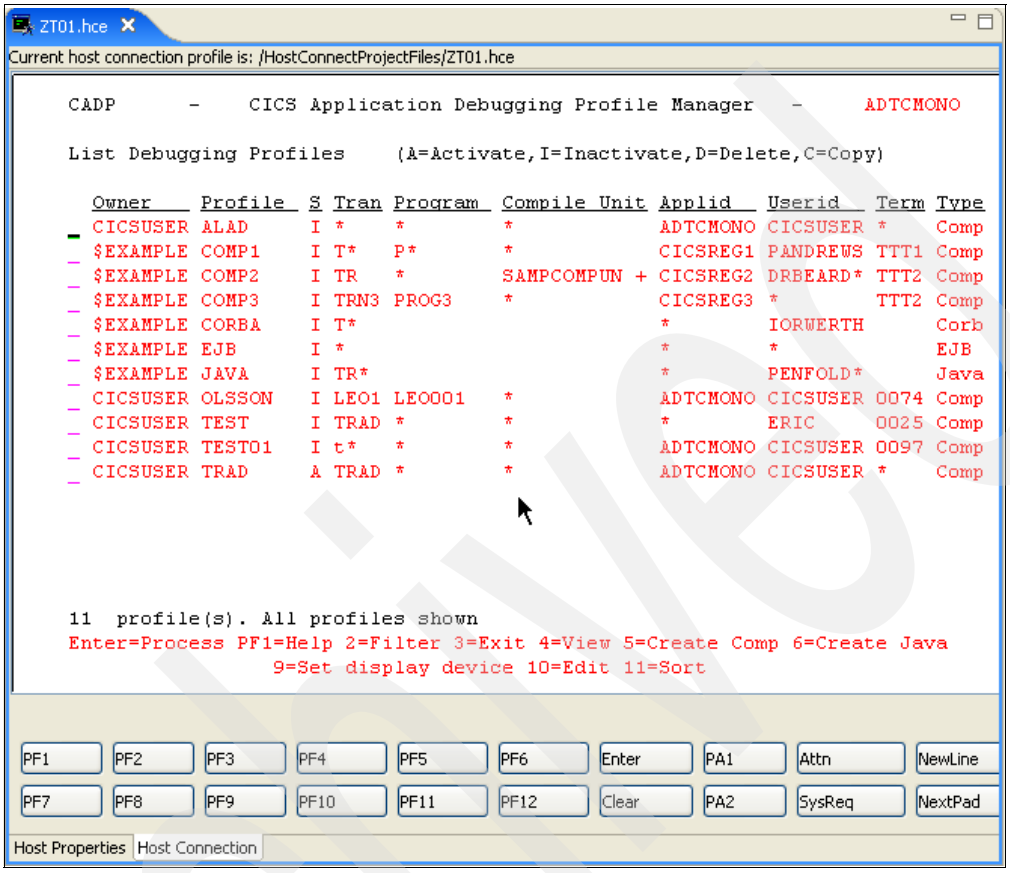


Figure 10-96 CICS Debugging Profile list

Archive

b. Press F5 (Create Comp). Enter a profile name (we entered MARC01) and the transaction code to debug (ACCT). All other parameters are automatically populated. Press Enter to create the profile. If it already exists, press F10 to replace it. Press F12 to return to the profile list, as shown in Figure 10-97 and Figure 10-98 on page 314.

Tip: If you want to debug a transaction called from outside the CICS, do not forget to remove the TERMID. Otherwise, the transaction will be only debugged when started from the current terminal. You can have several profiles for the same transaction.

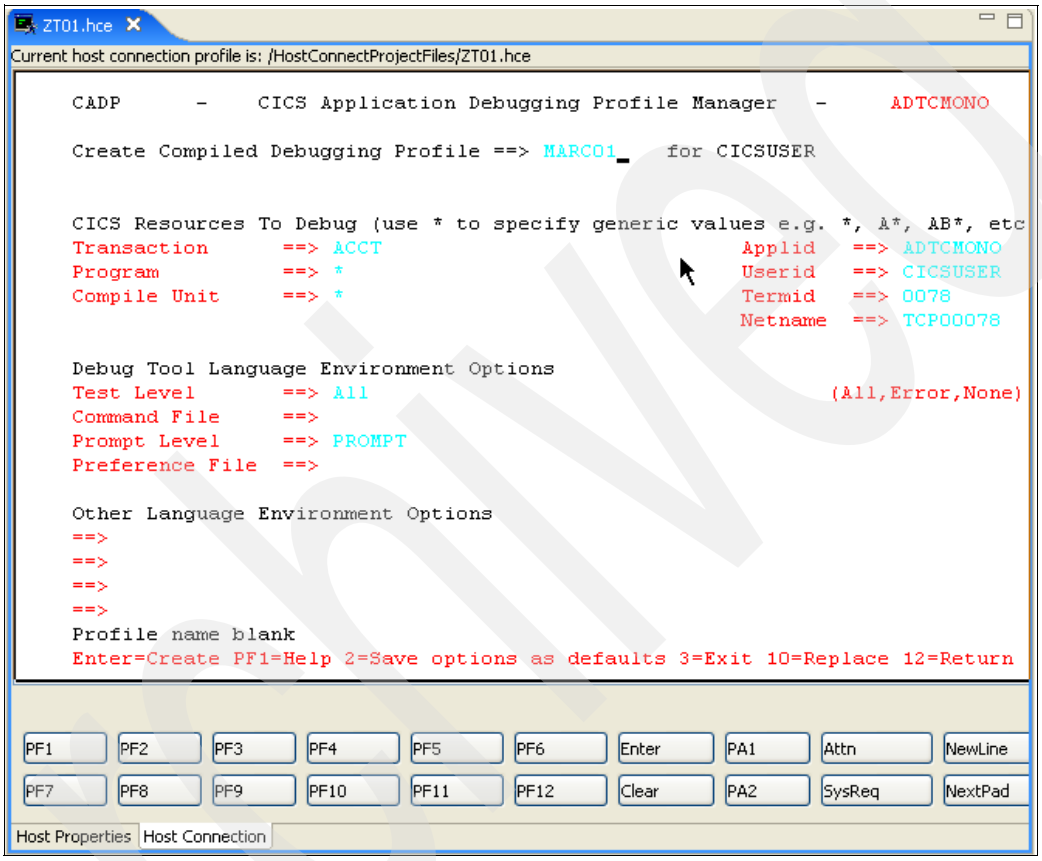


Figure 10-97 Debugging Profile creation

Archive

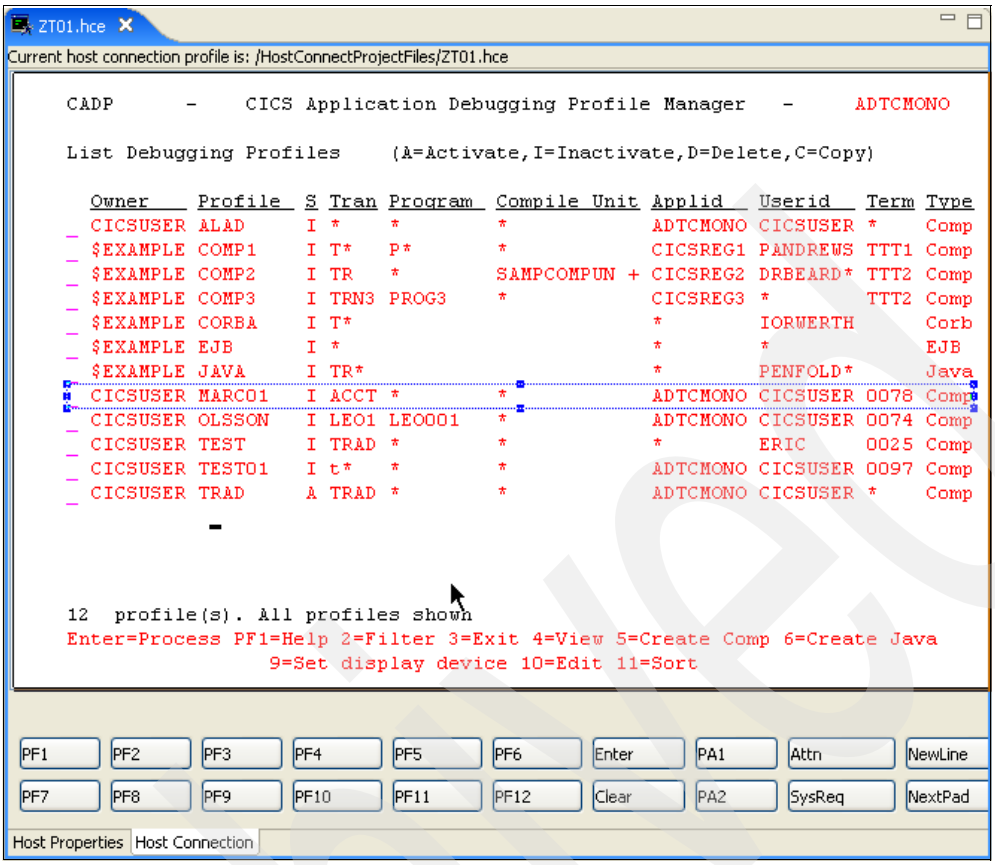


Figure 10-98 Debugging Profile list updated

c. A new profile appears in the list (named MARC01). Its status (see the third column) is inactive (I). This status means that the Debugging Profile exists, but no Debugging Session will be started at this point. Type A before the profile to activate it and press Enter.

d. At This point, you must find the IP address of your workstation and the Debug Daemon listening port. If do not know these values, open the Debug perspective, In the Debug view, click the **Debug Daemon** button. It should be green; if it is red, click it to start the daemon. Note in the first line of the drop-down list if the daemon is listening and at which port. Select **Get Workstation IP...** and a pop-up window appears, as shown in Figure 10-99 and Figure 10-100 on page 315.

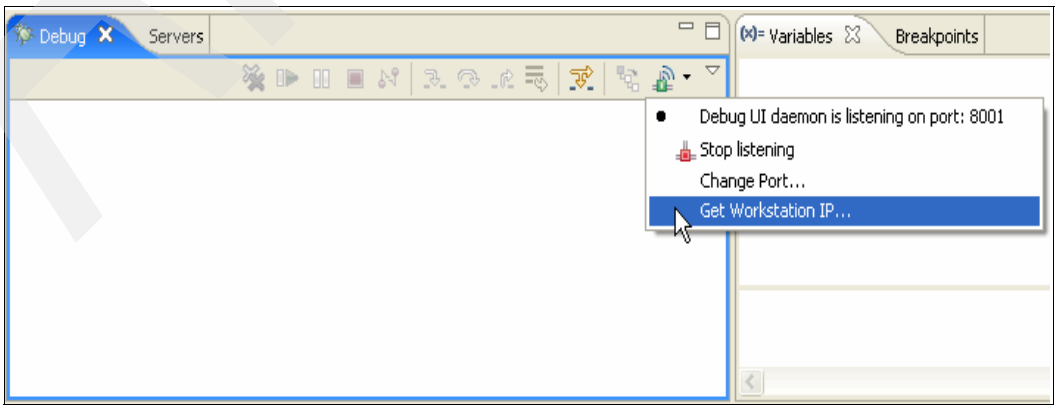


Figure 10-99 Debug perspective: Debug Daemon button

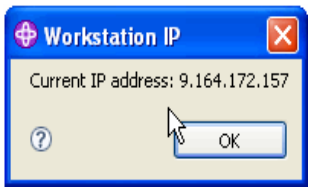


Figure 10-100 Workstation IP

e. Enter the right IP address and port and press Enter. Press F3 to quit, as shown in Figure 10-101.

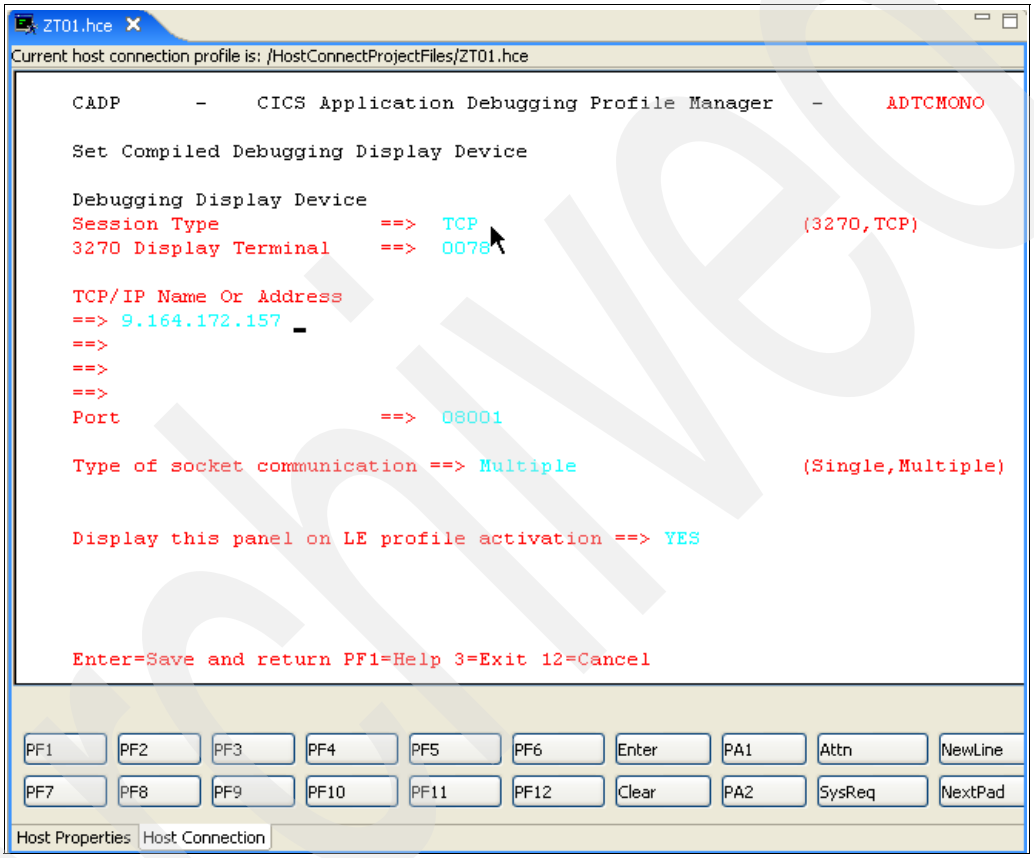


Figure 10-101 Debugging Profile activation

Archived

f. Enter the transaction code ACCT and press Enter. When WD/z is contacted by the Debugging Tool, it switches to the Debug perspective. A pop-up window could appear, asking if you accept the switch. Click **Yes**. You can check the **Remember my decision** check box if you wish, as shown in Figure 10-102.

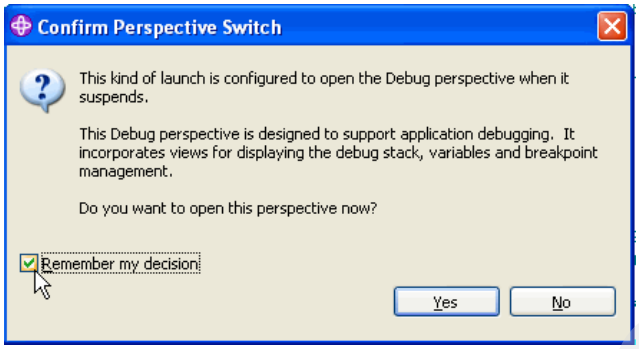


Figure 10-102 Confirm Perspective Switch

g. The Debug perspective appears, as shown in Figure 10-103 on page 317.

Attention: Because of the pseudo-conversational behavior of the CICS transaction, you will switch several times between the Debug view and the Terminal view.

Archived

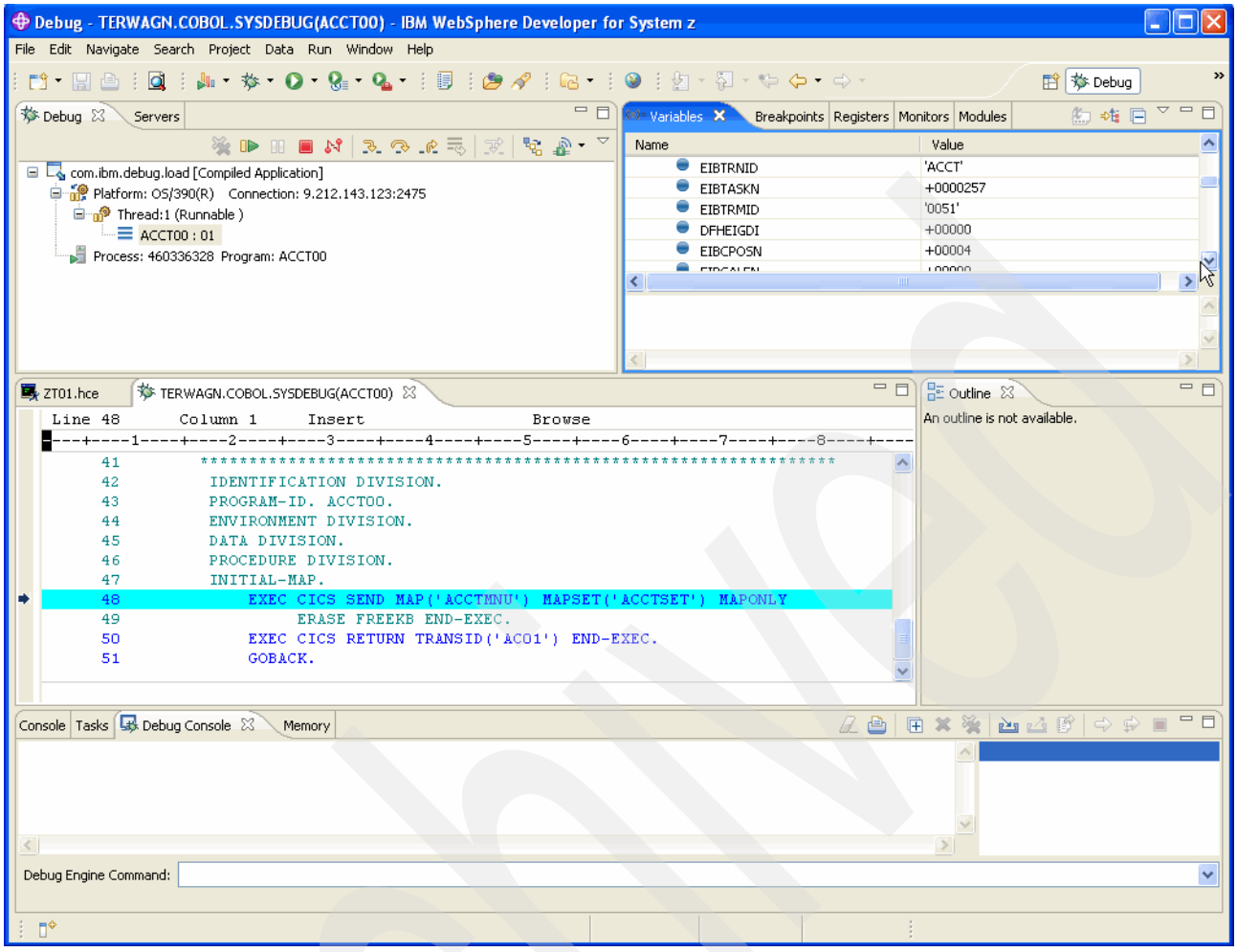


Figure 10-103 Debug view

Note: For more information about using the Debug view, refer to 10.3, “Developing and debugging a batch program” on page 276.

h. You are now ready to develop and debug CICS transaction using WD/z.

10.5 Developing and debugging an IMS application

> **Note:** The sample IMS application used in this section is delivered with IMS. This sample program may already be installed on your system. If you need it, we provide it in the Web material associated with this book; refer to Appendix B, “Additional material” on page 645 for more information.

This section describes how to develop, compile, and debug a MPP IMS programs using WD/z by following these steps:

1. In the z/OS Projects perspective, right-click in the z/OS Projects pane, click **New**, and select **z/OS Project...**, as shown in Figure 10-104.

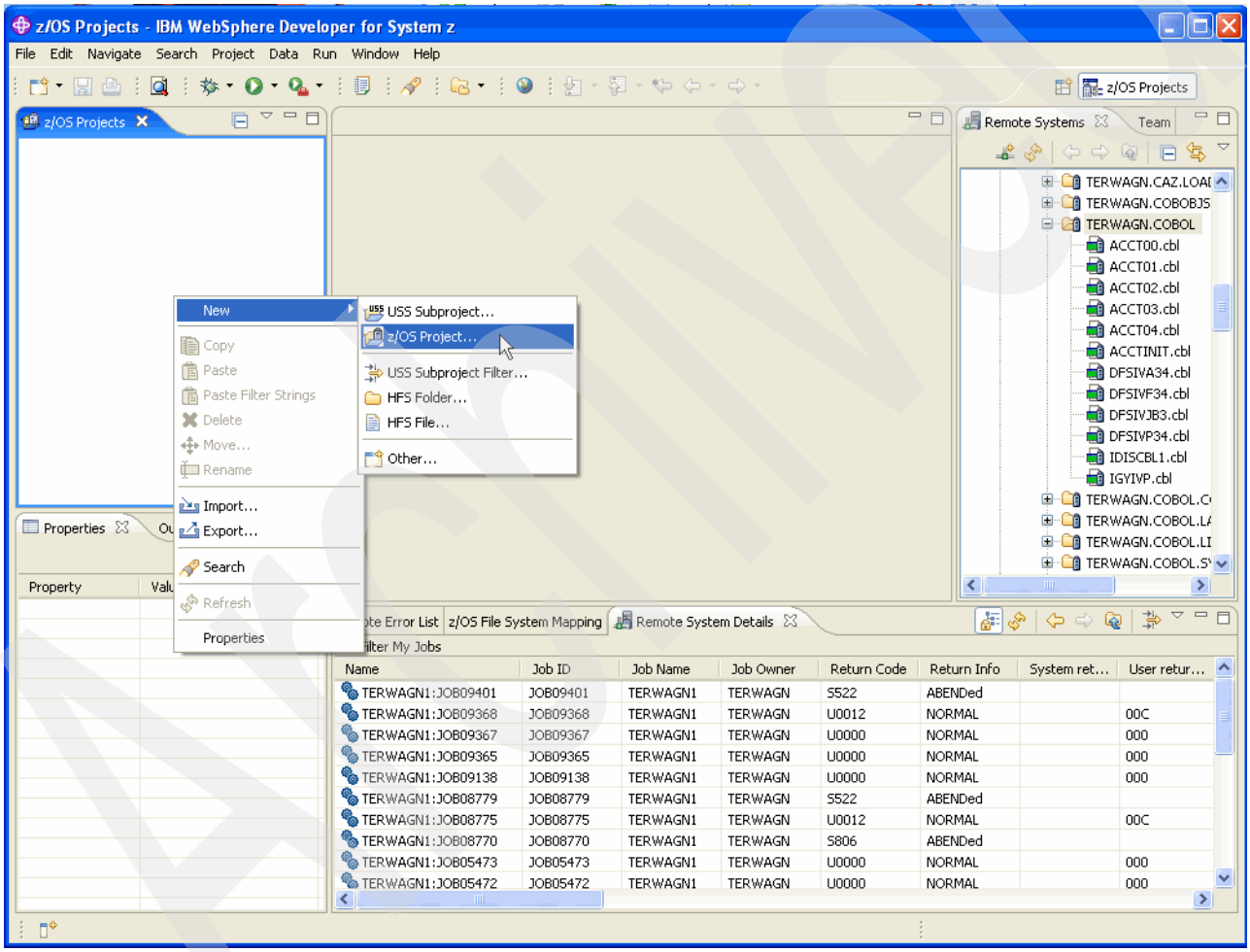


Figure 10-104 IMS application: New z/OS Project

2. Type in a project name of your choice and check **Create an MVS subproject**, as shown in Figure 10-105 on page 319.

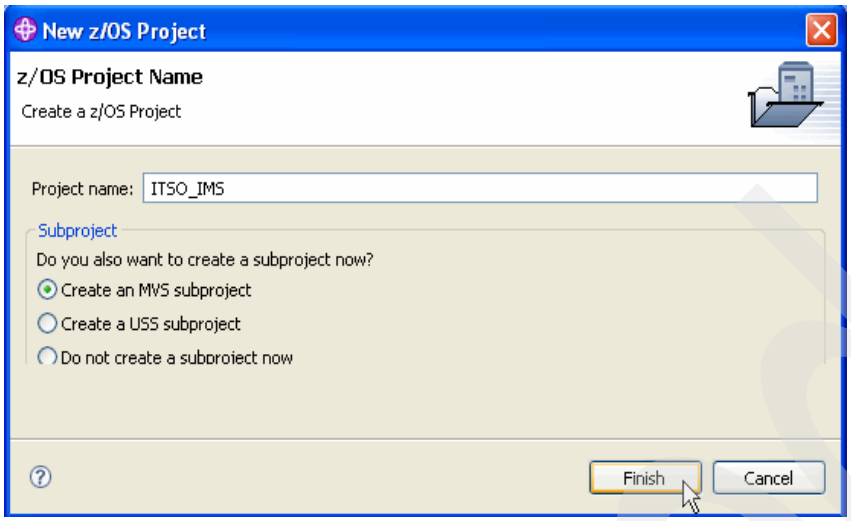


Figure 10-105 IMS application: Create a z/OS Project

3. Enter the name of the subproject. Click **Finish**, as shown in Figure 10-106.

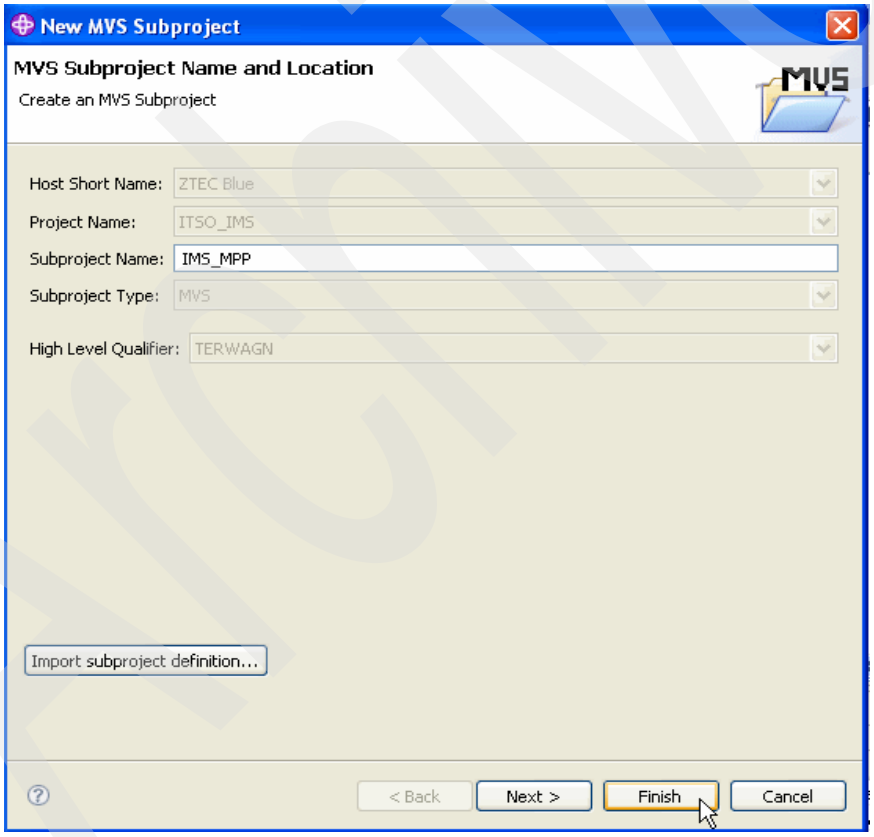


Figure 10-106 IMS application: Create an MVS subproject

4. Right-click the subproject and select **Properties**, as shown in Figure 10-107.

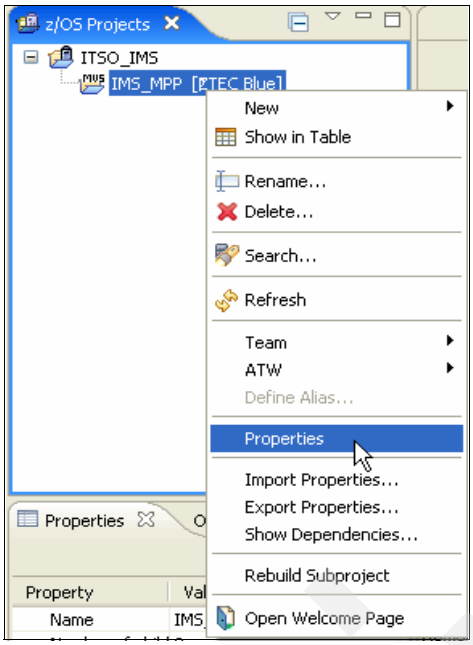


Figure 10-107 Subproject properties

5. Normally, if you set up the properties associated with the MVS Files under your connection correctly, you should get the correct parameters, except ones specific to the IMS application. Select the **IMS** Tab and enter the RESLIB library specific to your installation. Click **OK**, as shown in Figure 10-108.

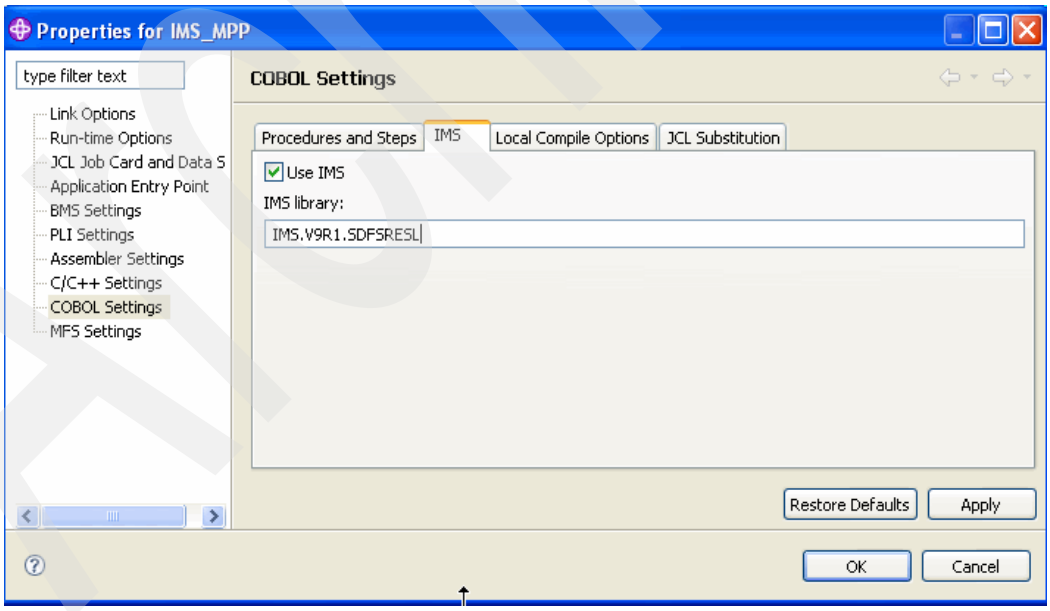


Figure 10-108 COBOL setting for an IMS program

Archived

6. Select **Link Options** and click **Edit Step** of the procedure ELAXFLNK, as shown in Figure 10-109.

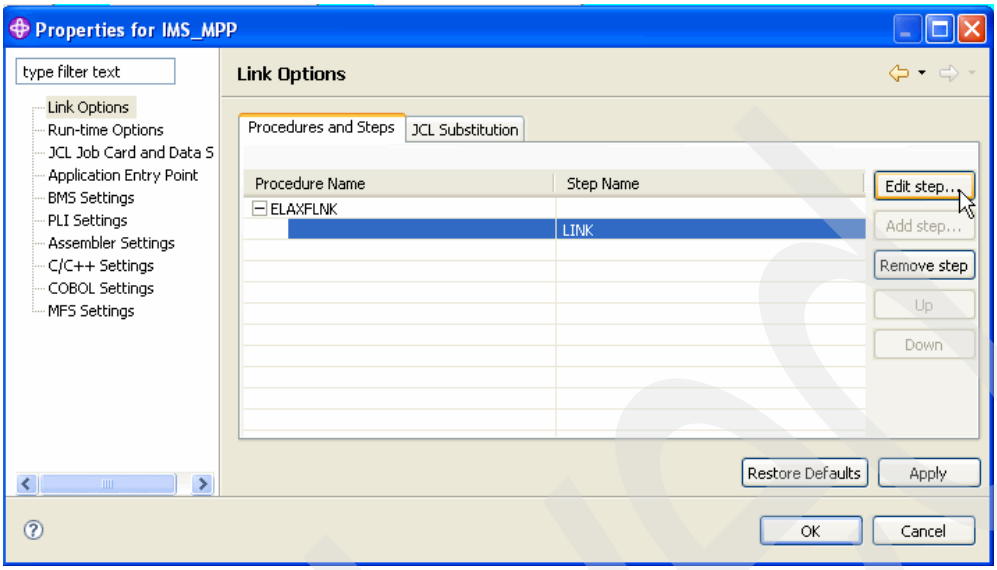


Figure 10-109 Link Options for an IMS program

Archive

7. Verify the Load Module Location, which is the loadlib of your IMS MPP region, as shown in Figure 10-110.

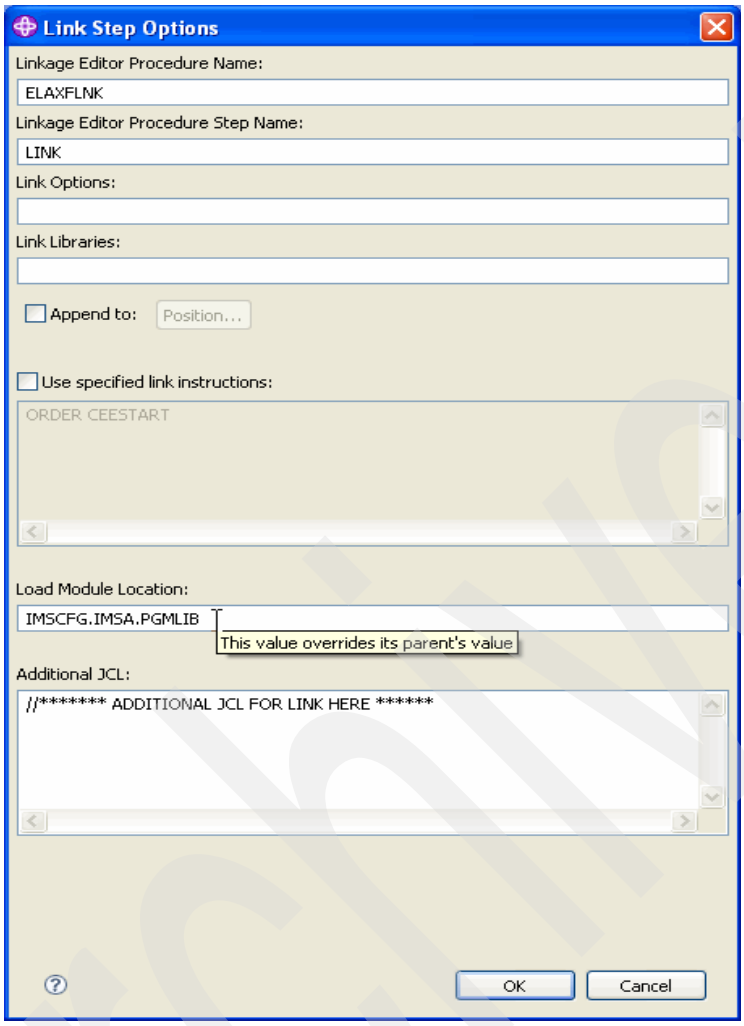


Figure 10-110 Link Step Options for an IMS program

8. Now, we add the program to the subproject. Open the Connection view for the MVS Files and PDS where the IMS Cobol sources are located. Select the COBOL source (**DFSIVA34**), right-click it, and select **Add selected resources to subproject**, as shown in Figure 10-111 on page 323.

Archived

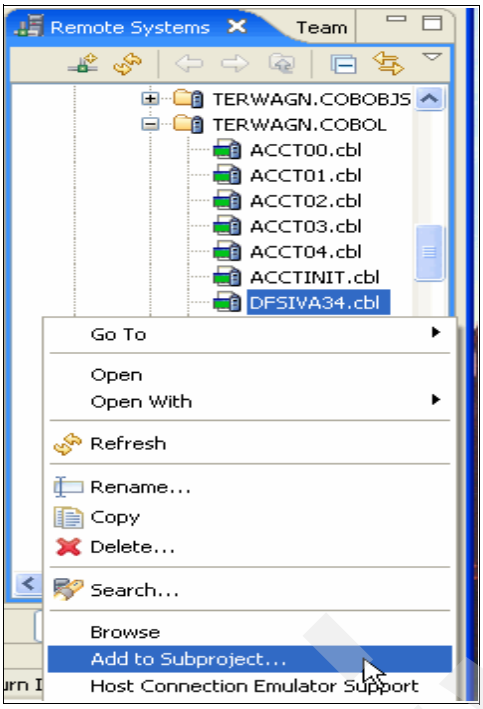


Figure 10-111 Adding the IMS COBOL source to the subproject

9. Select the project and subproject and click **Finish**, as shown in Figure 10-112.

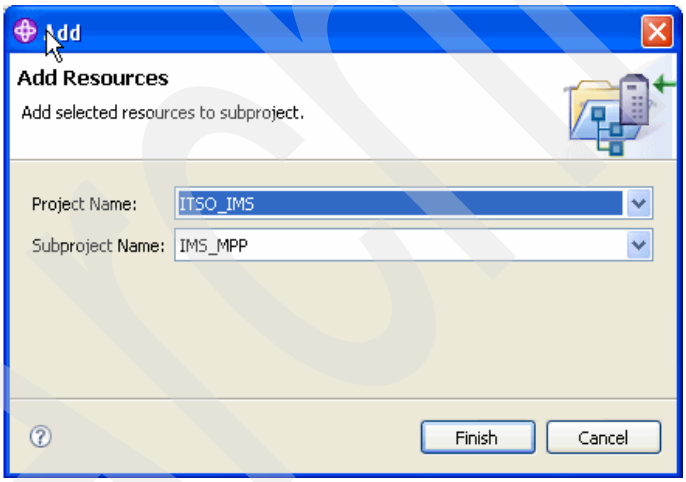


Figure 10-112 Add resource to subproject

Archived

10. Add the MFS source to the subproject (DFSIVF34), and then right-click the subproject and select **Properties**, as shown in Figure 10-113.

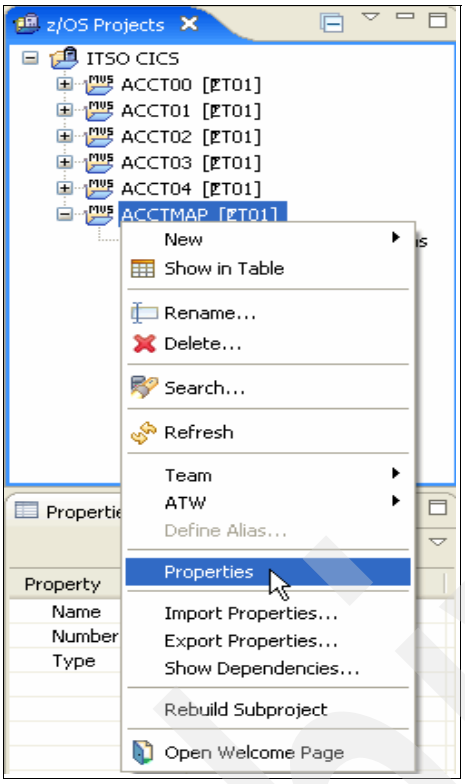


Figure 10-113 Properties of MVS subproject

11. Select **BMS Settings**, expand the procedure **ELAXFBMS**, select the step **ASMMAP**, and click the **Edit Step** button, as shown in Figure 10-114.

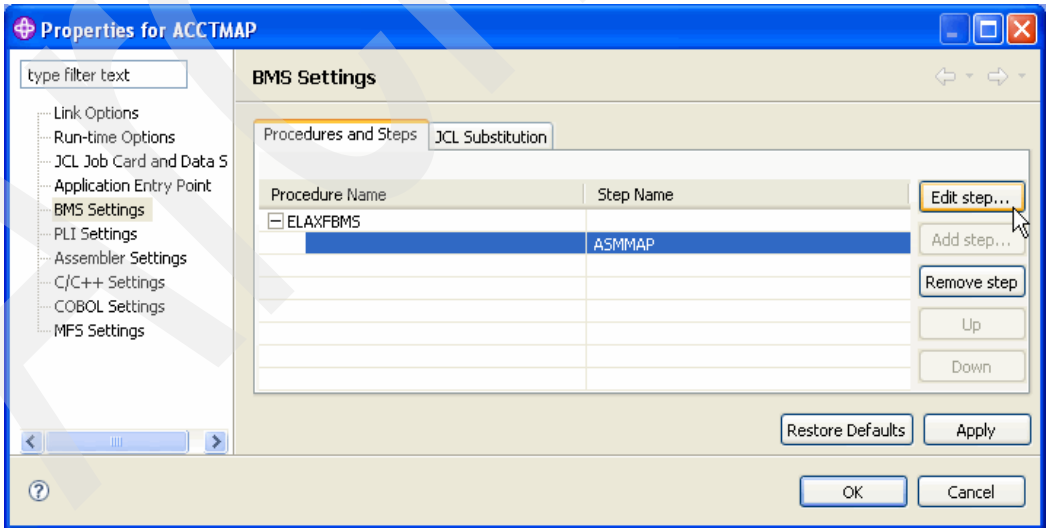


Figure 10-114 Edit ASMMAP step

12. Verify that the BMS Map Object Library (type in your object library) and the BMS DSCTLIB Library are not empty; if they are, correct the problem. Click **OK**.

13. Generate a JCL to Run MFS gen. Right-click the MAP source in the subproject and select **Generate JCL** → **To Run MFS gen**, as shown in Figure 10-115.

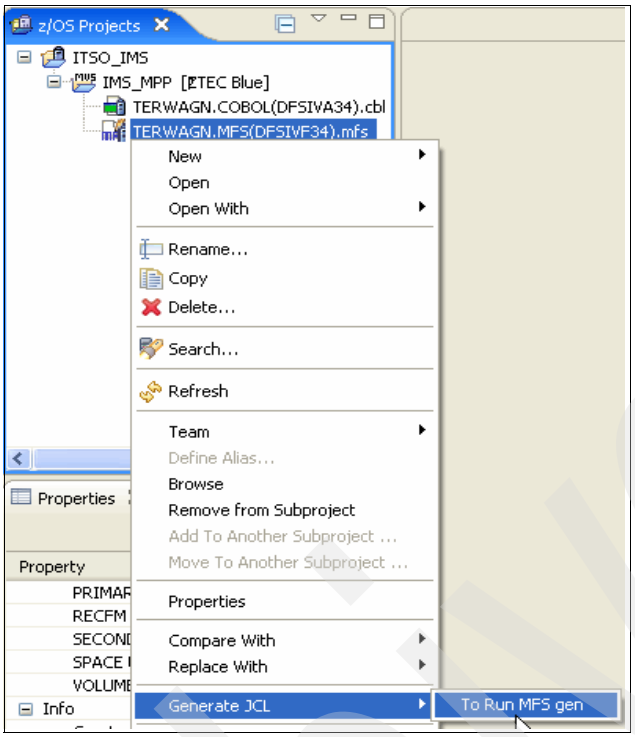


Figure 10-115 Generate JCL to Run MFS gen

14. A pop-up window asks you the job name, the PDS name to store the JCL, and the JCL name. Use the values suggested by the wizard. Click **OK**. When you see the JCL Generation Message, click **OK**. The JCL is added to your subproject, as shown in Figure 10-116 through Figure 10-118 on page 326.

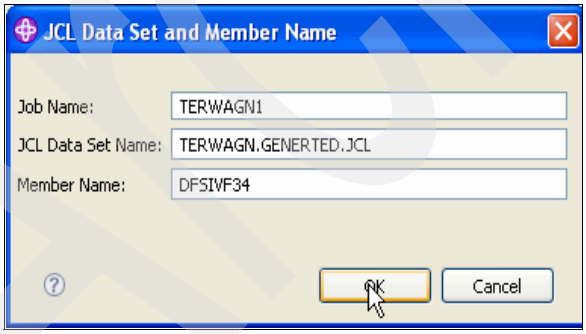


Figure 10-116 JCL generation pop-up window

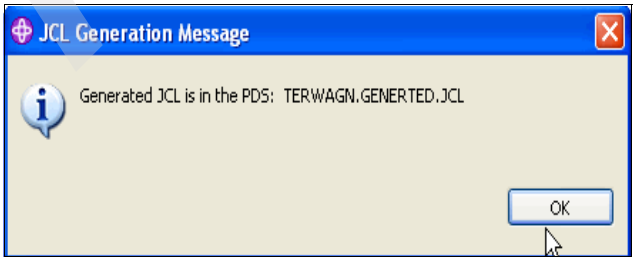


Figure 10-117 JCL Generation Message

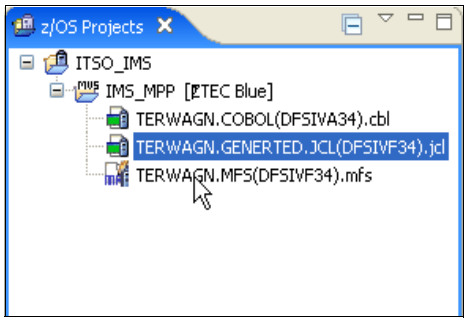


Figure 10-118 JCL added in subproject

15. Right-click the JCL and select **Submit**. The Job Submission Confirmation pop-up window appears, as shown in Figure 10-119 and Figure 10-120.

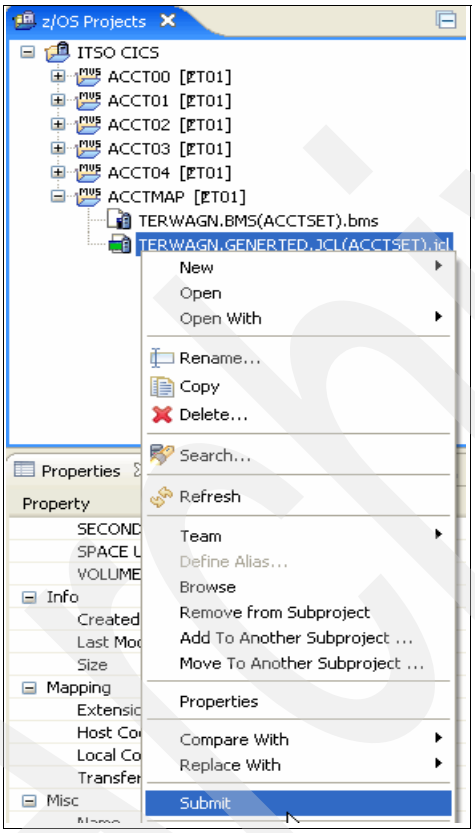


Figure 10-119 Submit JCL

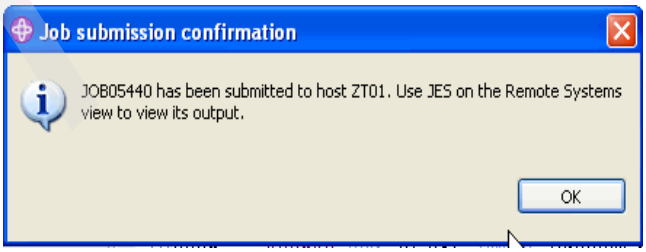


Figure 10-120 Job submission confirmation pop-up window

Archived

16. Open the Remote Systems view and select **My Jobs**. Do not forget to refresh the job list by right-clicking **My jobs** and selecting **Refresh**. You should see the submitted job and its sysouts. Double-click the first one (JES2:JESMSGLG) to see the result, as shown in Figure 10-121 and Figure 10-122.

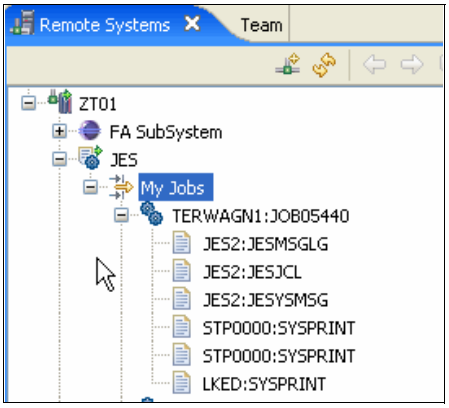


Figure 10-121 Job monitoring My Jobs

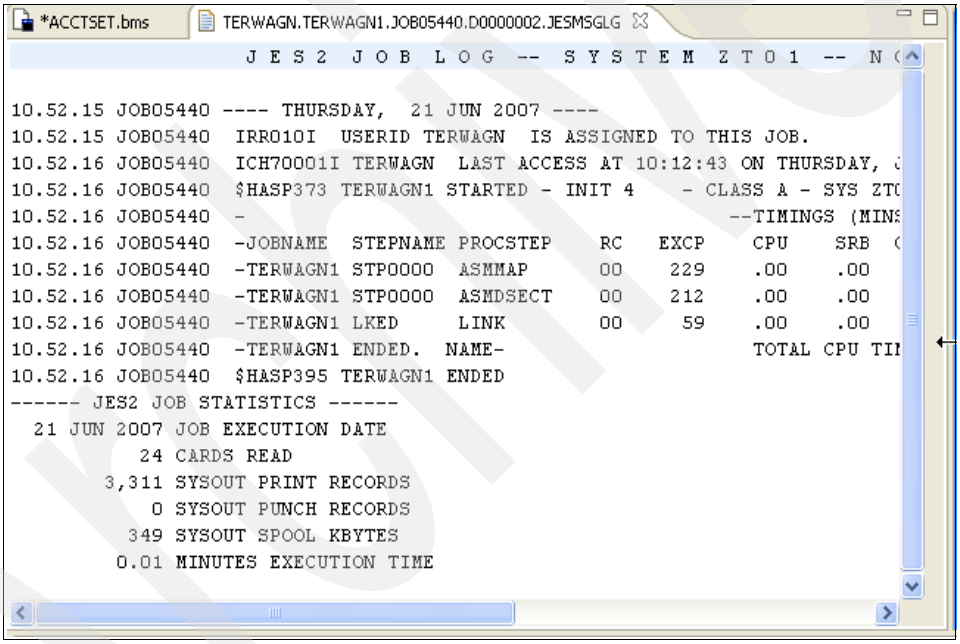


Figure 10-122 JES messages

Tip: You can also right-click **My jobs** and select **Show in table**. The job list appears in the Remote Systems Details view. This view shows more information about the jobs (especially the Return Code).

17. Open the Project view, right-click the IMS project, and select **Rebuild Project**. The application automatically generates the JCL to compile and link the program. It submits it to the mainframe (JES) and parses the errors from the compiler.

Archived

18. You should see several new members from different PDSs in the subproject, such as objects, load modules, listings, and so on, as shown in Figure 10-123.

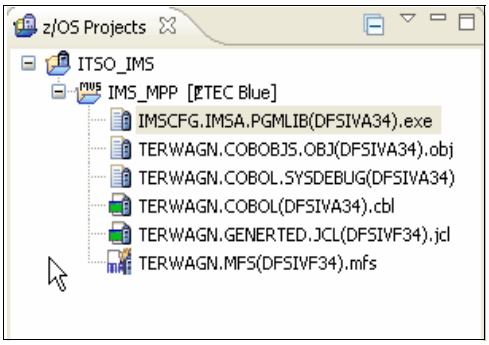


Figure 10-123 IMS project build

19. We are now ready to debug the IMS transaction. We need an IMS terminal to start the transaction. But we also need to log into TSO to set up the Language Environment debug option. In the Remote Systems view, right-click your system and select **Host Connection Emulator Support**, as shown in Figure 10-124.

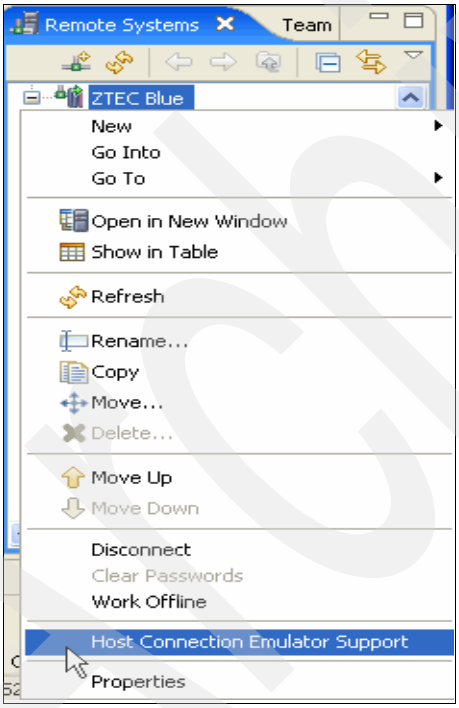


Figure 10-124 Starting a Host Connection Emulator

Archived

20. A terminal view opens. Log into TSO and open the ISPF Debug menu. Enter option 4, as shown in Figure 10-125.

> **Note:** If needed, you can change the properties of the terminal emulation by clicking the tab (see the bottom of the current view).

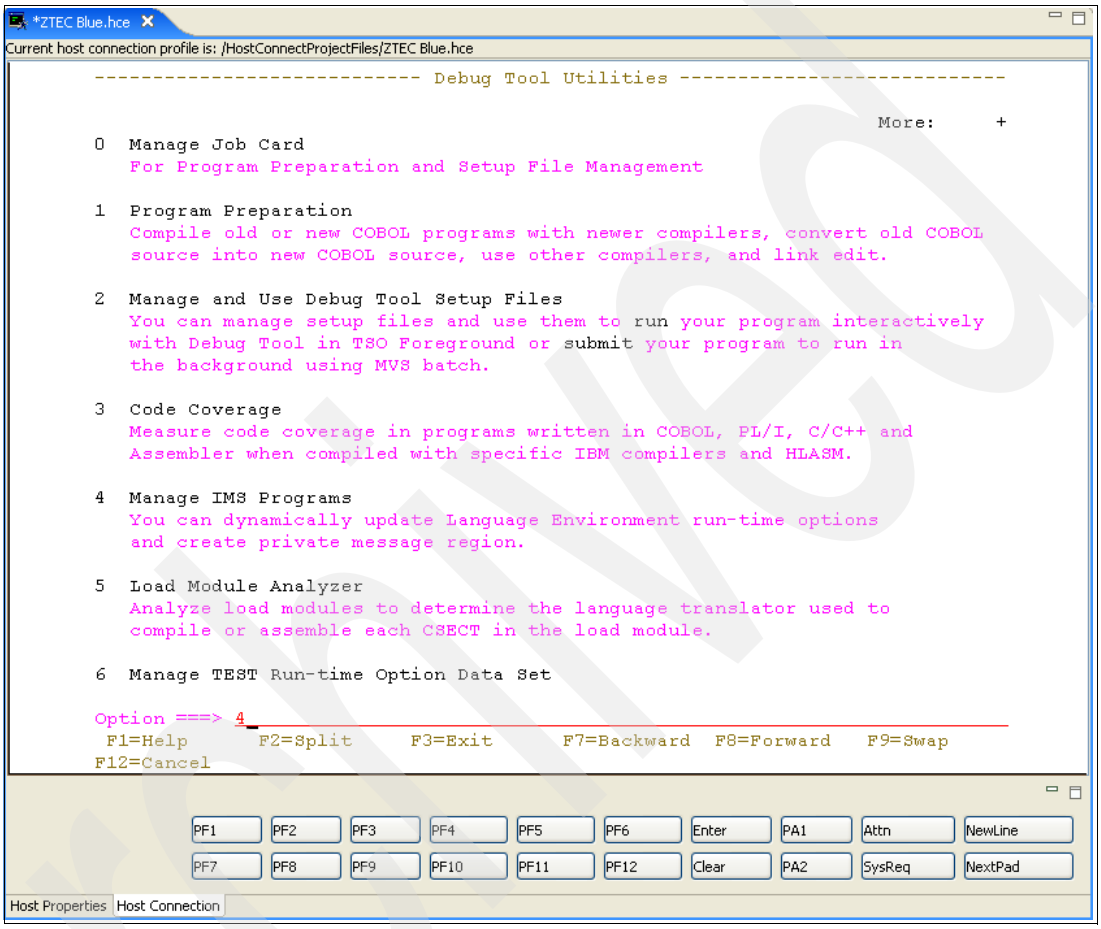


Figure 10-125 Debug Tool Utilities

Archived

21. When you see the “Manage IMS Programs” window, enter 1 to go to the LE runtime Options, as shown in Figure 10-126.

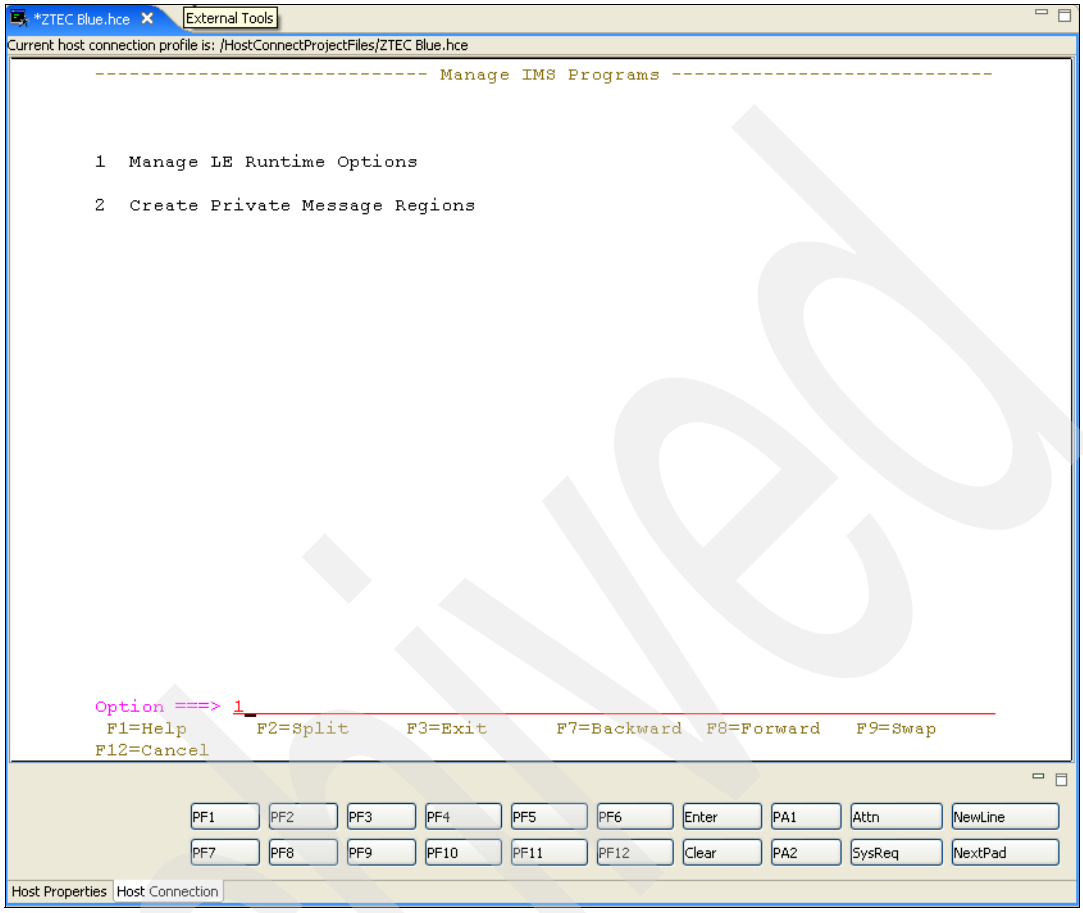


Figure 10-126 Manage IMS Programs

Archived

22. We must enter the IMSPlex ID; the other parameters are optional. Press Enter, as shown in Figure 10-127.

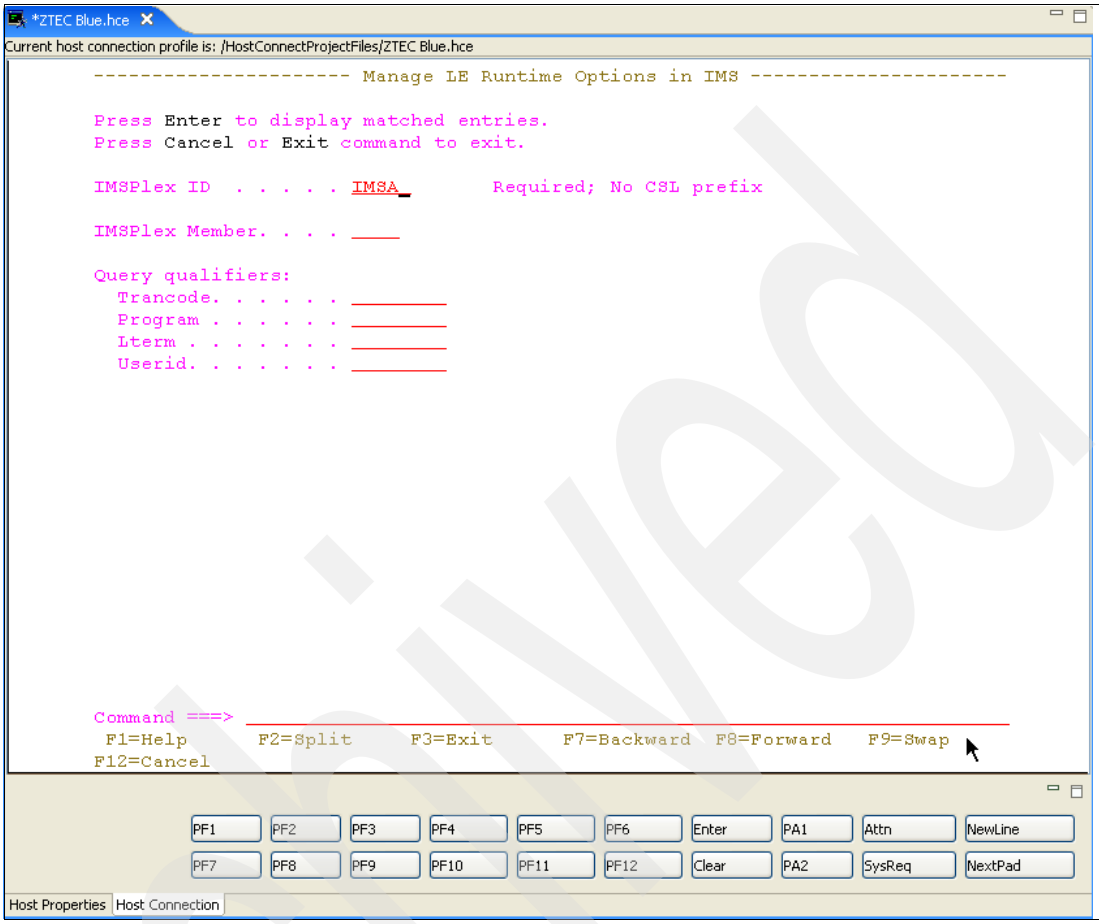


Figure 10-127 Manage Language Environment runtime Options in IMS

Archived

23. If you do not have any entries, you should see an empty list. Press F4 to add a new entry, as shown in Figure 10-128.

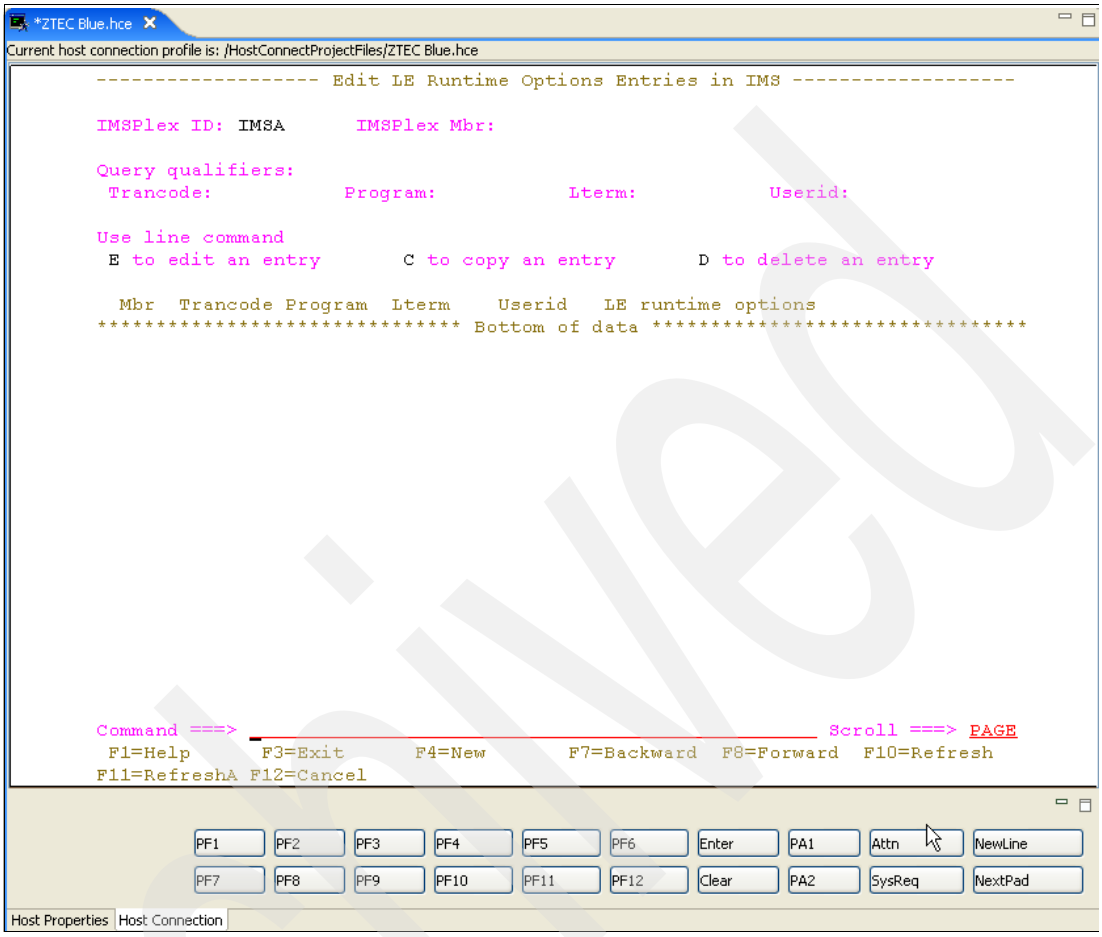


Figure 10-128 Edit Language Environment Runtimes Options Entries in IMS

24. At this point, you must find the IP address of your workstation and the Debug Daemon listening port. If do not know these values, open the Debug view click the **Debug Daemon** button. It should be green; if it red, click it to start the daemon. Note at the first line of the drop-down list if the daemon is listening and at which port. Select **Get Workstation IP...**, and a pop-up window appears, as shown in Figure 10-129 and Figure 10-130 on page 333.

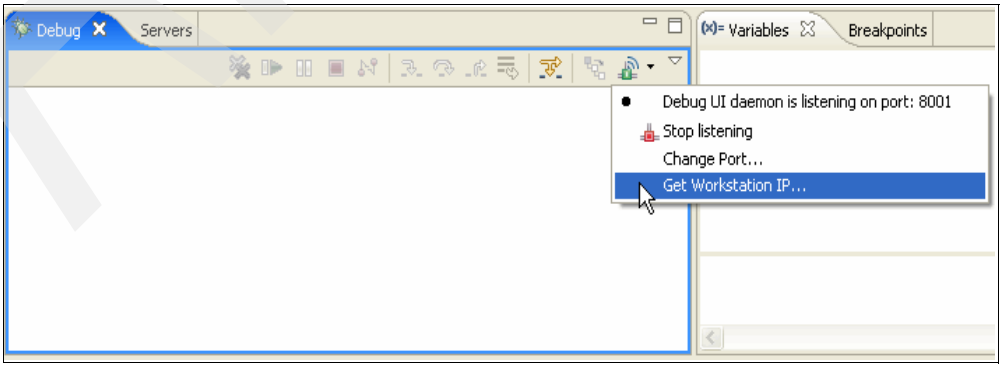


Figure 10-129 Debug perspective: Debug Daemon button

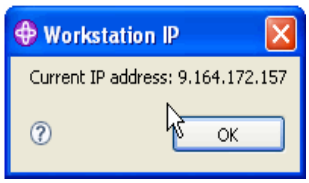


Figure 10-130 Workstation IP

25. Enter the Trancode (IVTCB) and the right IP address and port. Press Enter and press F3, as shown in Figure 10-131.

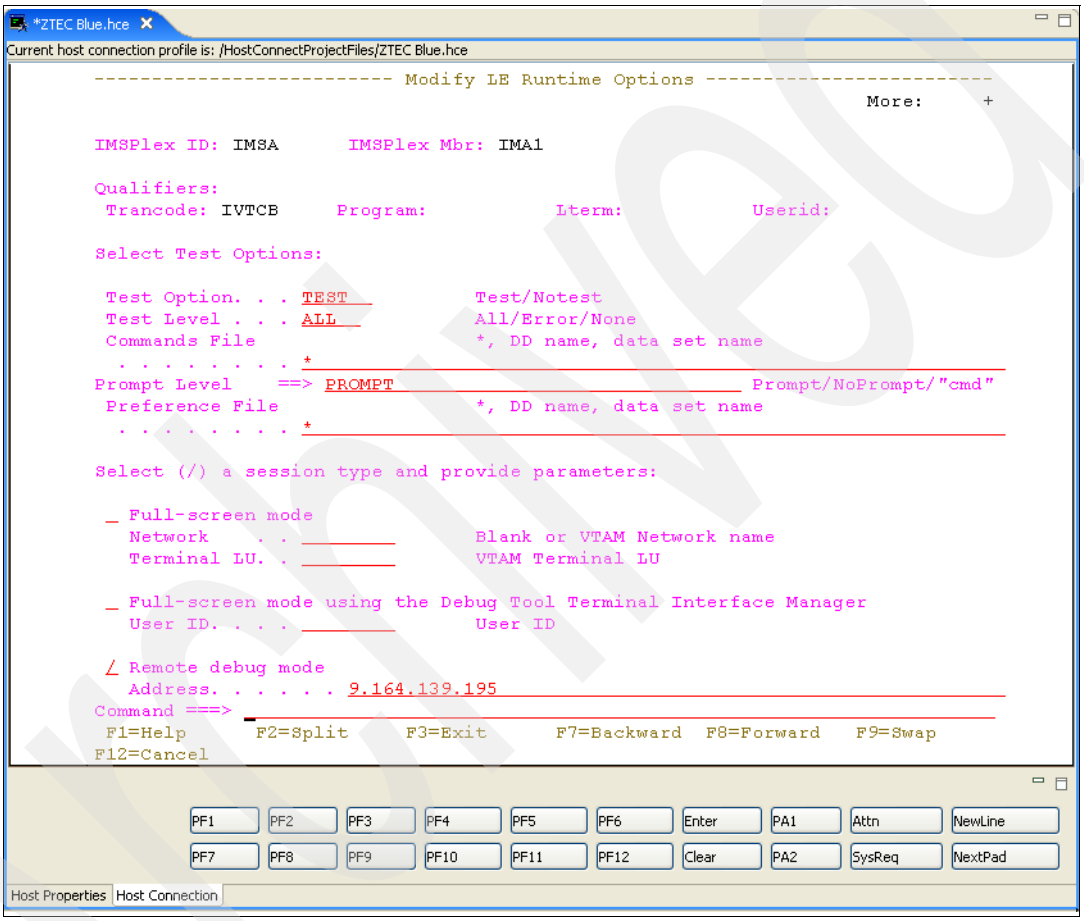


Figure 10-131 Modify Language Environment Runtime Options

Archived

26. You should see your new entry in the list. You can exit the TSO session if you wish, as shown in Figure 10-132.

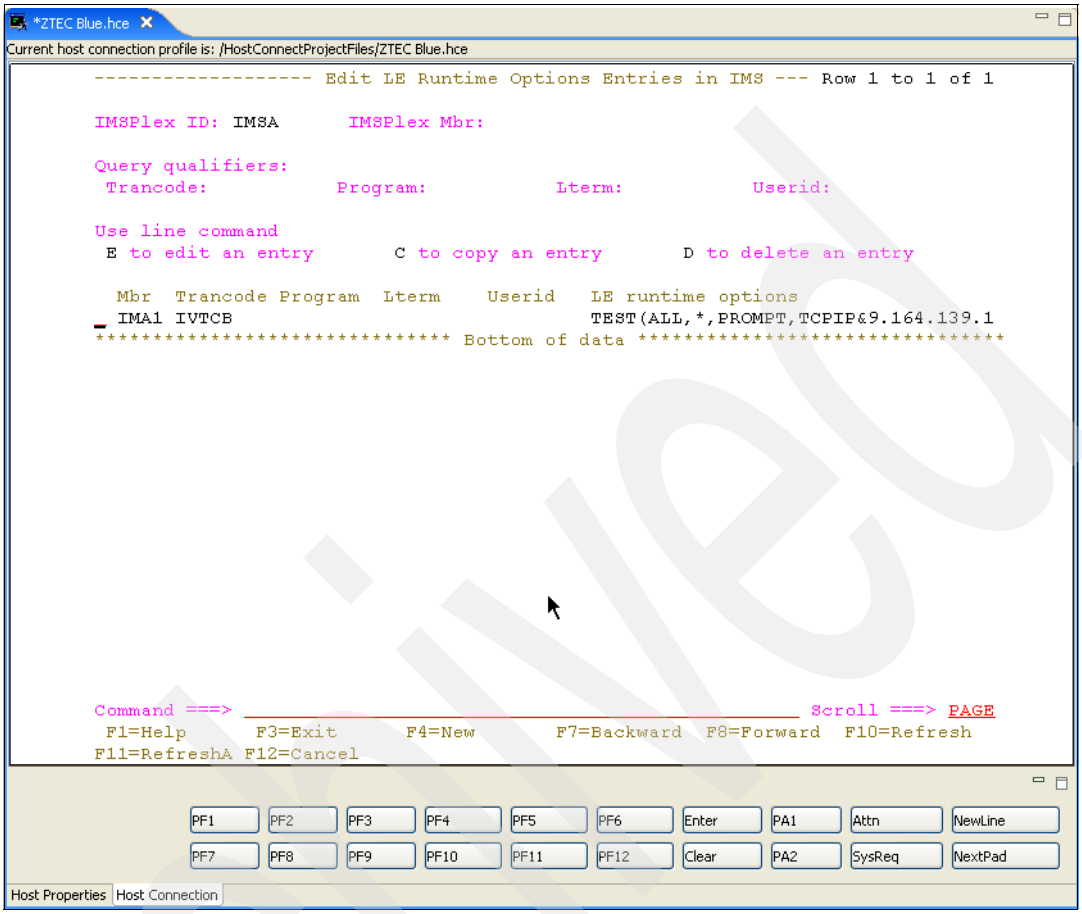


Figure 10-132 Edit Language Environment Runtime Options Entries in IMS

Archived

27. Open an IMS terminal and run the **/for IVTCB** command to call a transaction and press Enter. Enter a Process Code (possible values are listed) and press Enter, as shown in Figure 10-133 and Figure 10-134 on page 336.

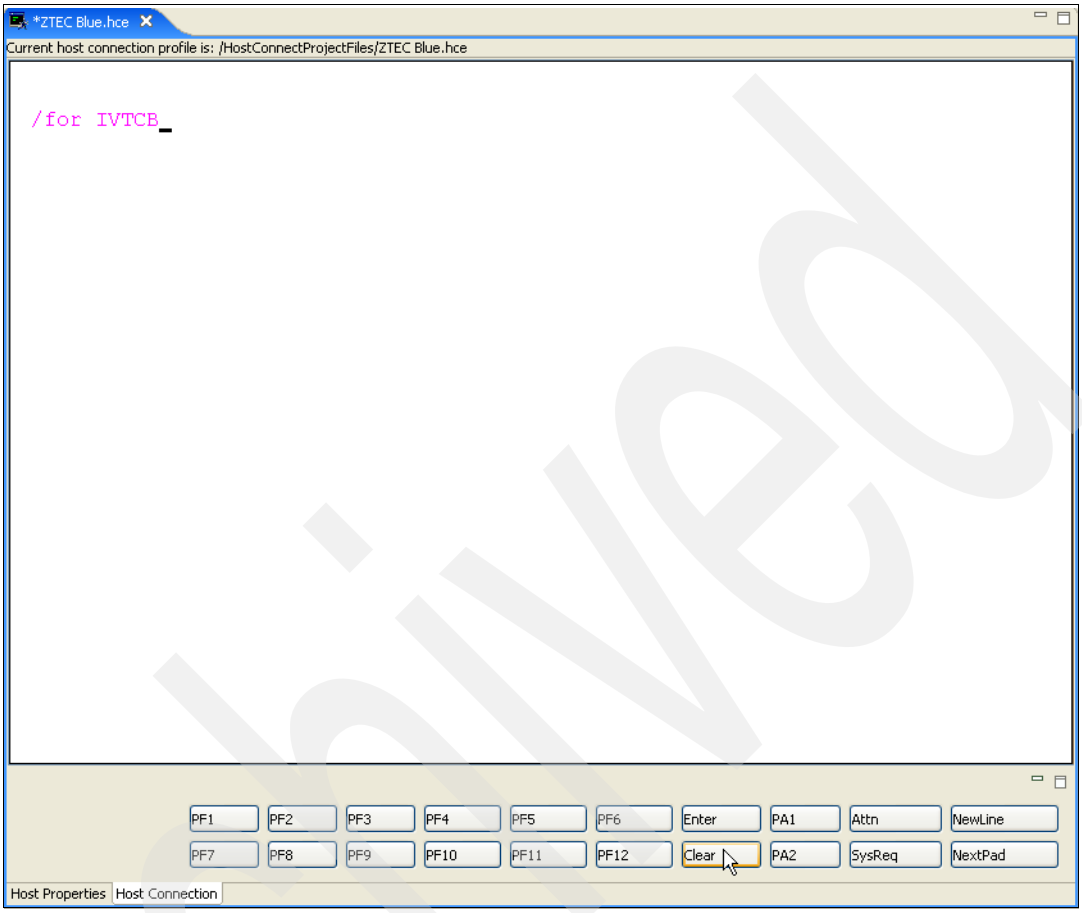


Figure 10-133 IMS terminal

Archived

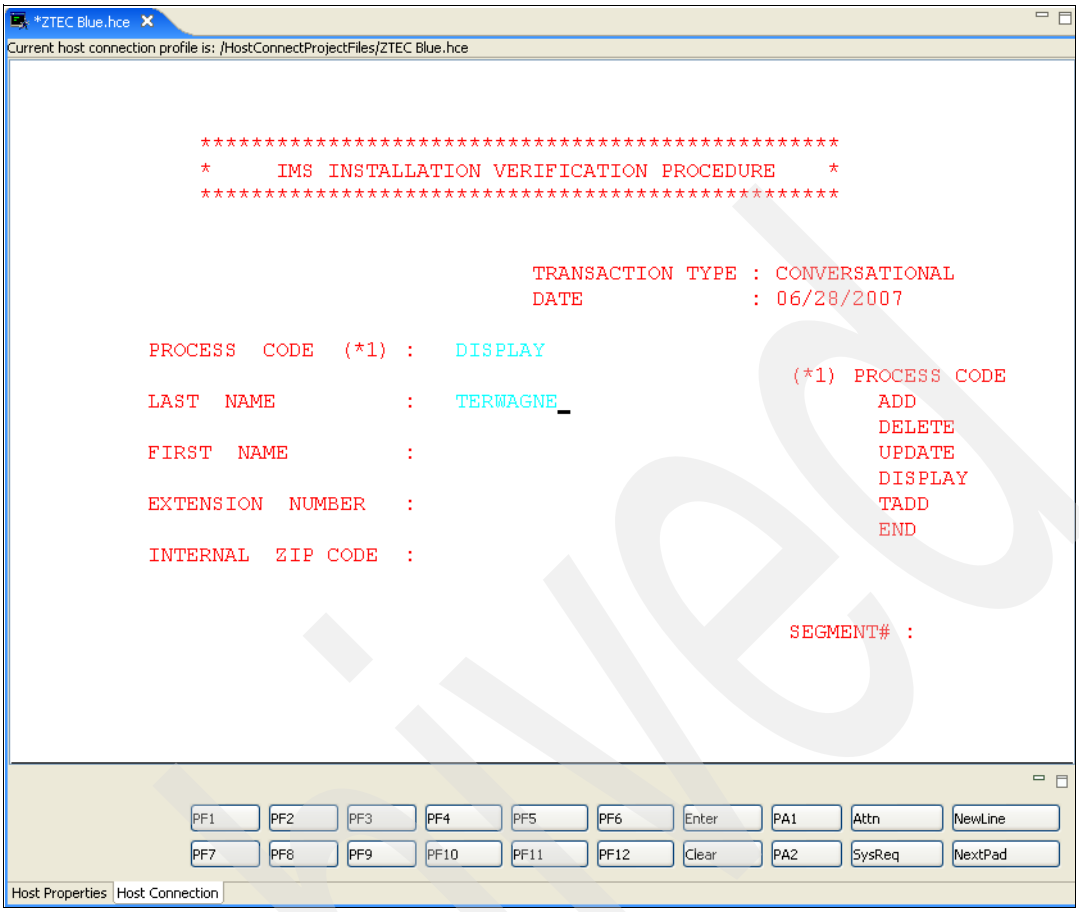


Figure 10-134 IMS transaction

28. When WD/z is contacted by the Debug Tool, it switches to the Debug perspective. A pop-up window could appear asking if you wish to accept to switch. Click **Yes**. You can check the **Remember my decision** check box if you wish, as shown in Figure 10-135.

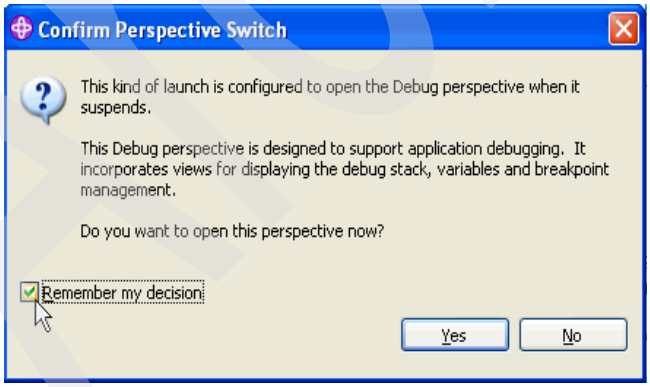


Figure 10-135 Confirm Perspective Switch

29. The Debug perspective appears, as shown in Figure 10-136 on page 337.

Attention: Because of the pseudo-conversational behavior of the CICS transaction, you will switch several times between the Debug view and the Terminal view.

Archived

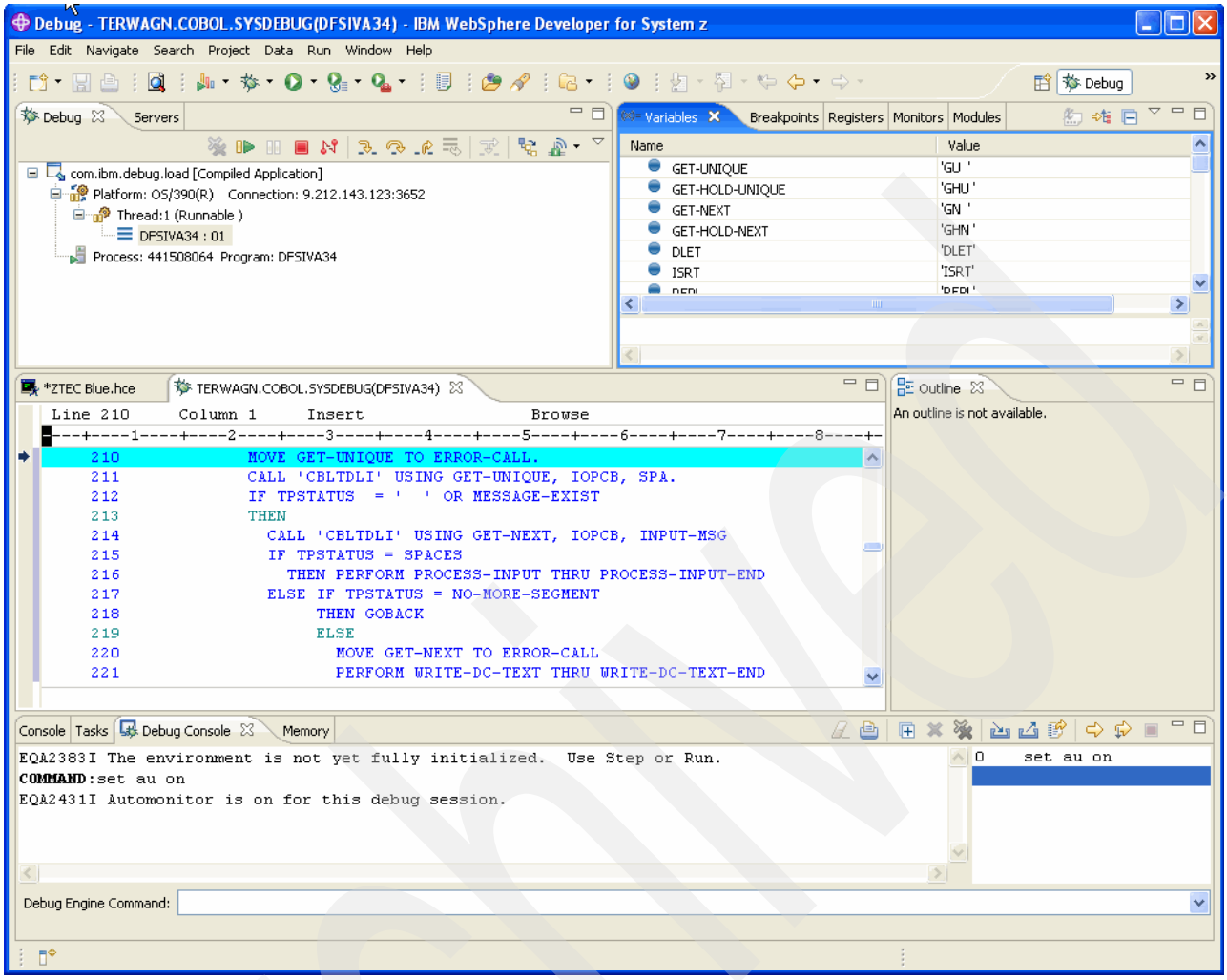


Figure 10-136 Debug view

> **Note:** For more information about using the Debug perspective, refer to 10.3, “Developing and debugging a batch program” on page 276.

30. You are now ready to develop and debug IMS transactions using WD/z.

10.6 DB2 stored procedures builder for PL/I and COBOL

This chapter describes how to create a COBOL stored procedure using the Data perspective of WebSphere Developer for System z. It describes also how to deploy, execute, and debug it. You can follow the same steps to create a PL/I stored procedure:

1. Open the Data perspective. You see two specific views in the left pane: Data Project Explorer and Database Explorer. The first step is to establish a connection to DB2 for z/OS. Right-click **Connections** in the Database Explorer and select **New Connection ...**, as shown in Figure 10-137.

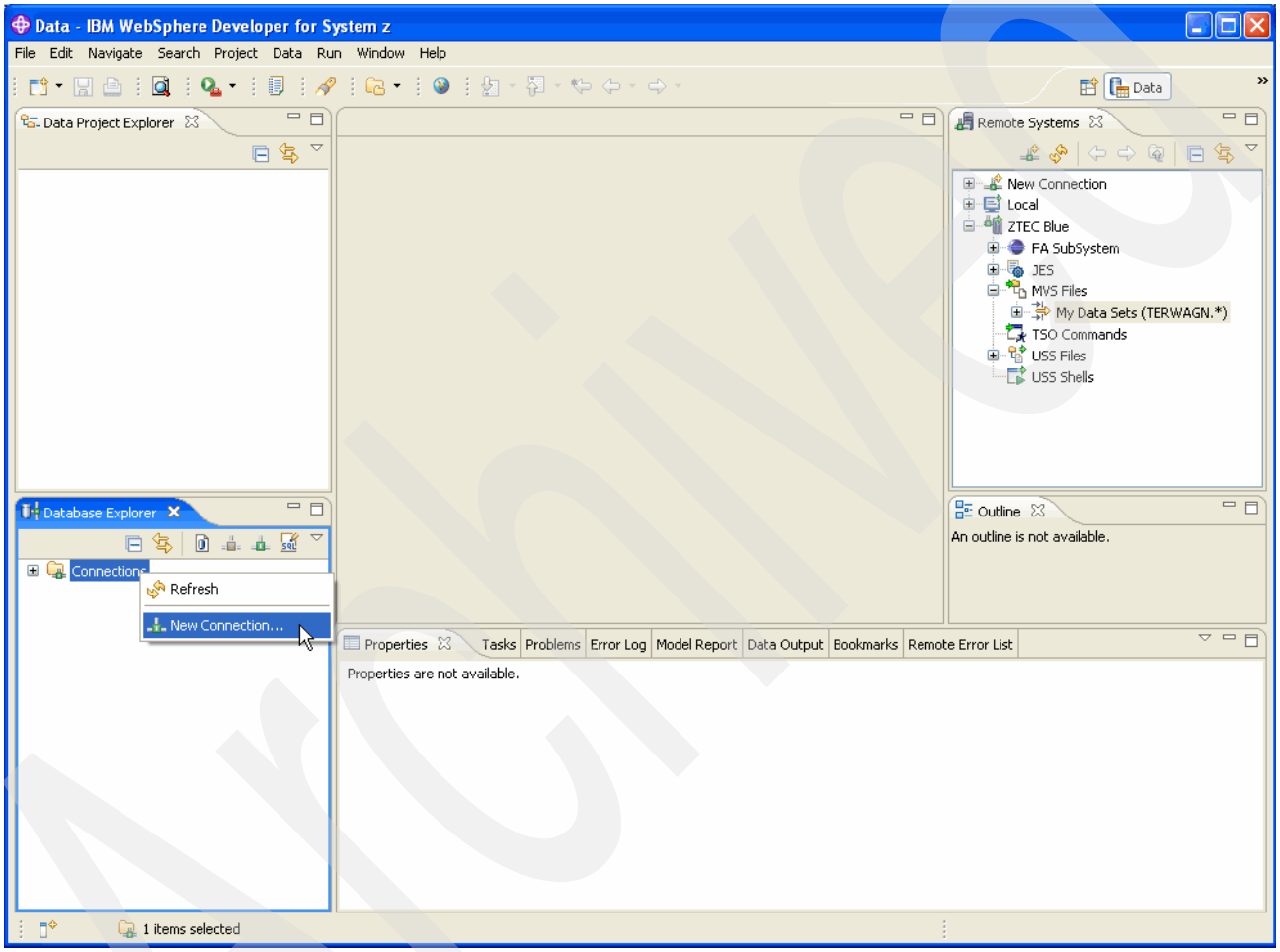


Figure 10-137 Create new database connection

2. There are several ways to connect to DB2 for z/OS. We will use the IBM DB2 Universal JDBC™ driver. Feel free to chose another method depending on your installation or standards. The goal is to be able to access your DB2 objects (tables, views, stored procedures, and so on). You can ask the DB's system engineer what parameters to use. Figure 10-138 on page 339 and Figure 10-139 on page 339 are specific to our installation. When the parameters are entered, you are able to test the connection before saving it. Click the **Test Connection** button.

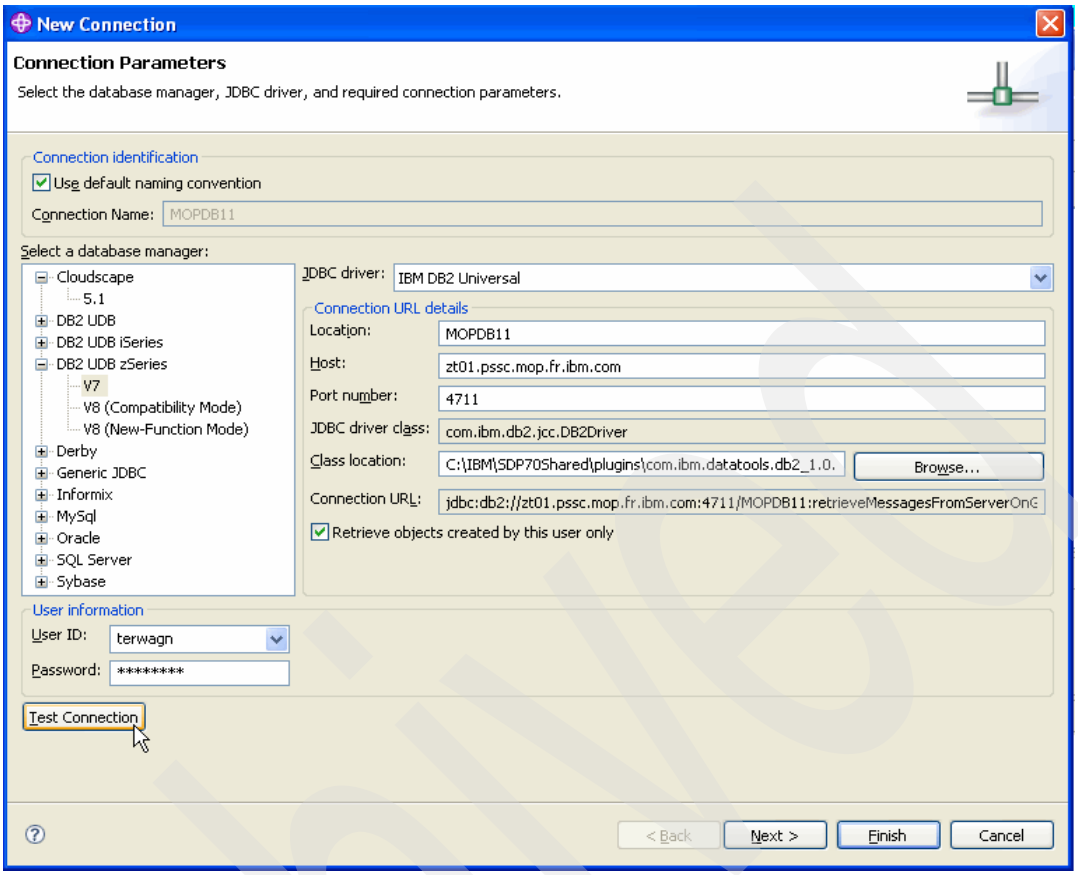


Figure 10-138 Create an DB2 connection

The window shown in Figure 10-139 will appear if the test connection is successful.

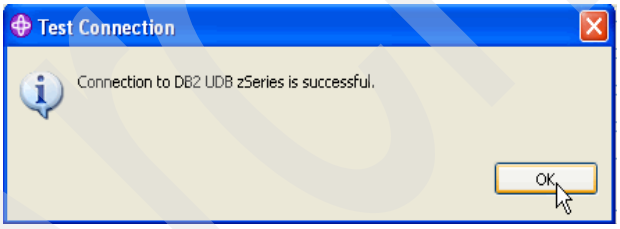


Figure 10-139 Test Connection

3. When the connection is defined, we are ready to create a Database Project. In the Database Explorer view, right-click and select **New** → **Database Development Project**, as shown in Figure 10-140.

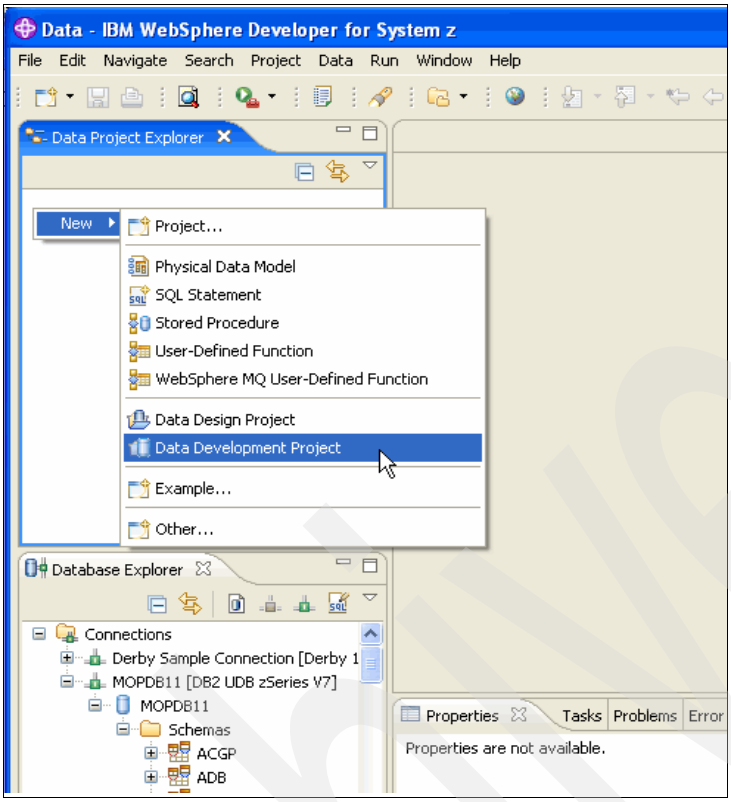


Figure 10-140 Create Database Development Project

4. Enter a name and click **Next**, as shown in Figure 10-141.

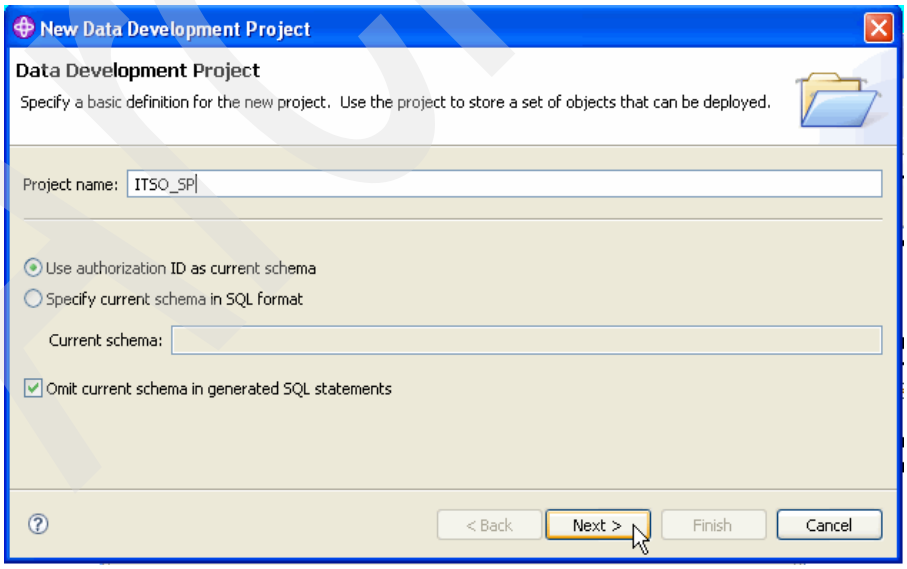


Figure 10-141 New Database Development Project wizard

5. Select the existing connection we created before and click **Finish**, as shown in Figure 10-142 on page 341.

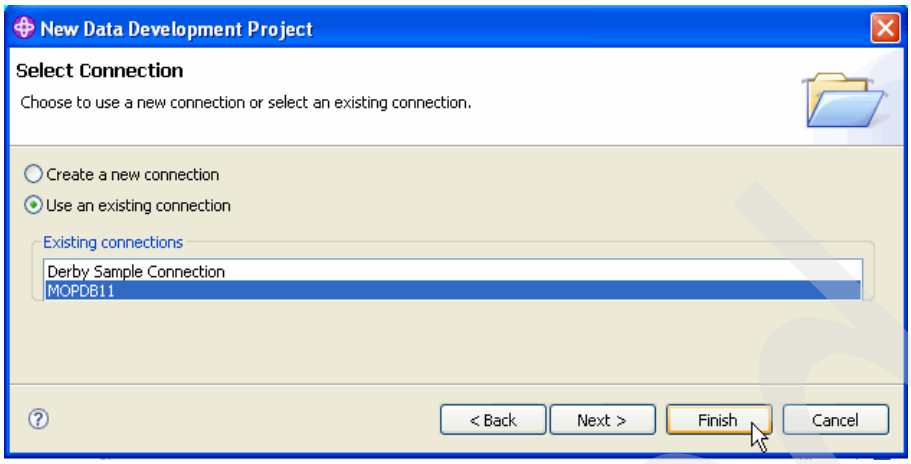


Figure 10-142 Select existing connection

6. When the project is created, you get two folders: SQL Scripts and Store Procedures. Right-click the **Stored procedures** folder and select **New** → **Stored Procedure**, as shown in Figure 10-143.

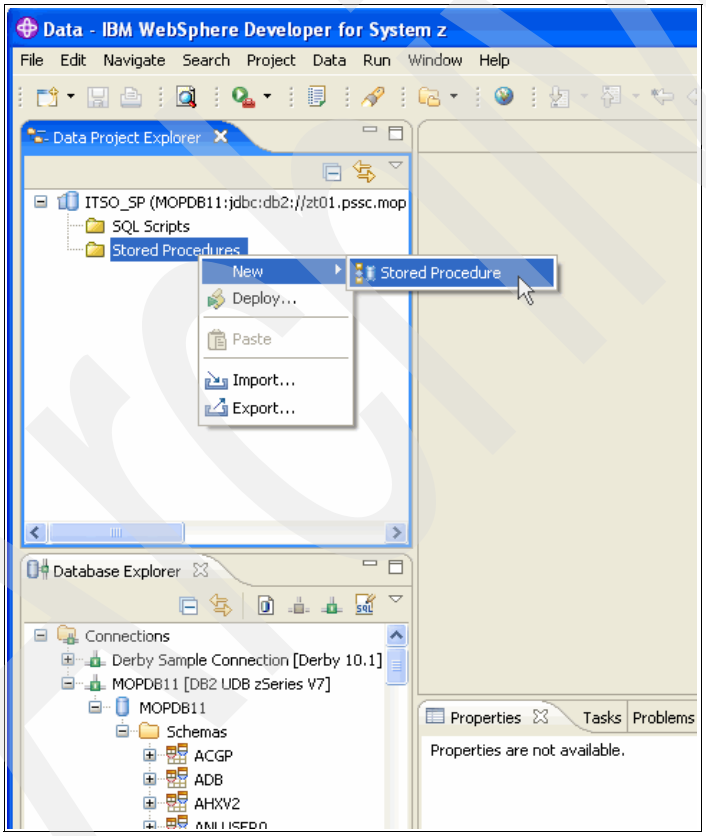


Figure 10-143 Start New Stored Procedure wizard

7. The first step gives you the ability to choose the project or create a new one. Be sure to chose the project we created before. Click **Next**, as shown in Figure 10-144.

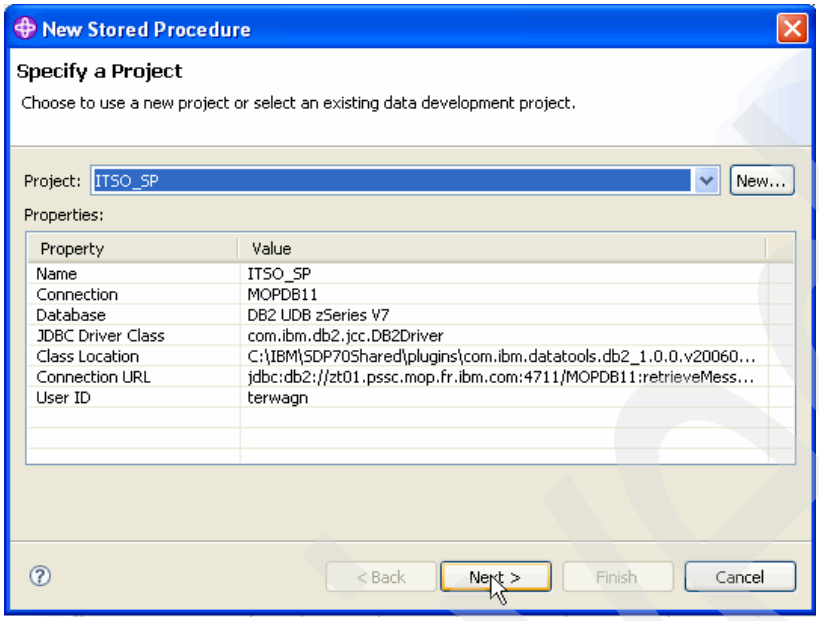


Figure 10-144 Select the project

8. Enter the name of the procedure (this name is not the public one, but is only visible in the Database Development Project). You can also choose the language. Feel free to choose COBOL or PL/I; the next steps are the same regardless of your choice. Click **Next**, as shown in Figure 10-145.

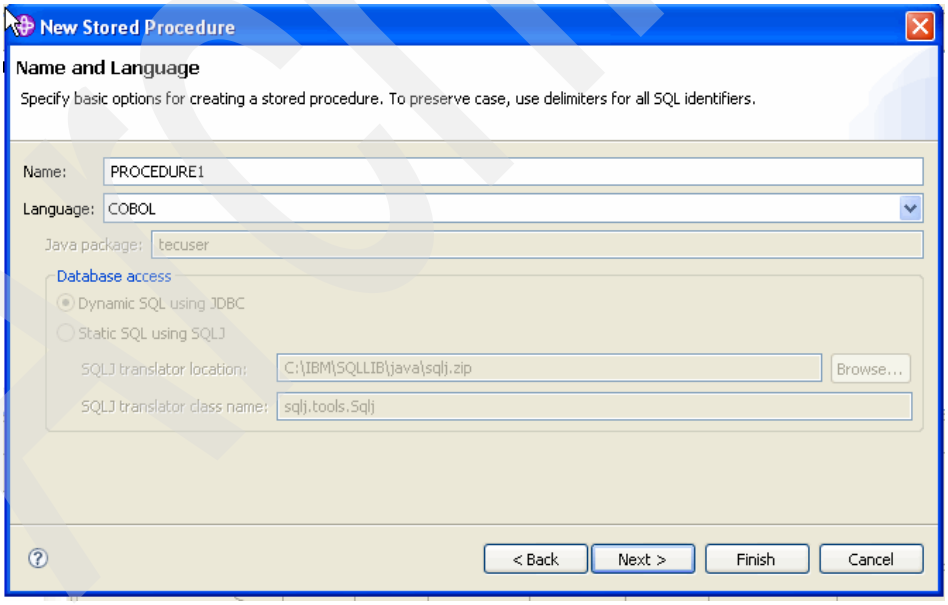


Figure 10-145 Chose the stored procedure language

9. Enter the public name of the stored procedure. Do not use the name proposed by the wizard, but use one that conforms to your naming conventions. Click **Next**, as shown in Figure 10-146 on page 343.

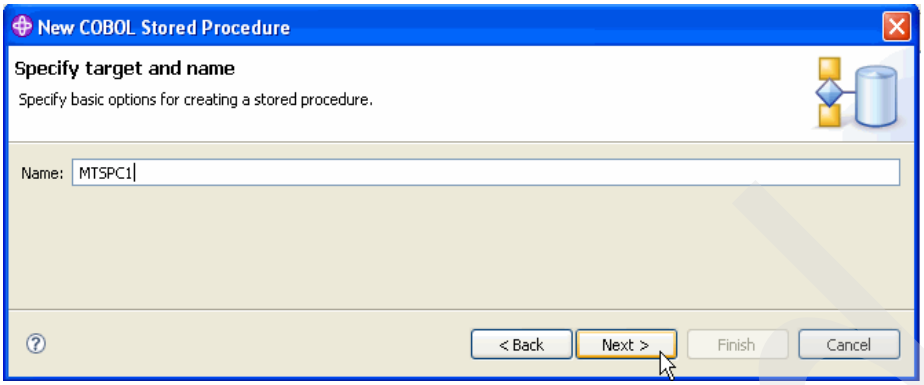


Figure 10-146 Enter the name of the stored procedure

10. The source of the stored procedure is stored in a PDS. Select your connected system and your COBOL or PL/I sources database and click **Next**, as shown in Figure 10-147.

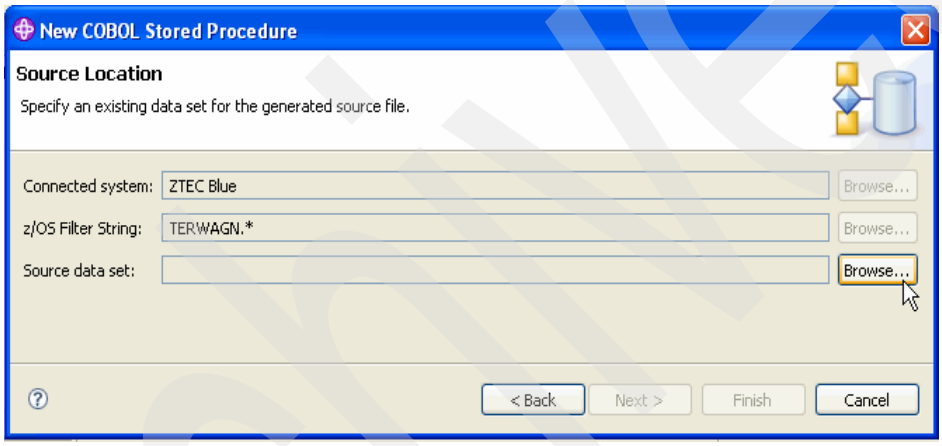


Figure 10-147 Select the source data set

Archived

11. Now we create a SQL statement. There a wizard to help you, but you may write the statement yourself if you wish. Note that you can have several SQL statements in a stored procedure. Click **Create SQL...**, as shown in Figure 10-148.

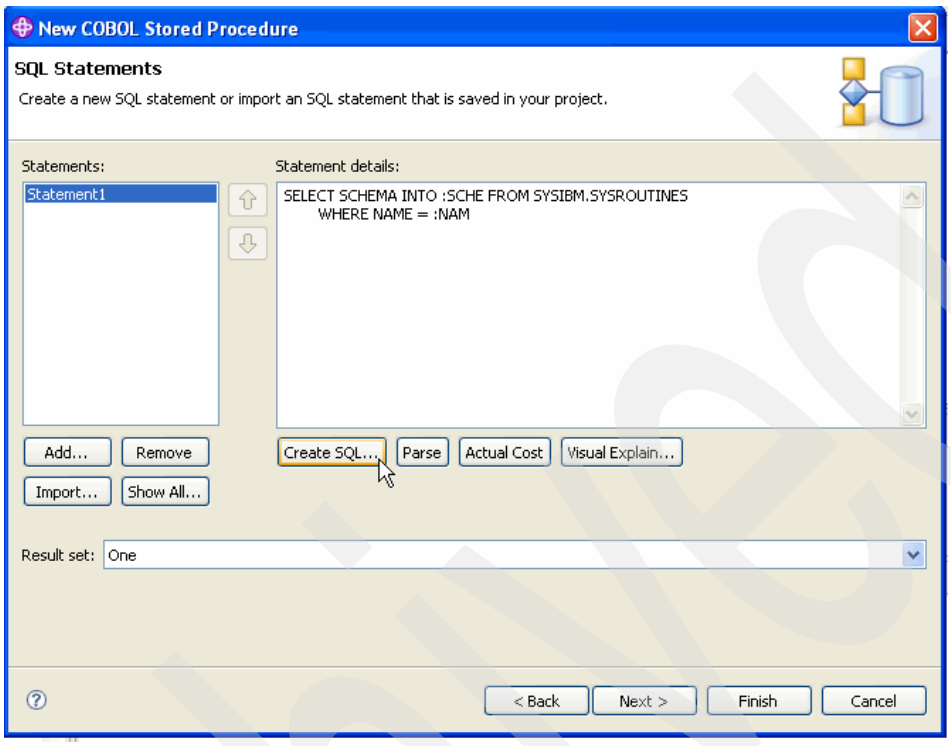


Figure 10-148 Create SQL statement

12. Choose the type of SQL statement, select **INSERT**, and check the **Be guided through creating an SQL statement** check box. Click **Next**, as shown in Figure 10-149.

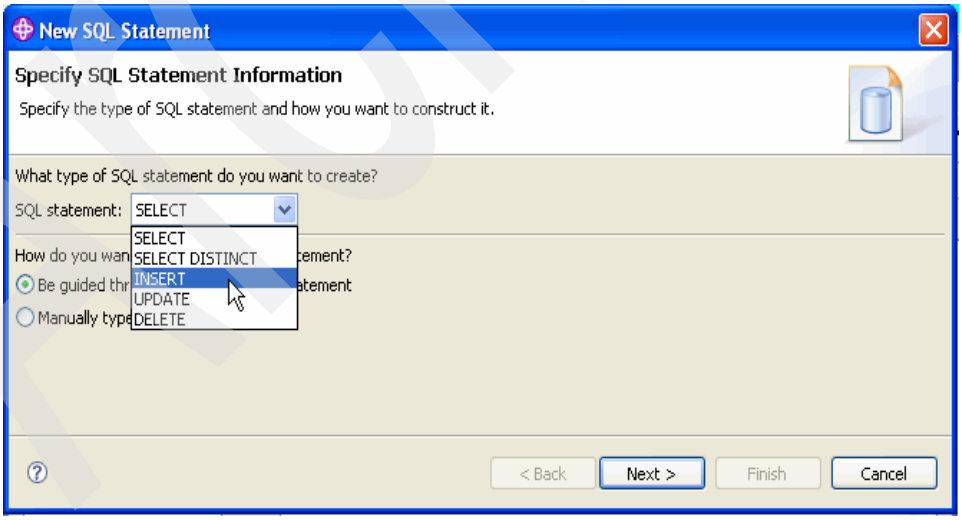


Figure 10-149 New SQL Statement

13. Choose a table from the list of available tables and click **>**. Click **Next**, as shown in Figure 10-150 on page 345.

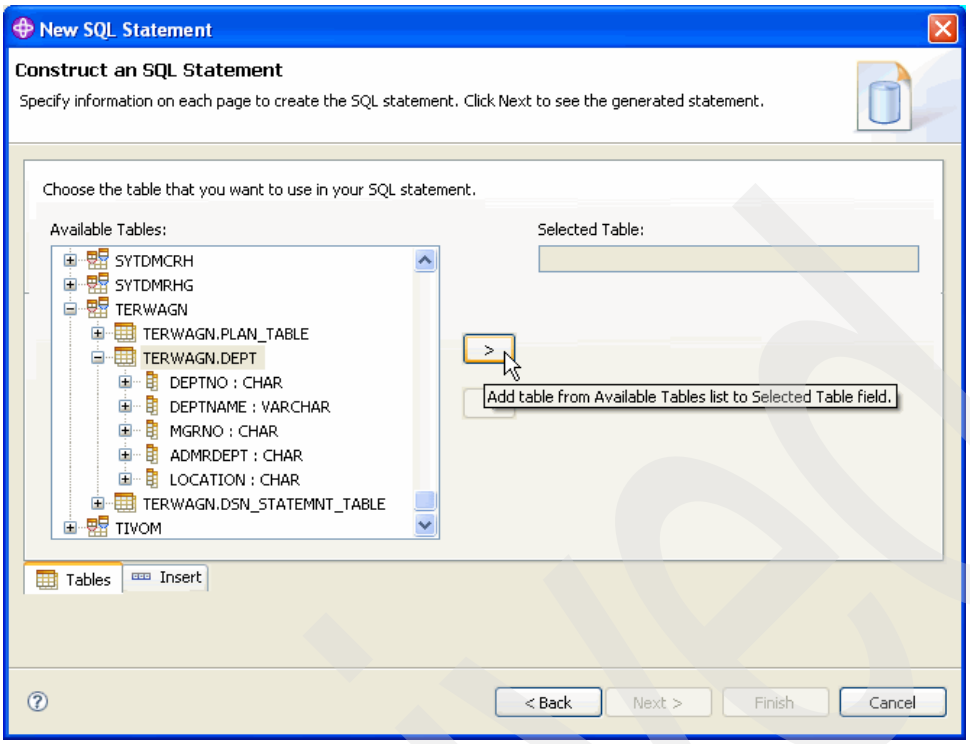


Figure 10-150 Construct an SQL statement

14. Click the **Insert** tab to specify values for a new row, as shown in Figure 10-151.

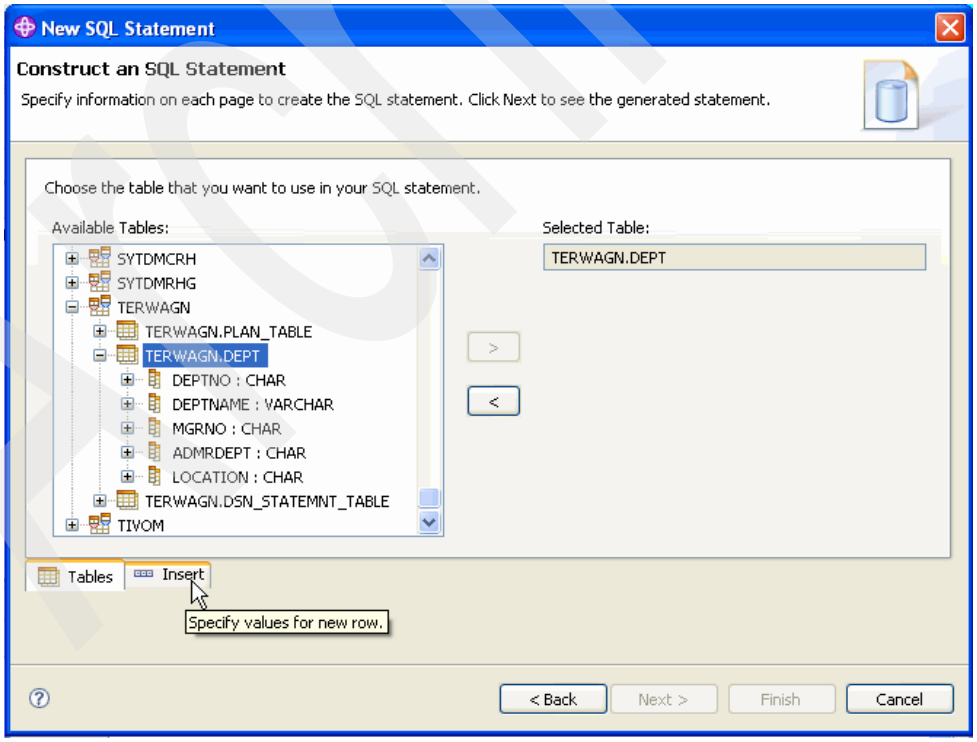


Figure 10-151 Construct an Insert statement

15. Enter host variables names (starting with ":") in the Value column. When done, click **Next**, as shown in Figure 10-152.

> **Tip:** When typing a value or a variable, press Enter at the end to validate it.

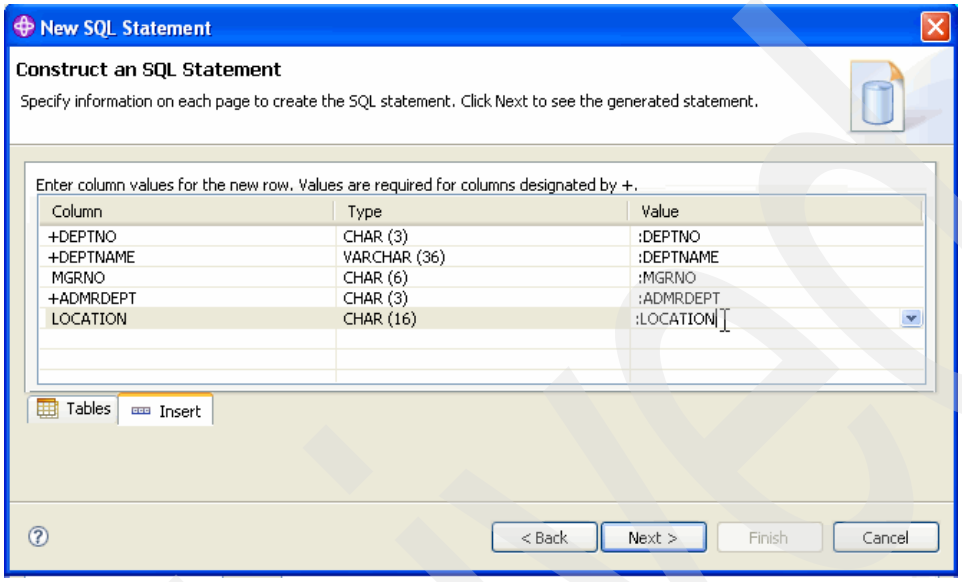


Figure 10-152 Enter host variables

16. We now see the generated SQL statement. If you change it, do not forget to parse it by clicking the **Parse** button. You can click the **Run** button to test it as well. Click **Finish**, as shown in Figure 10-153.

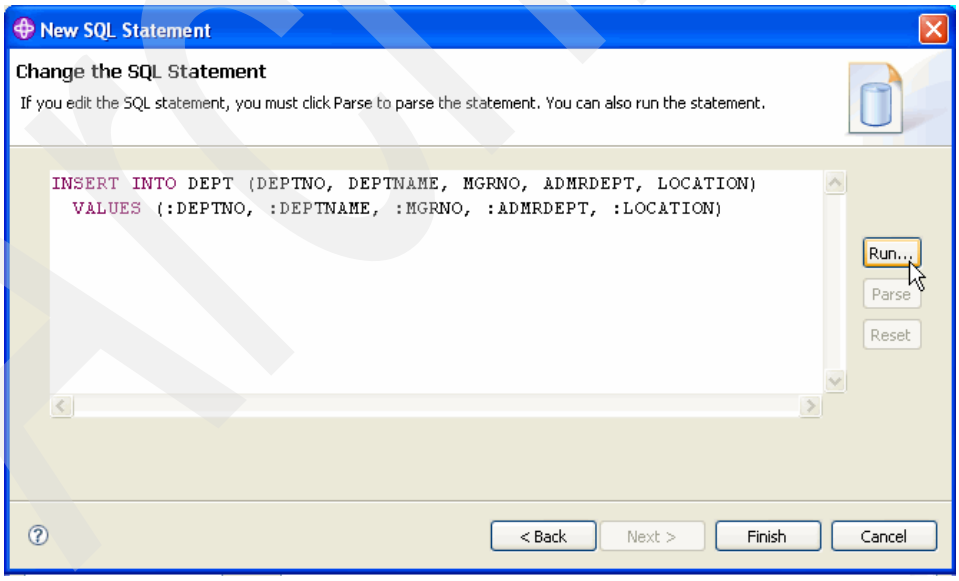


Figure 10-153 Change and run SQL statement

17. Because it is only an Insert statement, we will not get a result set. Select **None** in the **Result Set** drop-down list. Click **Next**, as shown in Figure 10-154 on page 347.

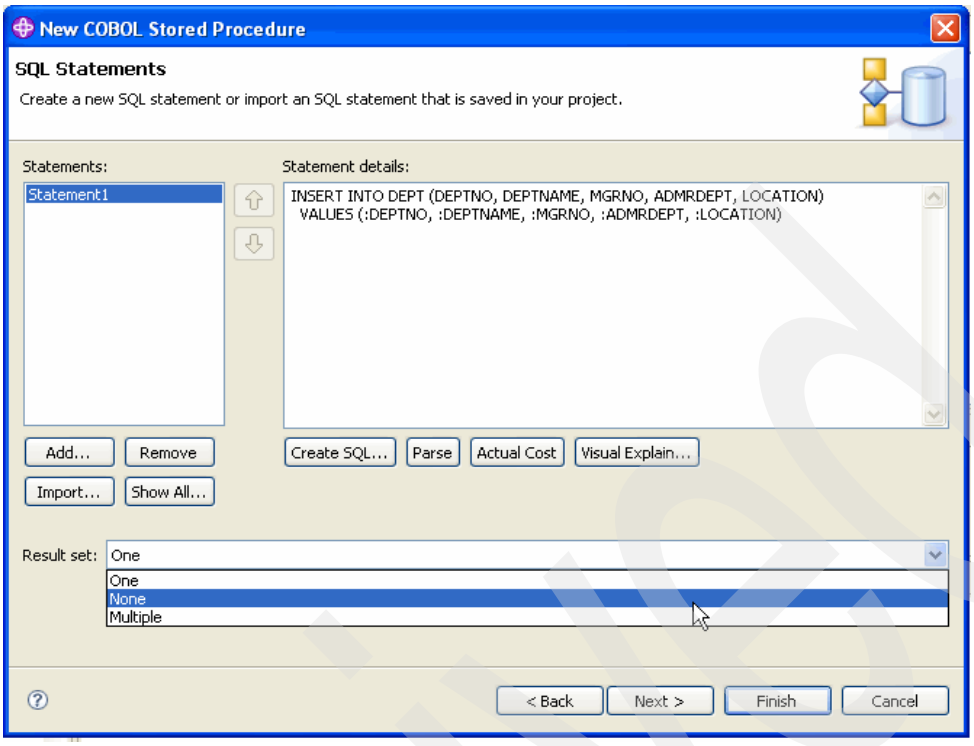


Figure 10-154 Setting Result set

18. We define the parameters of the stored procedure. Click **Add...**, as shown in Figure 10-155.

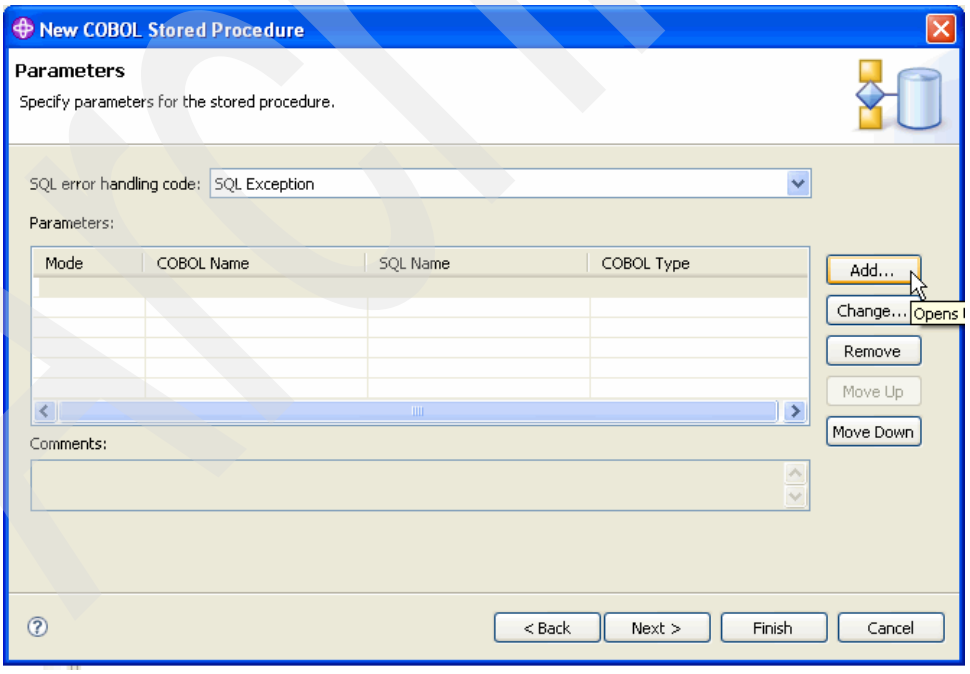


Figure 10-155 Stored procedure parameters

19. Select Parameter mode **OUT**, Enter a name (RET-CODE) and select the SQL type (**Integer**). Click **OK**, as shown in Figure 10-156.

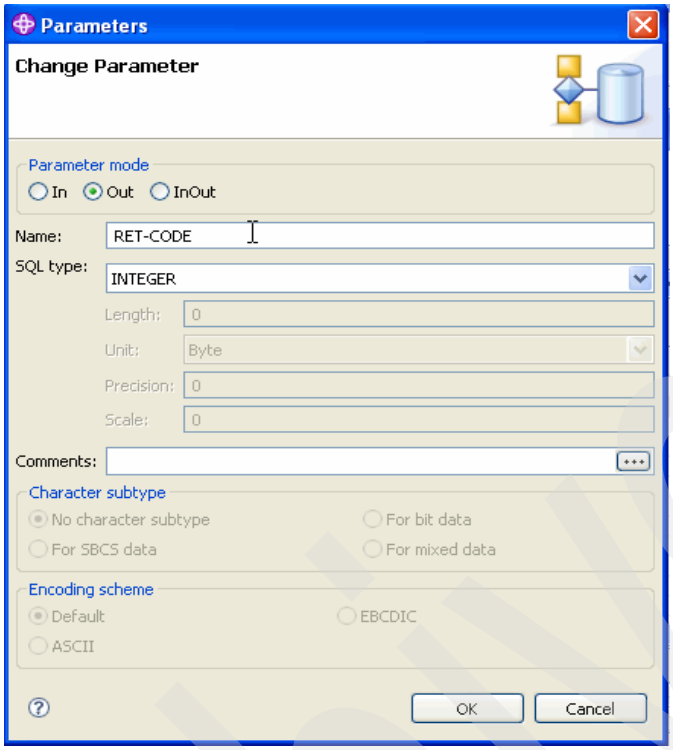


Figure 10-156 Add Out parameter

20. When finished, click **Next**, as shown in Figure 10-157 on page 349.

Archived

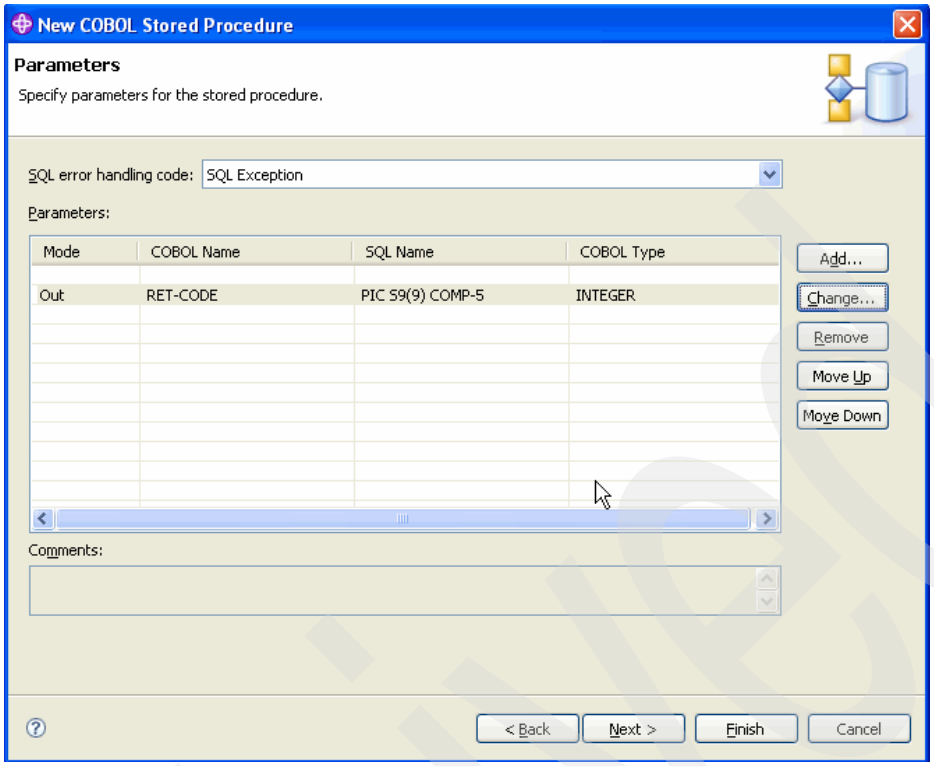


Figure 10-157 Parameters list

21. Enter a Collection ID if you wish. You can leave it empty (in that case, NULLID will be used). A collection ID is related to packages and permits group packages sharing something in common. For more information, read the DB2 documentation. Click **Advanced**, as shown in Figure 10-158.

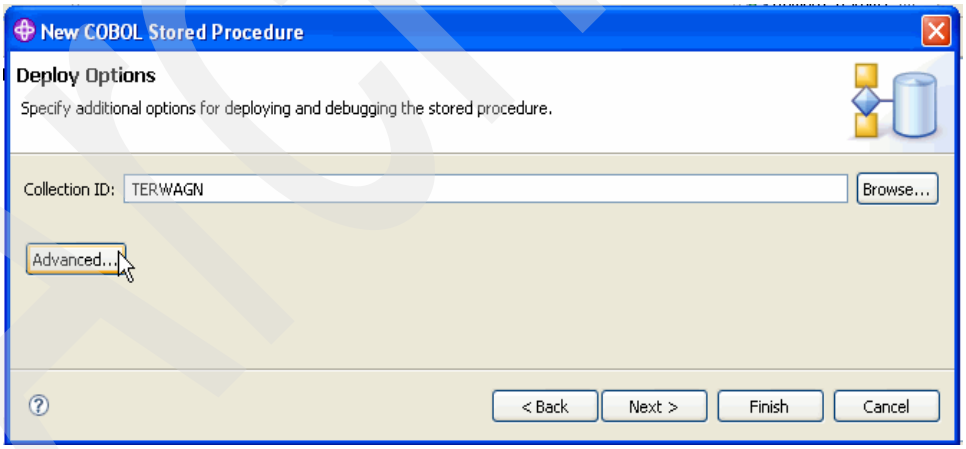


Figure 10-158 Deploy Options

22. Check **Build Stored Procedure for Debugging** and enter a WLM Environment if you wish. If not, your stored procedure will run in the DB2-managed Address space. Click **OK**, as shown in Figure 10-159.

Tip: A WLM-managed address space is preferable (but it needs to be defined on the mainframe) because it offers an isolated environment, the ability to stop a looping stored procedure, or to refresh the memory when you deploy a new version of the stored procedure.

These options can be changed later.

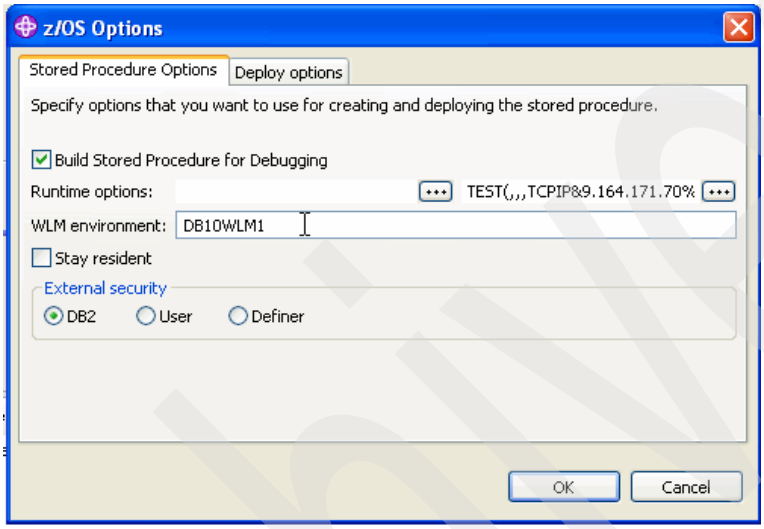


Figure 10-159 Stored procedure options

Archive

23. We are now able to select code fragments to include at four different locations of the generated program. We add a fragment of code before the return of the program. Enter the name of the file (local file) or click the **...** button to navigate the local disk and select the file. Click **Next**, as shown in Figure 10-160.

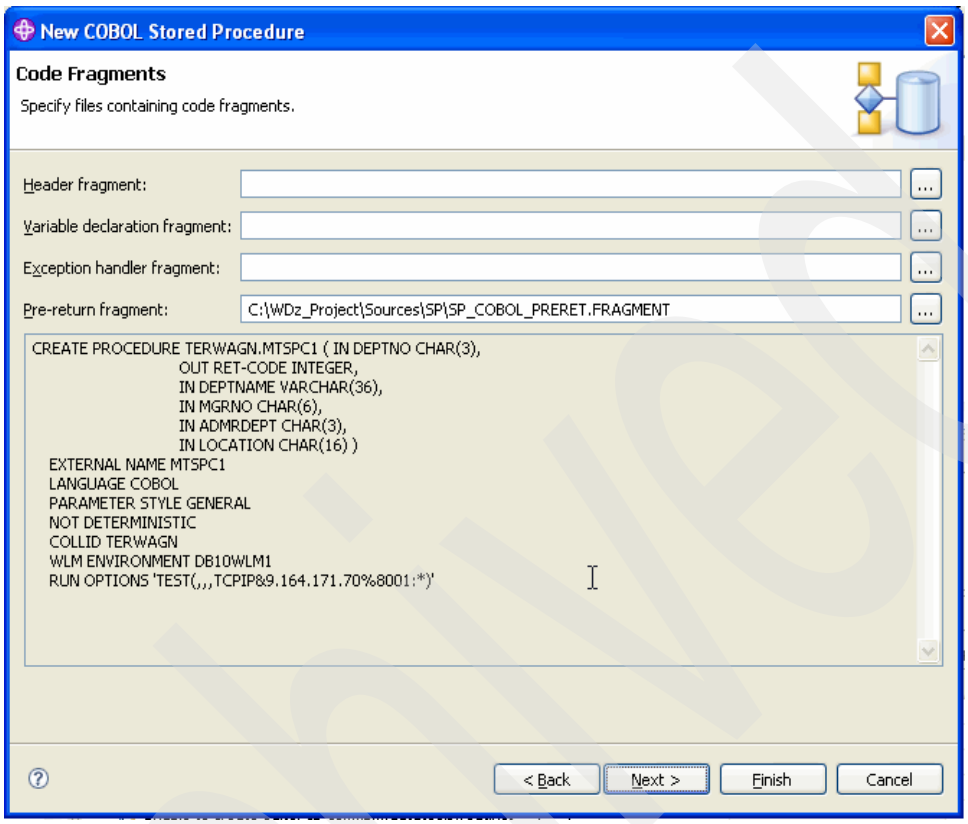


Figure 10-160 Add code fragments

Archived

24. Now we see the summary before we generate the program. Click **Show SQL** if you wish to see the SQL statement again. Click **Finish**, as shown in Figure 10-161.

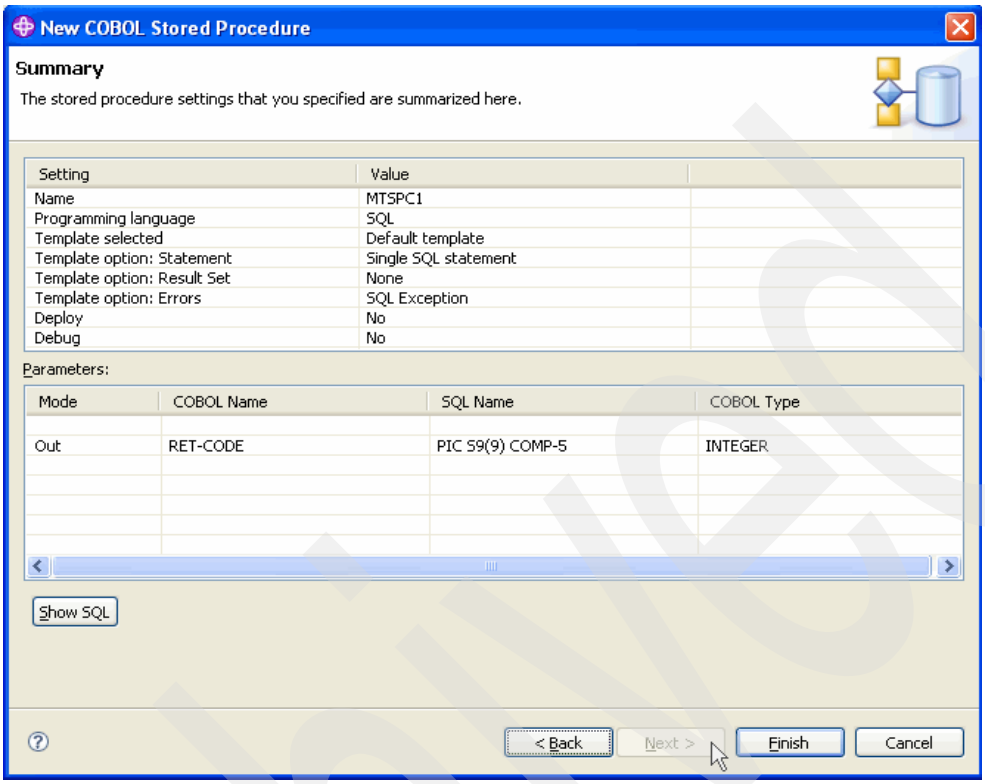


Figure 10-161 New stored procedure: summary

Archived

25. When the stored procedure is generated, WD/z opens it automatically and shows the COBOL (or PL/I) source. Note the different tabs at the bottom of the view, as shown in Figure 10-162.

> **Note:** You are remotely editing this code and are able to add your own COBOL/PL/I instructions if you wish.

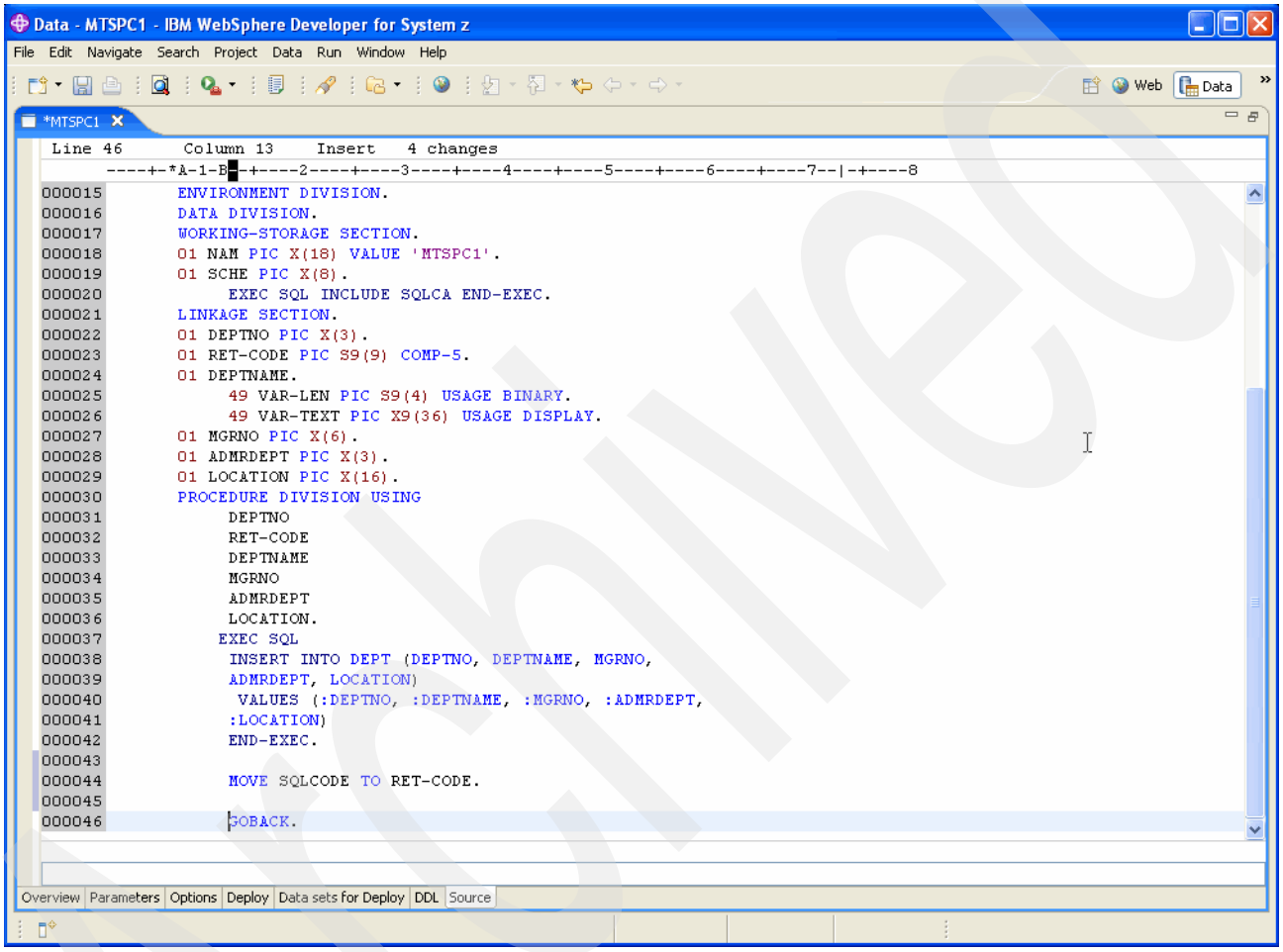


Figure 10-162 Generated stored procedure

Archive

26. Click the **Data sets for Deploy** tab and change data sets names as needed, especially the load module location, which must be accessible by the runtime environment (DB2-managed address space or WLM-managed address space) and the link libraries, which must match to your installation (Language Environment link libraries, DB2 loadlib, and so on), as shown in Figure 10-163.

Tip: The separator character is a simple space.

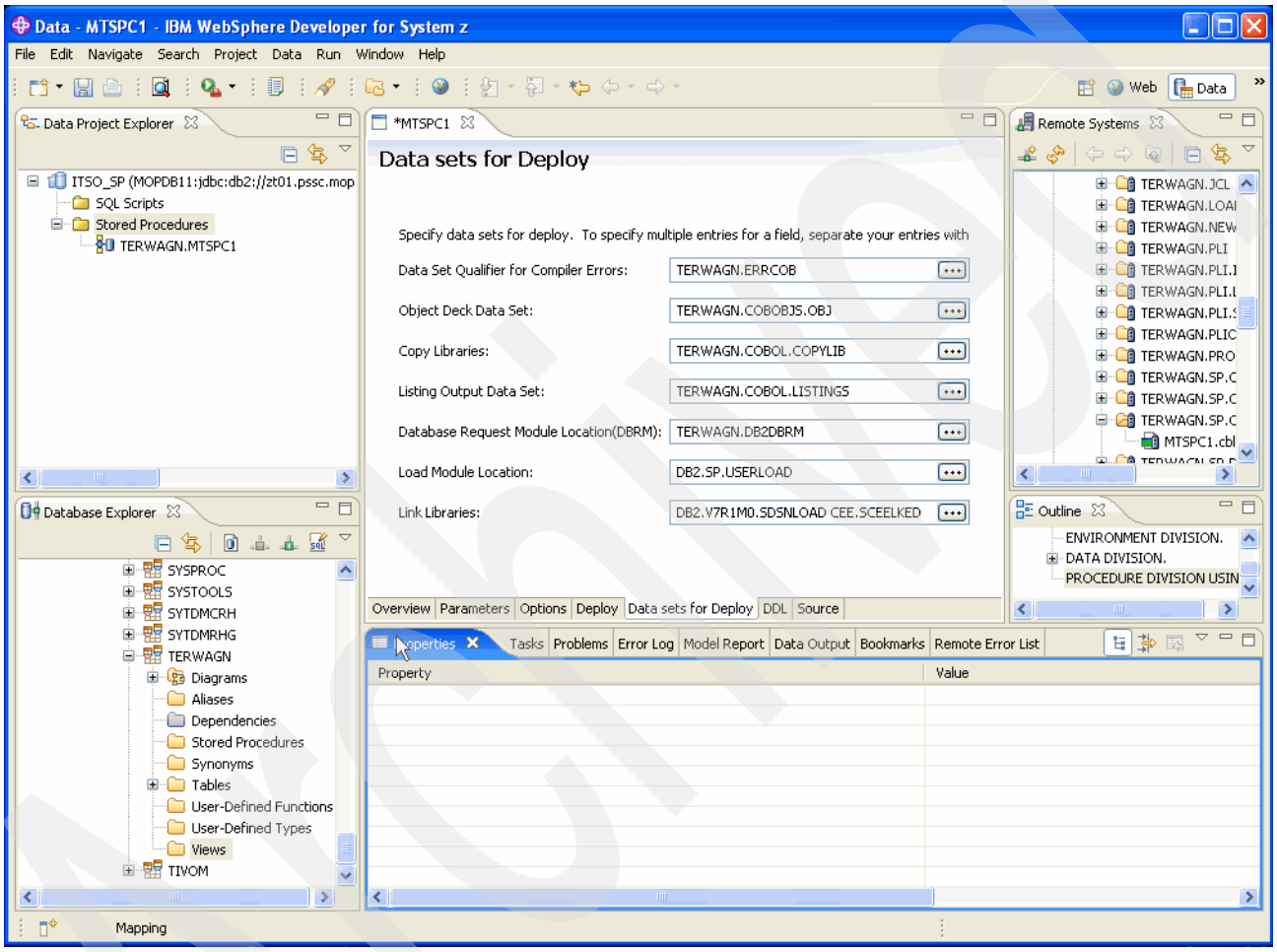


Figure 10-163 Data sets for Deploy

27. Click the **Deploy** tab, where we see several deploy options. We suggest adding the Bind option ACTION(REPLACE). If you deploy the stored procedure several times, this option will permit the replacement of the previous bind package. Click the **...** button for the Bind options field and enter the option, as shown in Figure 10-164.

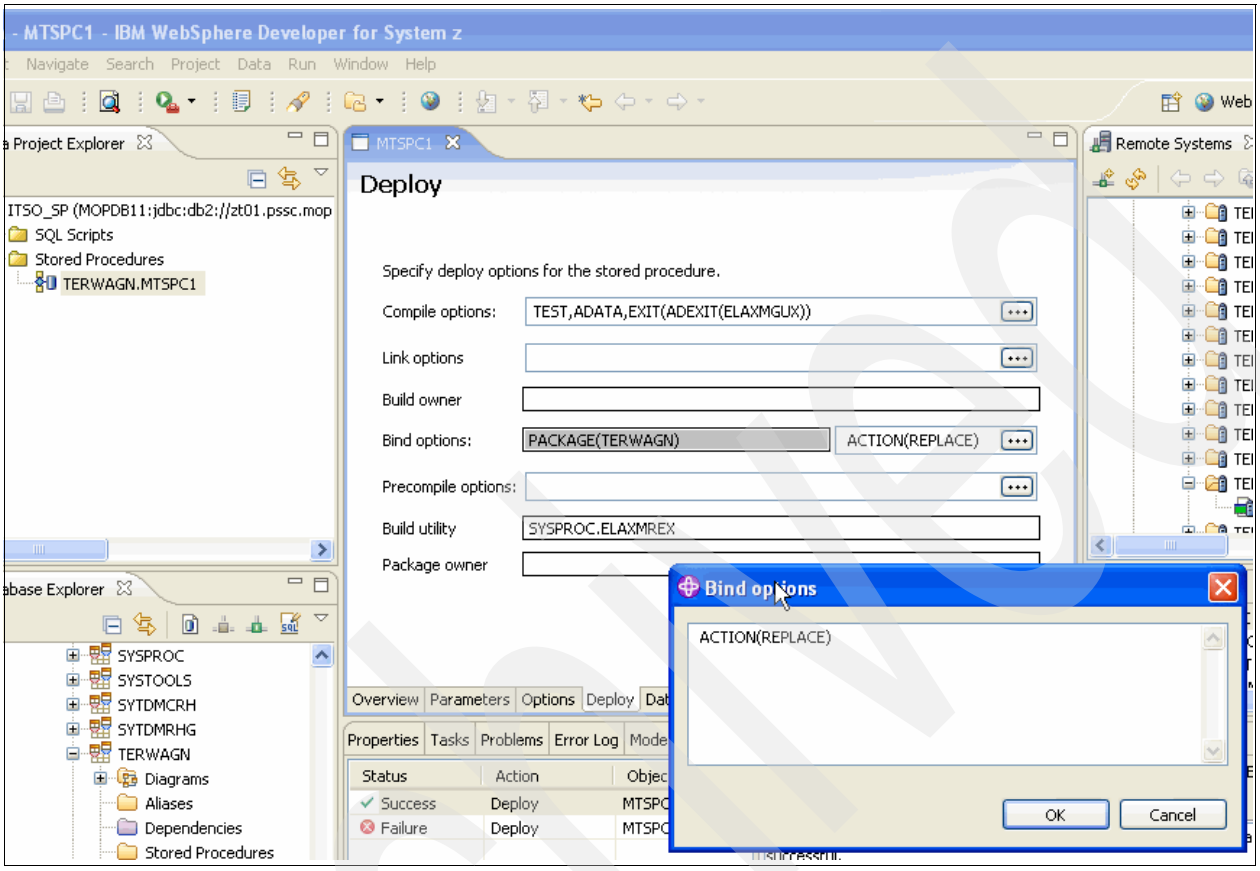


Figure 10-164 Bind options

28. We now deploy the stored procedure. Open the Data Project Explorer view, right-click the stored procedure you created, and select **Deploy**, as shown in Figure 10-165.

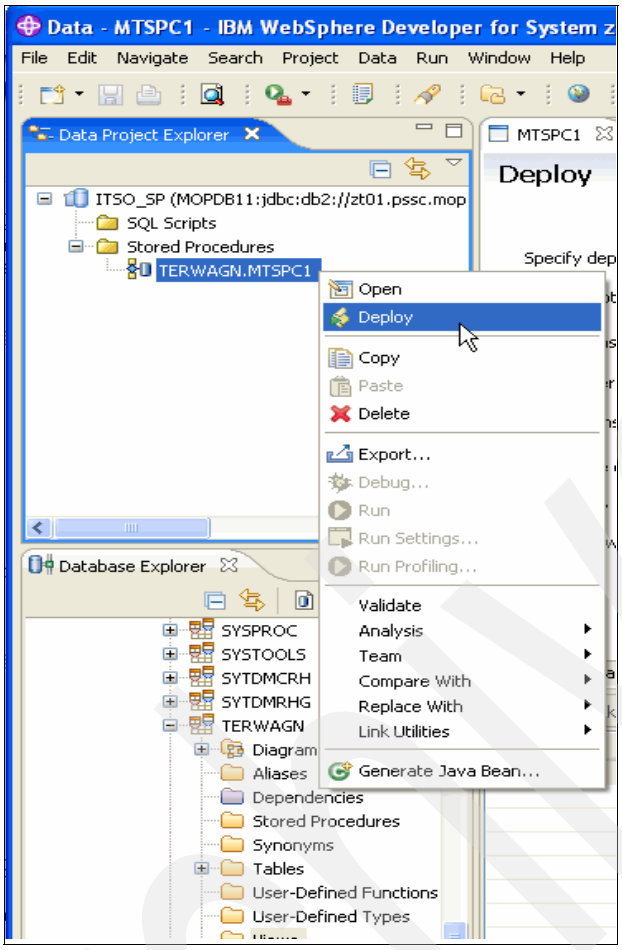


Figure 10-165 Stored procedure deploy

29. When done, WD/z shows the Data Output view with the status of the Deploy action, which should be Success, as shown in Figure 10-166 on page 357.

Archived

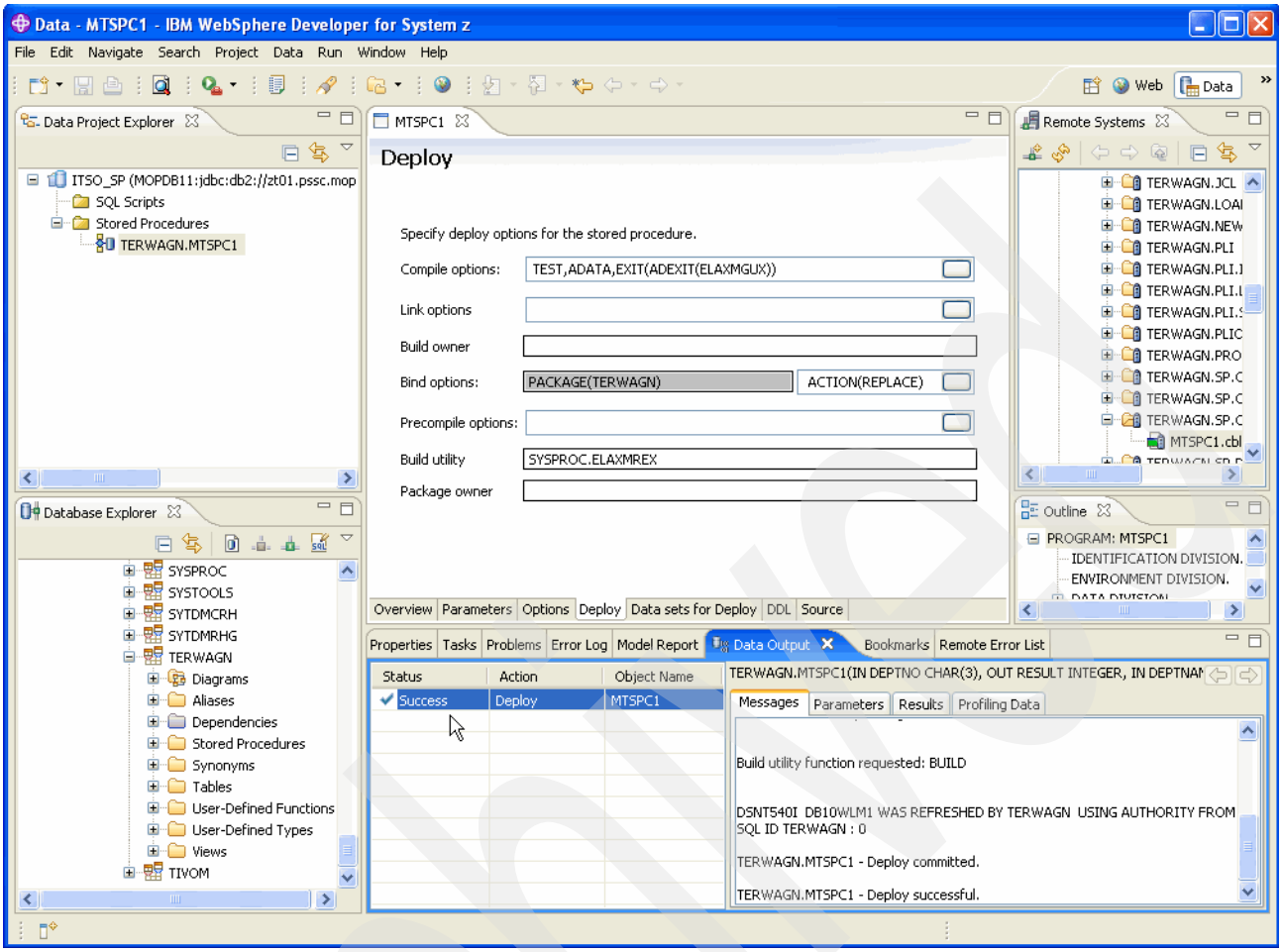


Figure 10-166 Data Output view: deploy action status

30. Go to the Message tab (in the right pane of the Data Output view) and review the information. If there is a failure, this tab should help you determine the cause of the failure, as shown in Figure 10-167.

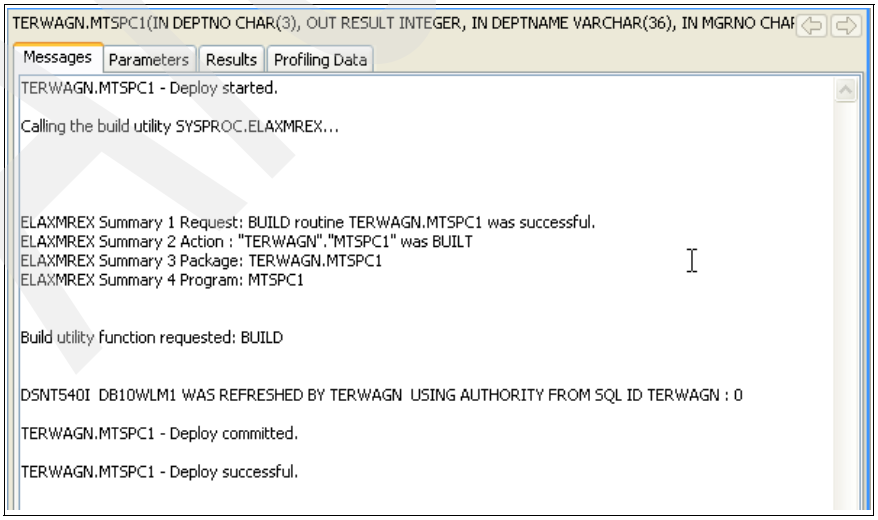


Figure 10-167 Deploy messages

31. Before testing the stored procedure, we take a look at the content of the table. Open the Database Explorer view, navigate to your connection to find the table, right-click it, and select **Data** → **Edit**, as shown in Figure 10-168.

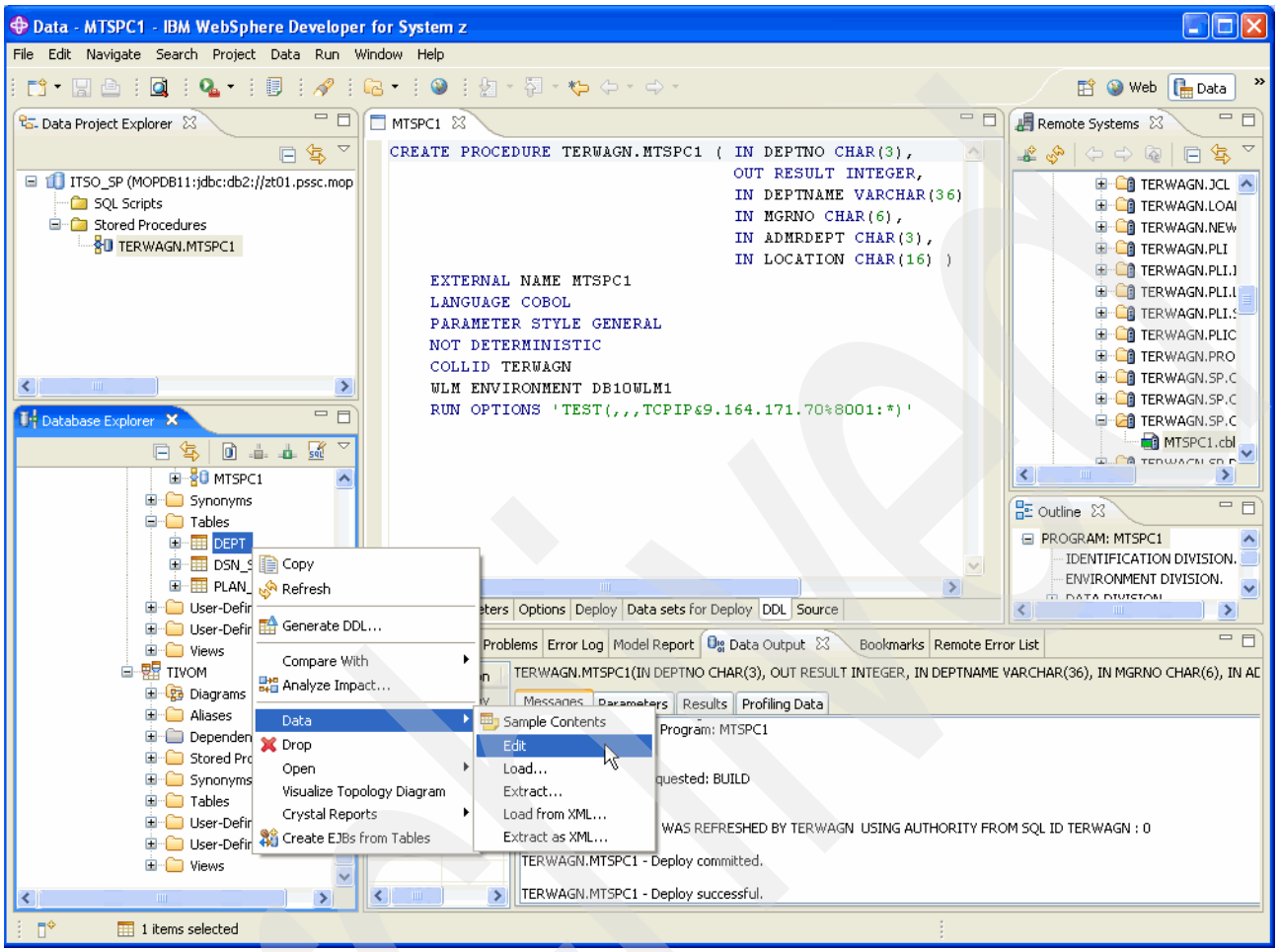
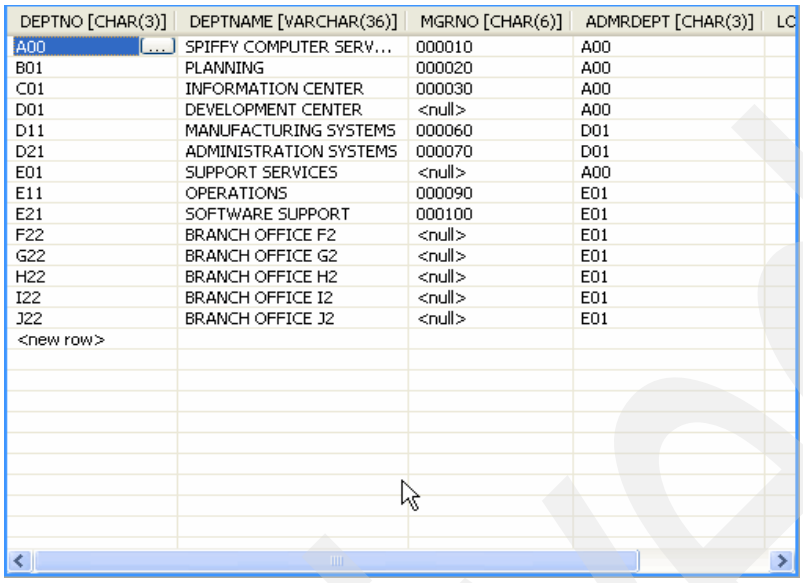


Figure 10-168 DB2 table: How to edit

32. An edit view where you are able to insert, update and delete rows opens, as shown in Figure 10-169.



DEPTNO [CHAR(3)]	DEPTNAME [VARCHAR(36)]	MGRNO [CHAR(6)]	ADMRDEPT [CHAR(3)]	LC
A00 ...	SPIFFY COMPUTER SERV...	000010	A00	
B01	PLANNING	000020	A00	
C01	INFORMATION CENTER	000030	A00	
D01	DEVELOPMENT CENTER	<null>	A00	
D11	MANUFACTURING SYSTEMS	000060	D01	
D21	ADMINISTRATION SYSTEMS	000070	D01	
E01	SUPPORT SERVICES	<null>	A00	
E11	OPERATIONS	000090	E01	
E21	SOFTWARE SUPPORT	000100	E01	
F22	BRANCH OFFICE F2	<null>	E01	
G22	BRANCH OFFICE G2	<null>	E01	
H22	BRANCH OFFICE H2	<null>	E01	
I22	BRANCH OFFICE I2	<null>	E01	
J22	BRANCH OFFICE J2	<null>	E01	
<new row>				

Figure 10-169 Table editing

Archived

33. We are now ready to test the stored procedure. Open the Data Project Explorer view, right-click the stored procedure, and select **Run**, as shown in Figure 10-170.

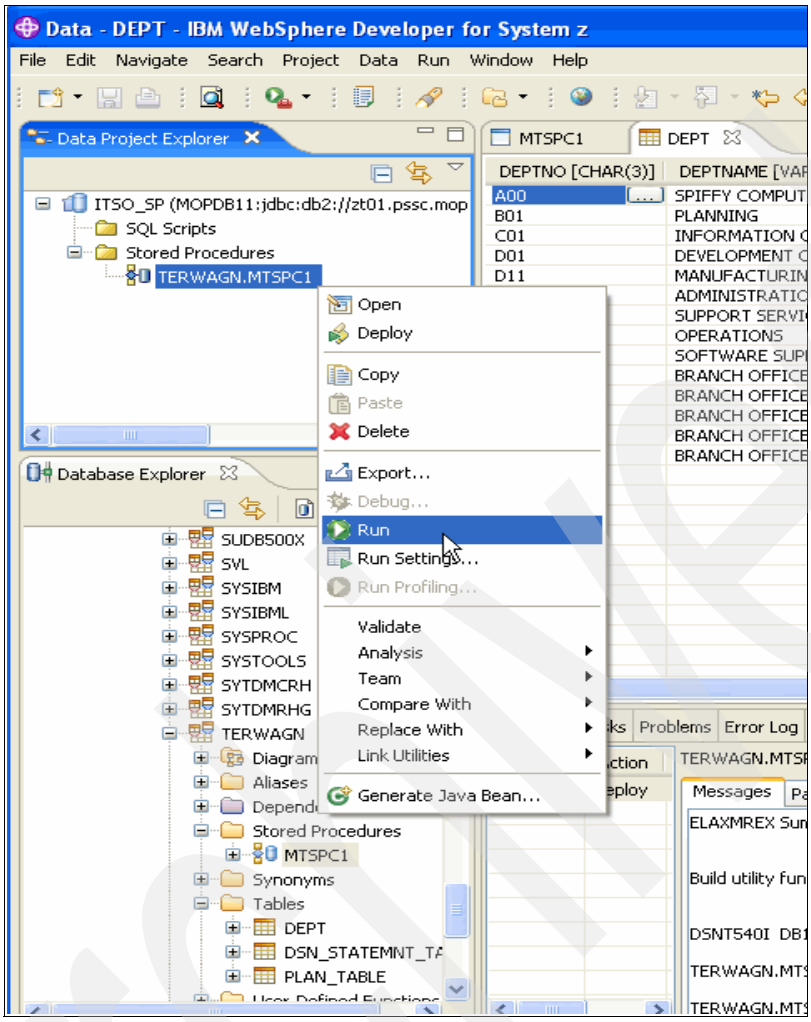


Figure 10-170 Test stored procedure

34. Because we built the stored procedure for debugging, WD/z is contacted by the Debug Tool and switches to the Debug perspective. You could see a pop-up window asking if you accept the switch. Click **Yes**. You can check the **Remember my decision** box if you wish, as shown in Figure 10-171 on page 361.

Attention: Your IP address may have changed since you generated the stored procedure (for example, after restarting your workstation). In that case, you must change this IP address in the Run options tab and deploy again. It is also possible to execute a SQL ALTER command on the stored procedure to change these Run options.

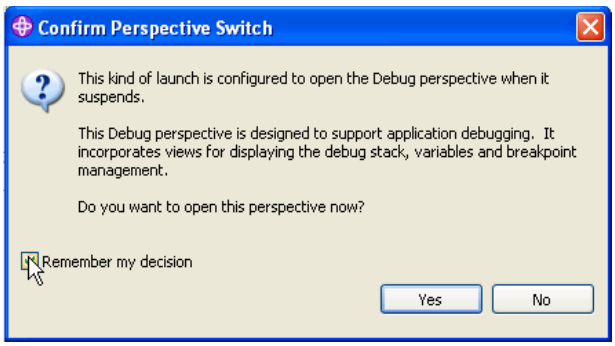


Figure 10-171 Confirm Perspective Switch

35. The Debug perspective appears, as shown in Figure 10-172.

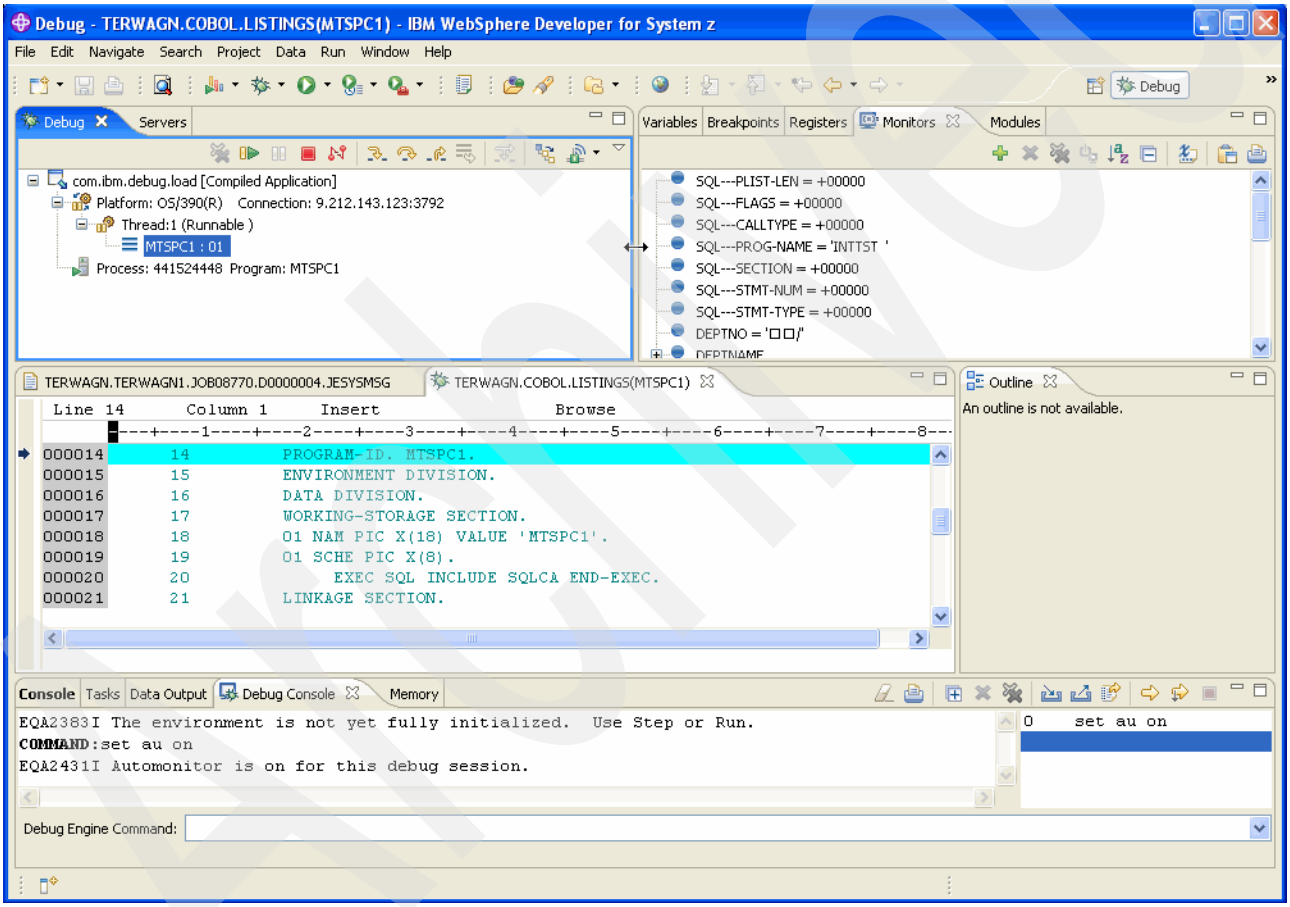


Figure 10-172 Debug perspective

36. When the program is terminated, the Data Output view shows the execution messages and results, as shown in Figure 10-173.

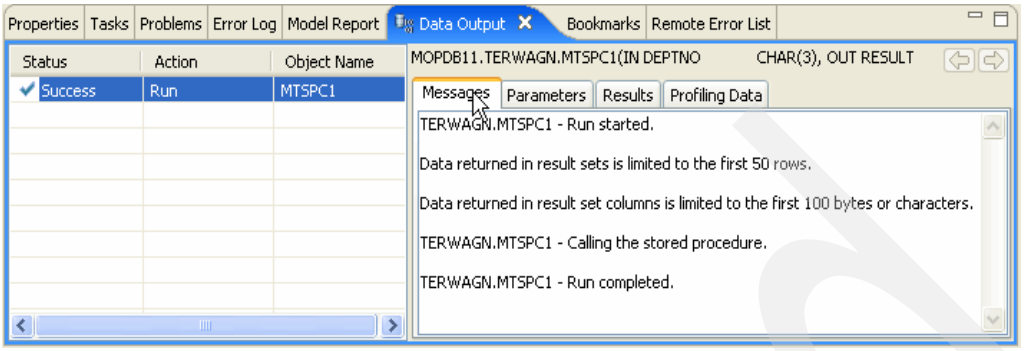


Figure 10-173 Data Output messages

37. To verify if the row is inserted in the table, edit the table again, as shown in Figure 10-174.

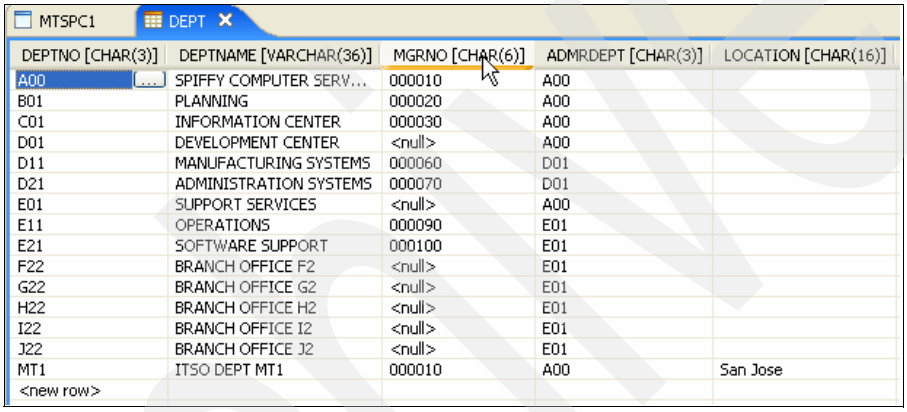


Figure 10-174 Table content verification

38. You are now ready to create additional stored procedures or maintain the existing ones.

10.7 Fault Analyzer plug-in for WD/z

Note: The sample batch program used in this section is provided as part of the Web material associated with this book; refer to Appendix B, “Additional material” on page 645 for more details. It is based on the Installation Validation Program (IVP) delivered with Fault Analyzer for WebSphere Developer for System z.

This section describes how to use the Fault Analyzer plug-in technical preview for WebSphere Developer for System z. This component allows you to work with fault entries created by Fault Analyzer during real-time error analysis. It allows you to browse fault entries stored in history files that contain real-time analysis of abending programs. The analysis also links to the source list of an abending program using the LANGX sidefile. In addition, the formatted dump storage can be displayed using mini-dump pages. Mini-dump pages are a subset of pages that are referenced during real-time analysis.

Attention: This plug-in must be installed on the host and on the workstation. We assume that these installations are already done. We suggest you also read the latest information about this plug-in on our Web site.

Follow these steps to use the plug-in:

1. When the plug-in is installed, you can see an additional topic under a connection in the Remote Systems view. Figure 10-175 shows FA Subsystem under the connection ZTEC MOP.

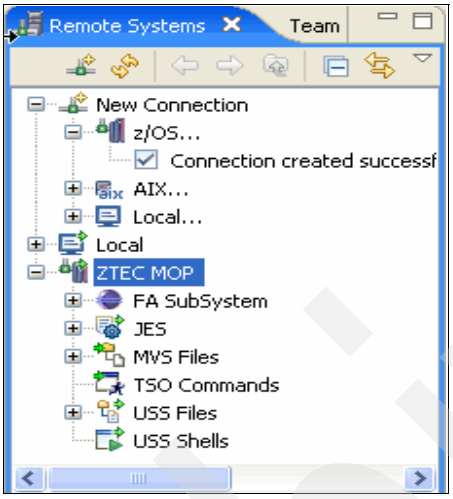


Figure 10-175 FA Subsystem

2. The Fault Analyzer plug-in has its own perspective. We can access it by selecting **Window** → **Open perspective** → **Other**, as shown in Figure 10-176.

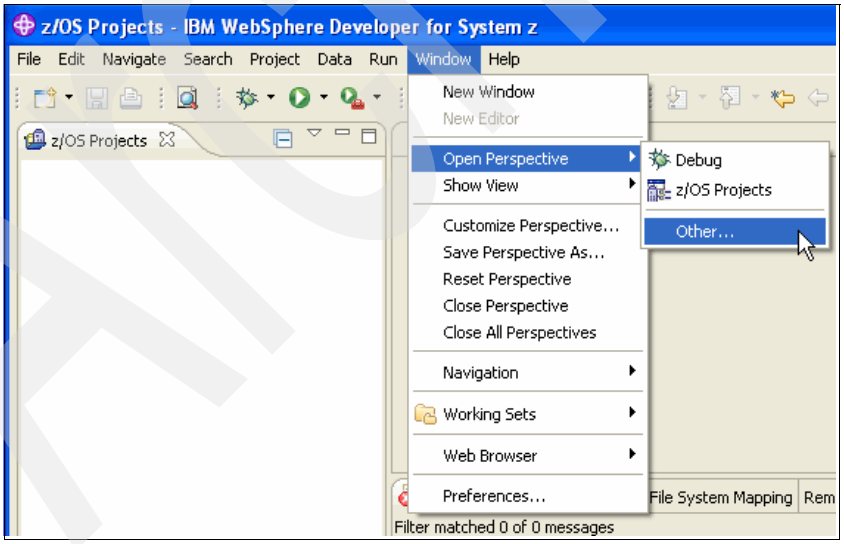


Figure 10-176 Fault Analyzer: Retrieving the Fault Analyzer perspective

Archived

3. Select **FA Perspective**, as shown in Figure 10-177.

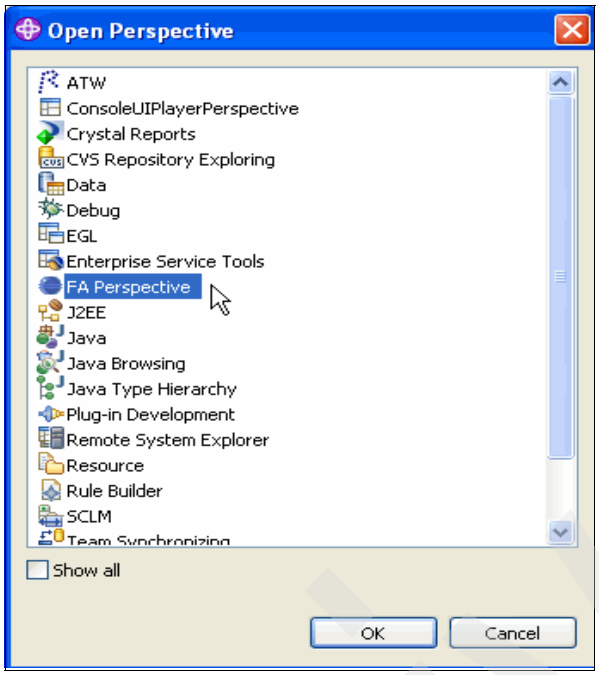


Figure 10-177 Choose the FA Perspective

4. If it is the first time you have started the FA Perspective since you opened the workspace, you should see an empty view named FA Artifacts. Right-click **FA Artifacts** and select **Add New History File**, as shown in Figure 10-178.

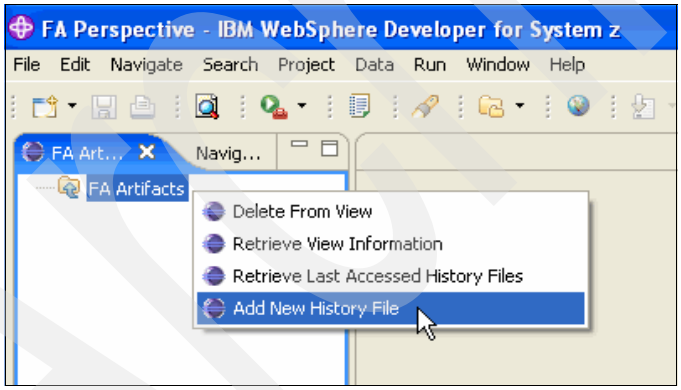


Figure 10-178 Add New History File

Note: Fault history files consist of fault entries that are stored separately. Fault entries contain the following types of information:

- Real-time analysis key information, such as abend code and the name of the failing program
- Execution environment details, such as job name, system ID, and the date and time when the fault occurred
- The associated real-time analysis report
- Saved mini-dump (if applicable)
- Name of associated SYSMDUMP data set (if applicable)

You can view fault entry details, browse the real-time analysis report, start batch and interactive fault reanalysis, or delete fault entries from an interactive display of a fault history file.

Although a fault history file can be either a partitioned data set (PDS) or a partition data set extended (PDSE), PDSE data sets are recommended because PDS member update is not serialized and therefore you might lose fault entries.

5. Select the connection and type in the History File Name. Click **OK**, as shown in Figure 10-179.

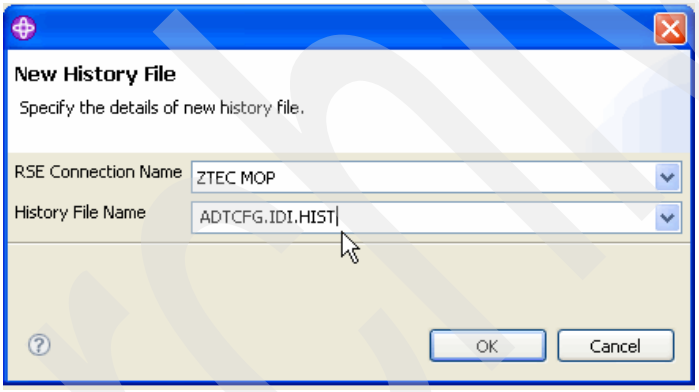


Figure 10-179 Enter the History File Name

6. You should see, in the FA Artifacts view, the System Name and the History File Name in the History Files folder. It is possible to add several systems and to have several History Files on the same system. View the content list of the History File, as shown in Figure 10-180.

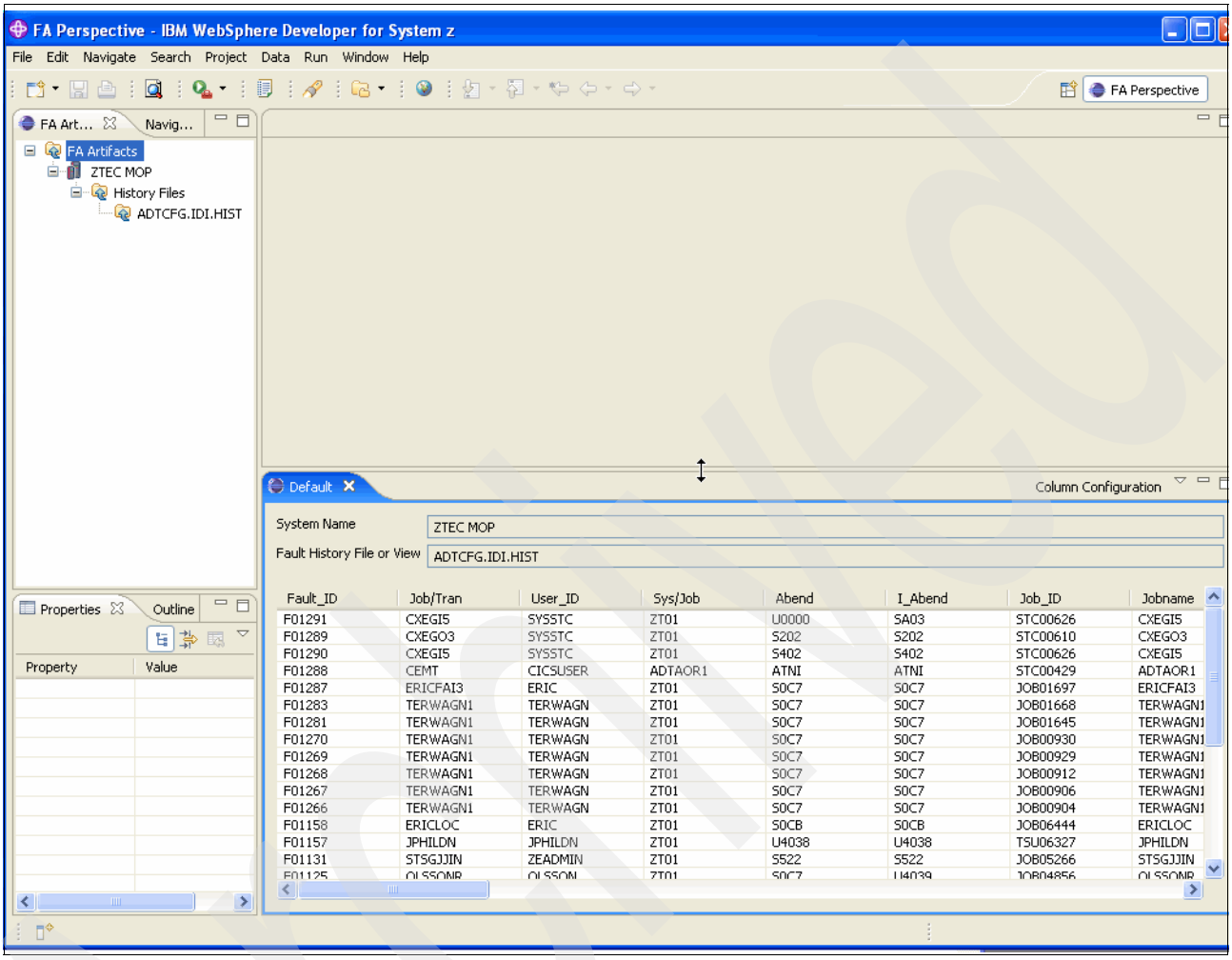


Figure 10-180 Fault Analysis viewing of an history file

7. We now run the sample program, which abends, to be able to analyze it. Open the z/OS Projects perspective and create a new z/OS Project by right-clicking in the z/OS Projects view and selecting **New** → **z/OS Project**, as shown in Figure 10-181 on page 367.

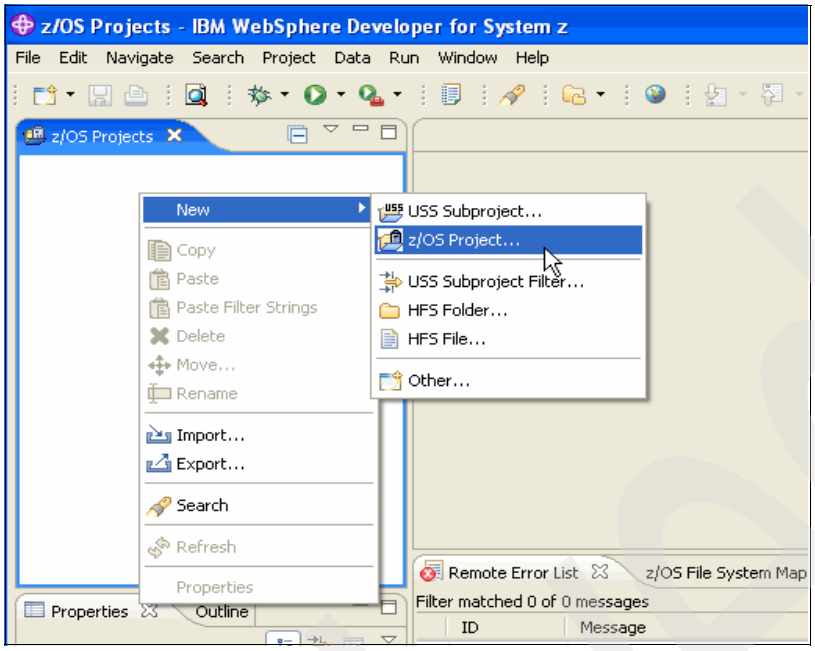


Figure 10-181 Fault Analysis: create a new z/OS Project

8. Give the project a name and select **Create an MVS subproject**. Click **Finish**, as shown in Figure 10-182.

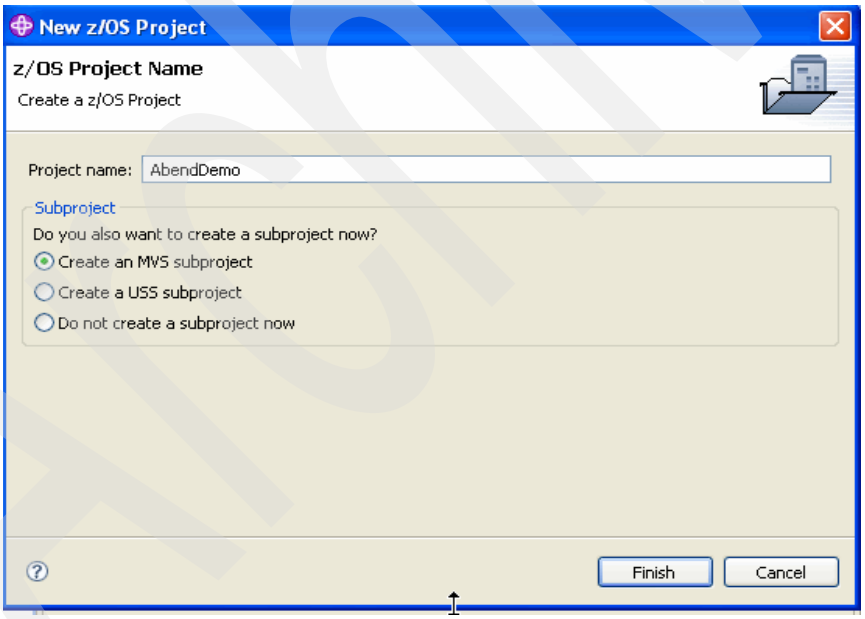


Figure 10-182 Fault Analysis: new project name

9. Give the subproject a name and click **Finish**, as shown in Figure 10-183.

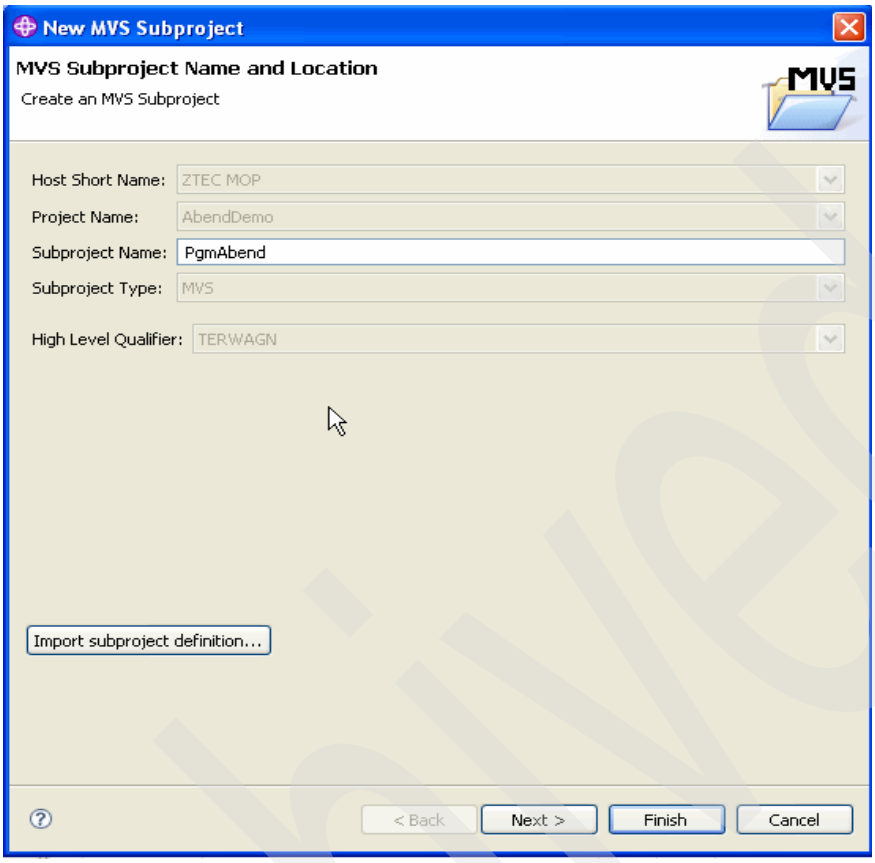


Figure 10-183 Fault Analysis: new subproject name

Archive

10. Assuming that the program is already uploaded to a PDS, we add this program to our subproject. Open the Remote System view, select the program source, and right-click it. Select **Add to Subproject**, as shown in Figure 10-184.

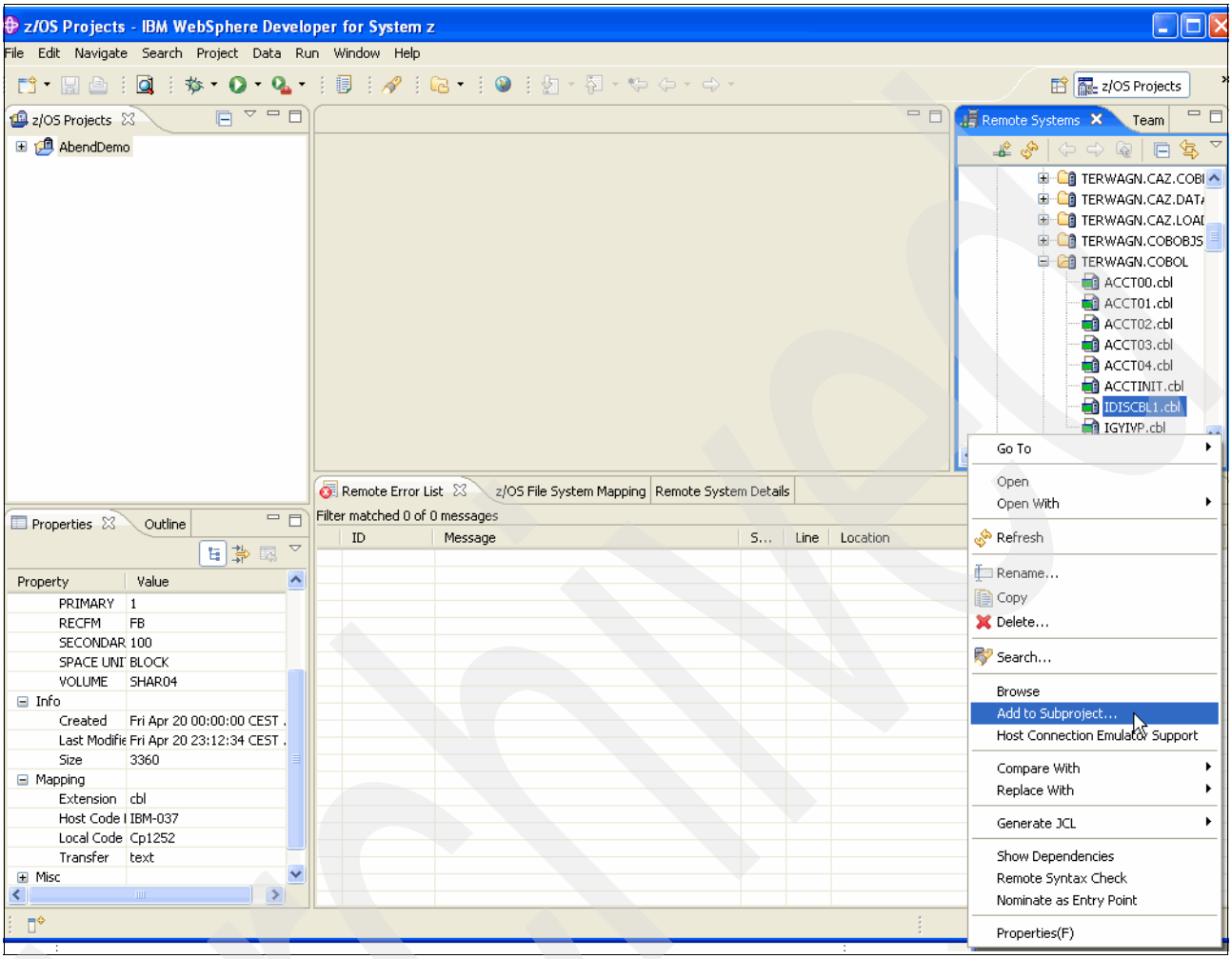


Figure 10-184 Fault Analysis: add the program to the subproject

11. Choose the correct project and subproject. Click **Finish**, as shown in Figure 10-185.

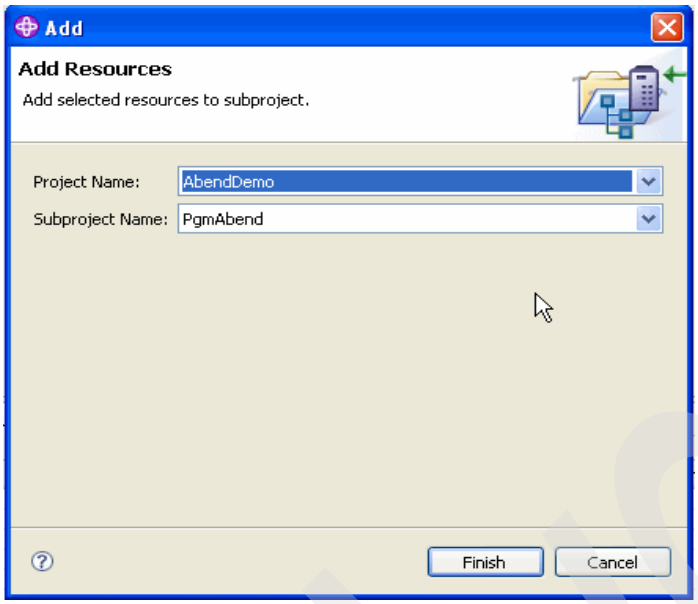


Figure 10-185 Fault Analysis: select the project and the subproject

12. To run the sample program, we generate a JCL to compile-link and run the program. But because the Fault Analysis technical preview supports only sidefile listings, we must change the properties (used by the JCL generator) of the program to produce a sidefile from the listing using a Fault Analyzer program (IDILANGX). Right-click the program and select **Properties**, as shown as Figure 10-186 on page 371.

Archived

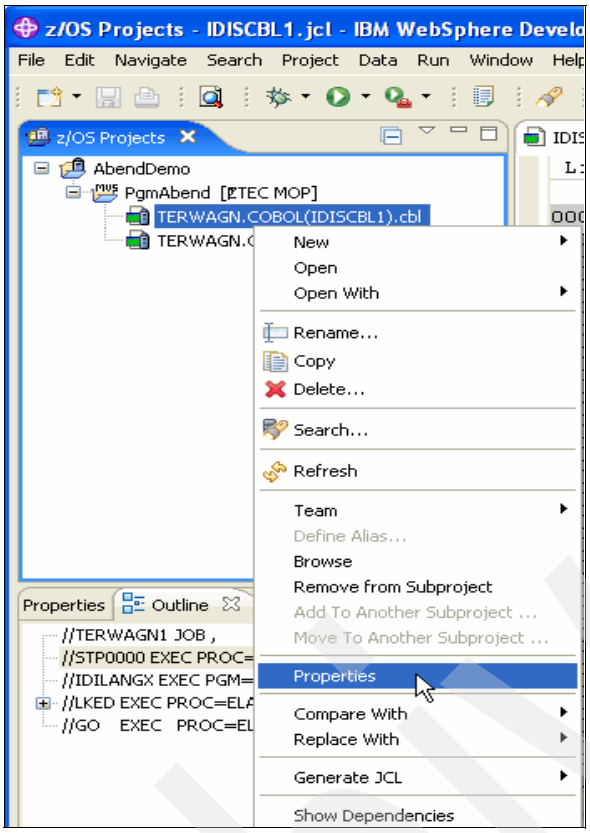


Figure 10-186 Fault Analysis: properties of program

a. Select **COBOL Settings**, expand the enabled procedure (it should be **ELAXFCOC**), select it, and click **Add Step**, as shown in Figure 10-187.

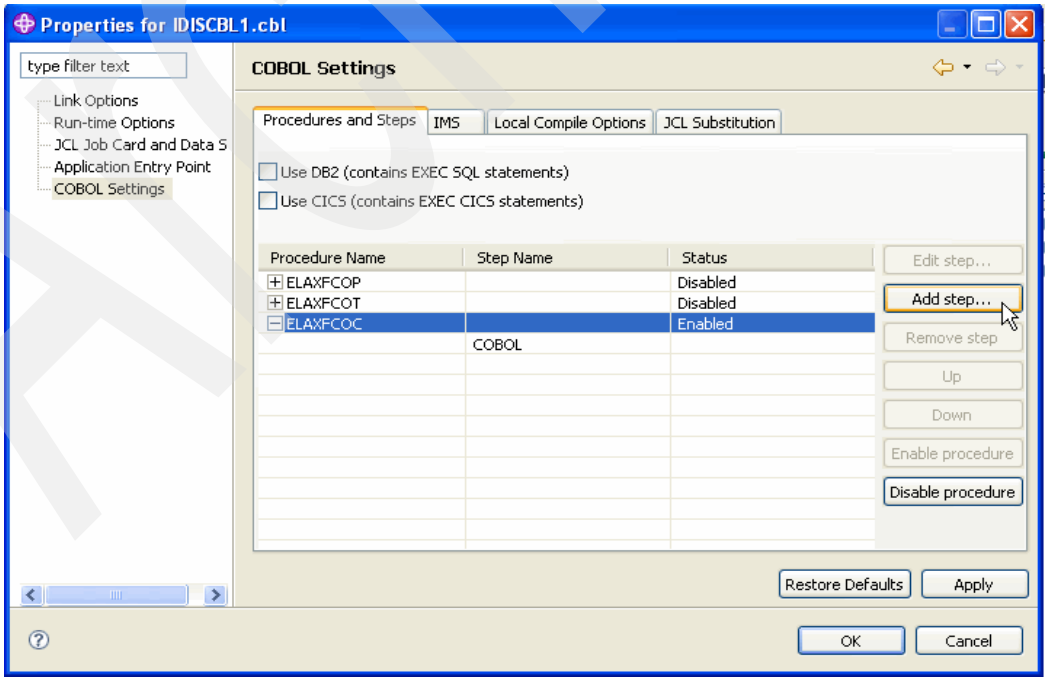


Figure 10-187 Fault Analysis: COBOL Settings

Archived

b. Name the step LANGX and enter the text shown in Example 10-2. Adapt it to your environment, as shown in Figure 10-188.

Example 10-2 LANGX step

```
//IDILANGX EXEC PGM=IDILANGX,REGION=4096K,
// PARM='IDISCBL1 (COBOL ERROR'
//STEPLIB DD DISP=SHR,DSN=FAULTANL.V7R1.SIDIMOD1
//LISTING DD DISP=(OLD,PASS),DSN=TERWAGN.COBOL.LISTINGS
//IDILANGX DD DISP=SHR,DSN=TERWAGN.COBOL.LANGX
//SYSUDUMP DD SYSOUT=*
```

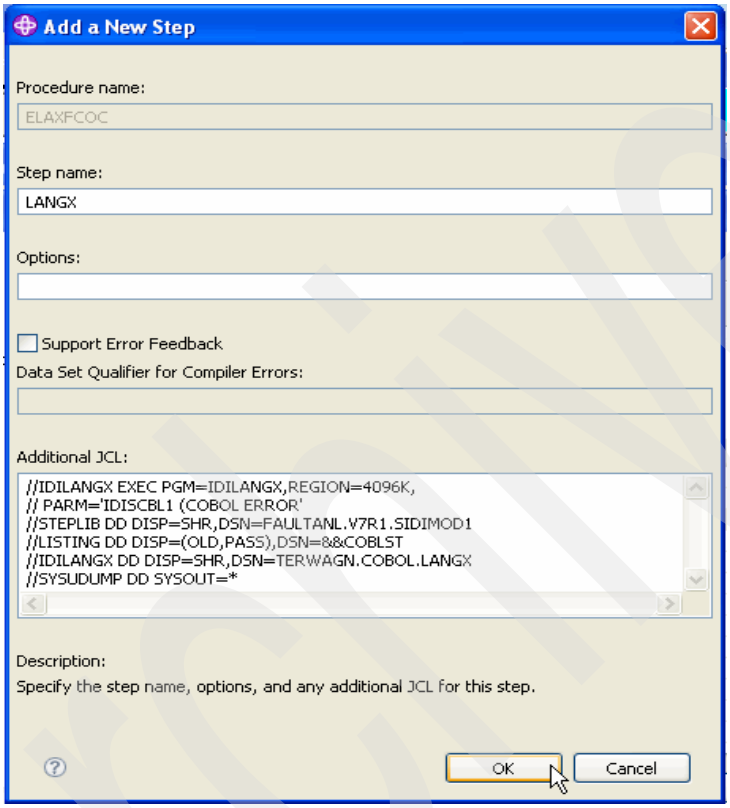


Figure 10-188 Fault Analysis: add LANGX step

c. Because Fault Analyzer needs to point to the LANGX sidefile during the real-time analysis (when an abend occurs), we add a JCL card to the Run time Options. Adapt the sample shown in Example 10-3, Figure 10-189, and Figure 10-190.

Example 10-3 Fault Analysis: add JCL card to Run time Options

```
//IDILANGX DD DSN=TERWAGN.COBOL.LANGX,DISP=SHR
```

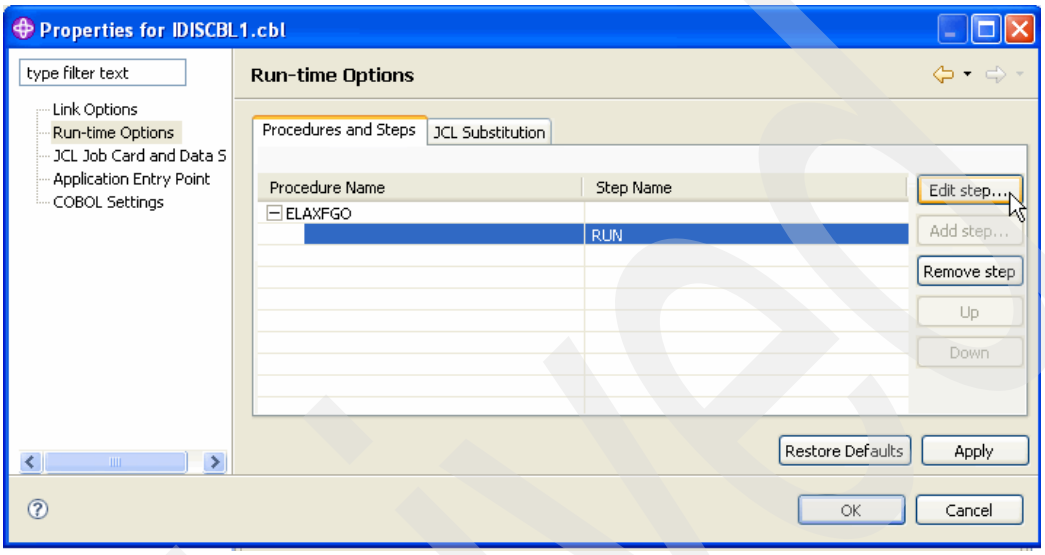


Figure 10-189 Fault Analysis: Run time Options

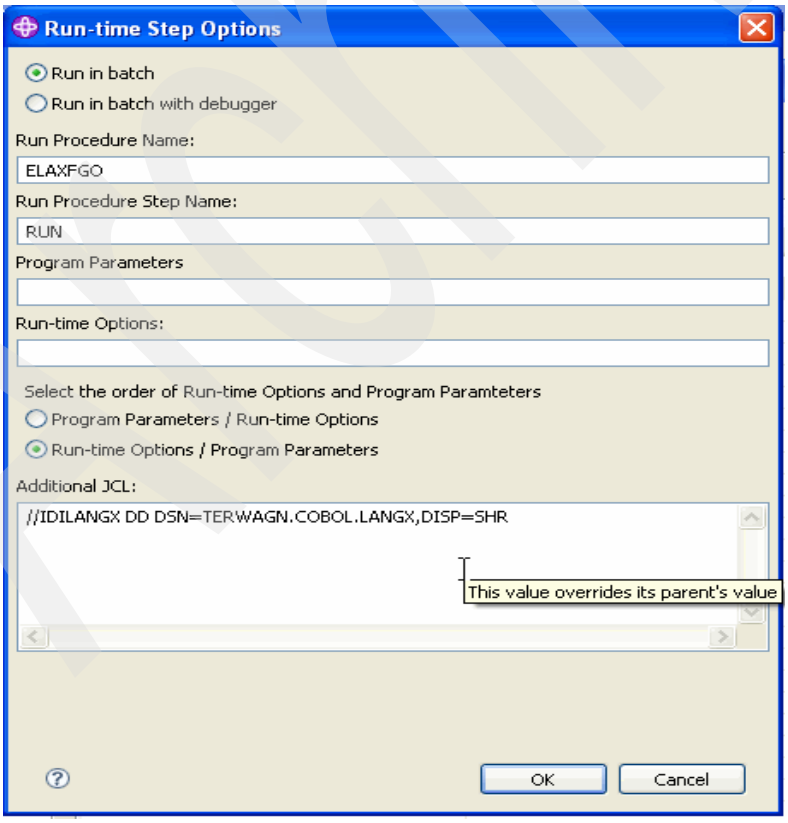


Figure 10-190 Fault Analysis: add a card to Run-time Options

13. Now we can generate the JCL. Open the z/OS Project view, right-click the program, and select **Generate JCL → For Compile Link Go**, as shown in Figure 10-191.

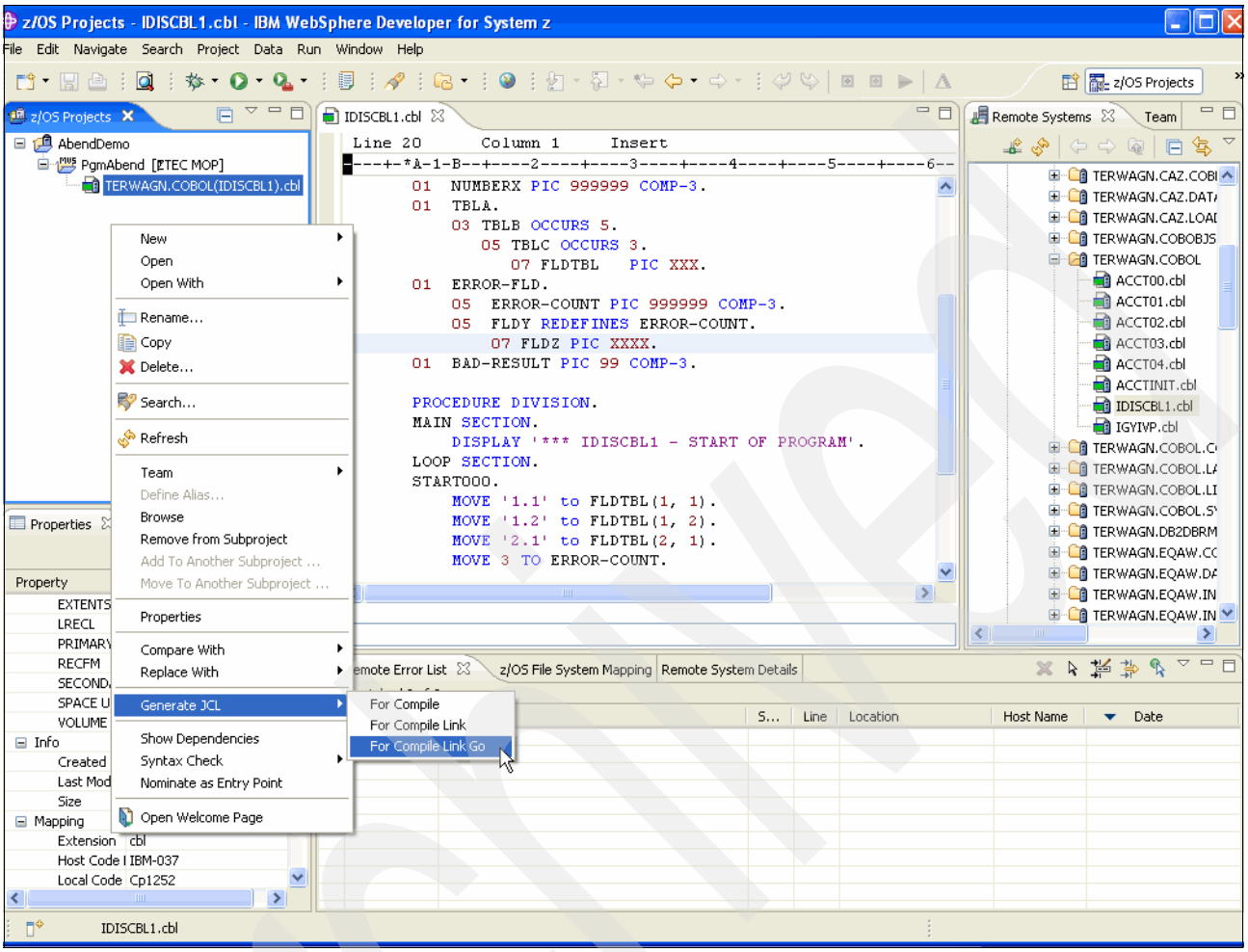


Figure 10-191 Fault Analysis: generate the JCL Compile Link Go

14. A pop-up window asks you for the job name, the PDS name to store the JCL in, and the JCL name. Use the values proposed by the wizard. Click **OK**, as shown in Figure 10-192.

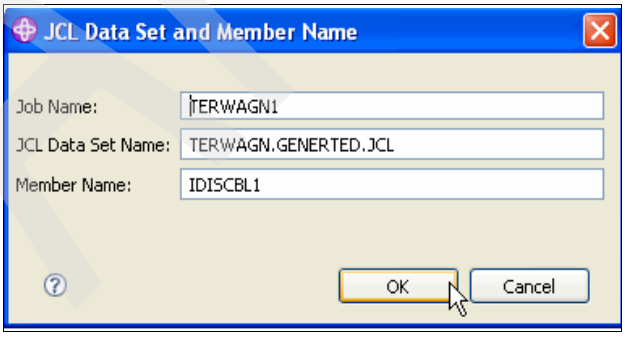


Figure 10-192 Fault Analysis: JCL generation pop-up window

15. When done, you should see the generated JCL in the subproject. You can double-click it to open it. Enter `Submit` in the command line at the bottom of the source window, press Enter, and pay attention to the JOBID returned, as shown in Figure 10-193, Figure 10-194, and Example 10-4.

Important: You should see four steps (see the Outline view):

- STP0000
- IDILANGX
- LKED
- GO

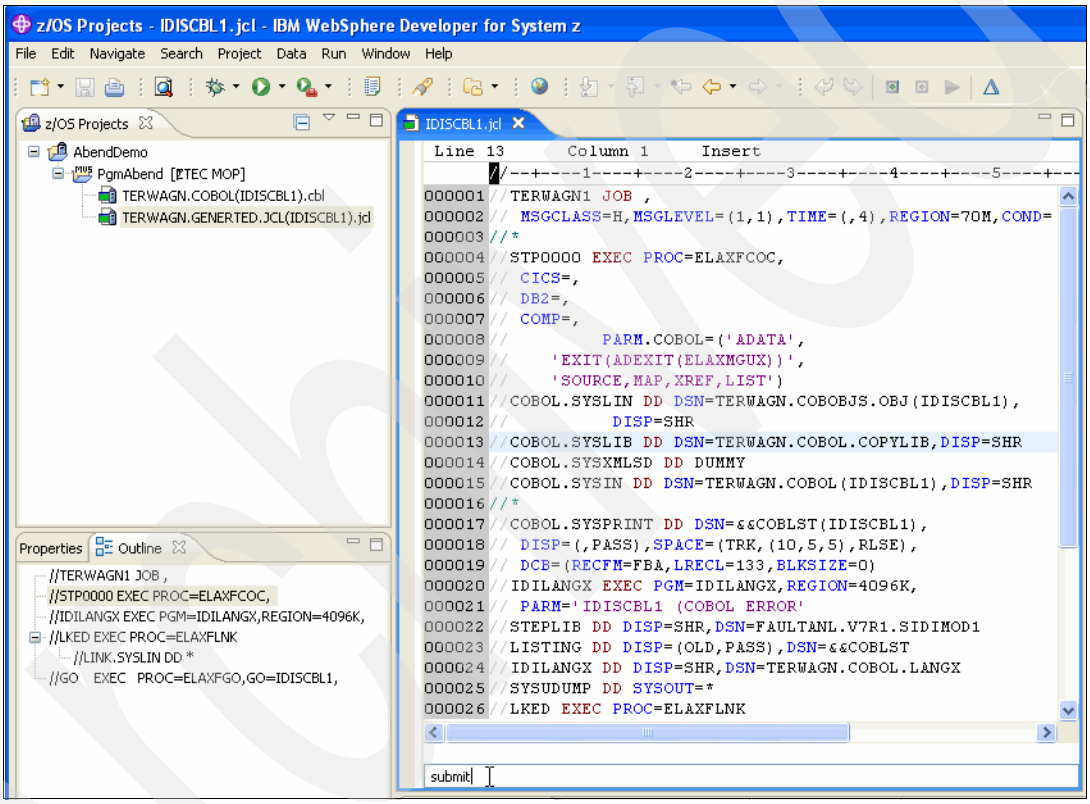


Figure 10-193 Fault Analysis: JCL generated - Compile Link Go - Submit job

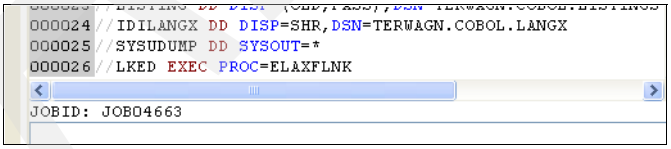


Figure 10-194 Fault Analysis: JOBID returned

Example 10-4 Fault Analysis: JCL Compile Link Go

```
//TERWAGN1 JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=(,4),REGION=70M,COND=(16,LT)
//*
//STP0000 EXEC PROC=ELAXFCOC,
// CICS=,
```

```
// DB2=,
// COMP=,
//         PARM.COBOL=('ADATA',
//    'EXIT(ADEXIT(ELAXMGUX))',
//    'SOURCE,MAP,XREF,LIST')
//COBOL.SYSLIN DD DSN=TERWAGN.COBOBJS.OBJ(IDISCBL1),
//          DISP=SHR
//COBOL.SYSLIB DD DSN=TERWAGN.COBOL.COPYLIB,DISP=SHR
//COBOL.SYSXMLSD DD DUMMY
//COBOL.SYSIN DD DSN=TERWAGN.COBOL(IDISCBL1),DISP=SHR
//*
//COBOL.SYSPRINT DD DSN=&&COBLST(IDISCBL1),
// DISP=(,PASS),SPACE=(TRK,(10,5,5),RLSE),
// DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//IDILANGX EXEC PGM=IDILANGX,REGION=4096K,
// PARM='IDISCBL1 (COBOL ERROR'
//STEPLIB DD DISP=SHR,DSN=FAULTANL.V7R1.SIDIMOD1
//LISTING DD DISP=(OLD,PASS),DSN=&&COBLST
//IDILANGX DD DISP=SHR,DSN=TERWAGN.COBOL.LANGX
//SYSUDUMP DD SYSOUT=*
//LKED EXEC PROC=ELAXFLNK
//LINK.OBJ0000 DD DSN=TERWAGN.COBOBJS.OBJ(IDISCBL1),DISP=SHR
//LINK.SYSLIN DD *
     INCLUDE OBJ0000
/*
//LINK.SYSLMOD   DD  DSN=TERWAGN.LOAD(IDISCBL1),DISP=SHR
//*
//******* ADDITIONAL JCL FOR LINK HERE ******
//GO    EXEC   PROC=ELAXFGO,GO=IDISCBL1,
//           LOADDSN=TERWAGN.LOAD
//IDILANGX DD DSN=TERWAGN.COBOL.LANGX,DISP=SHR
//
```

16. Open the Remote Systems view and expand **JES** → **My jobs**. You should see the submitted JOBID; if not, right-click **My jobs** and choose **Refresh**. When the JOBID is found, expand it to view the sysouts attached to it, as shown in Figure 10-195.

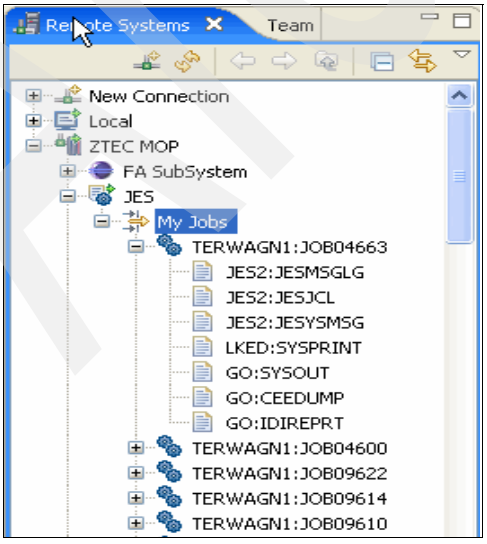


Figure 10-195 Fault Analysis: view the job

Archived

17. You should get seven sysouts. The last one (GO:IDIREPRT) is produced by the Fault Analyzer real-time analysis. Double-click it to view it. The first one contains the normal messages, where you should see a couple of messages from Fault Analyzer, such as the Fault ID associated with the abend. Remember it (in our example, F01426), for we will use it in the Fault list, as shown in Figure 10-196.

```
JOBNAME  STEPNAME PROCSTEP    RC    EXCP    CPU     SRB   CLOCK    SERV  PG    PAGE   SWAP     VIO SWAPS STEPNO
ERWAGN1 STP00000  COBOL       00     541    .00     .00     .00    1775   0       0      0       0     0      1
ERWAGN1           IDILANGX    00      59    .00     .00     .00     360   0       0      0       0     0      2
ERWAGN1 LKED      LINK        00     159    .00     .00     .00     547   0       0      0       0     0      3
DI0001I Fault Analyzer V7R1M0 (UK25564 2007/05/29) invoked by IDIXCEE using SYS1.PARMLIB.ZTOOPLEX.MOP(IDICNF00)
DI0002I Module IDISCBL1, program IDISCBL1, source line # 38: Abend S0C7 (Data Exception)
DI0003I Fault ID F01426 assigned in history file ADTCFG.IDI.HIST
F450I TERWAGN1 RUN GO - ABEND=S0C7 U0000 REASON=00000007
ERWAGN1 GO        RUN       *S0C7    726    .00     .00     .01    5456   0       0      0       0     0      4
ERWAGN1 ENDED.  NAME-                      TOTAL CPU TIME=    .00  TOTAL ELAPSED TIME=    .02
```

Figure 10-196 Fault Analyzer messages

18. Go back to the Fault Analyzer perspective. Refresh the history file by right-clicking the History file and selecting **Populate/Refresh History File**, as shown in Figure 10-197.

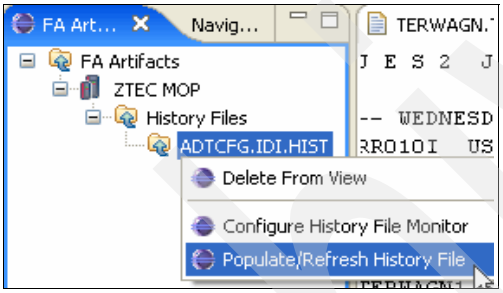


Figure 10-197 Fault Analyzer: Populate/Refresh History File

Archived

19. You now see the refreshed history file list. Notice Fault ID F01428 at the beginning of the list, as shown in Figure 10-198.

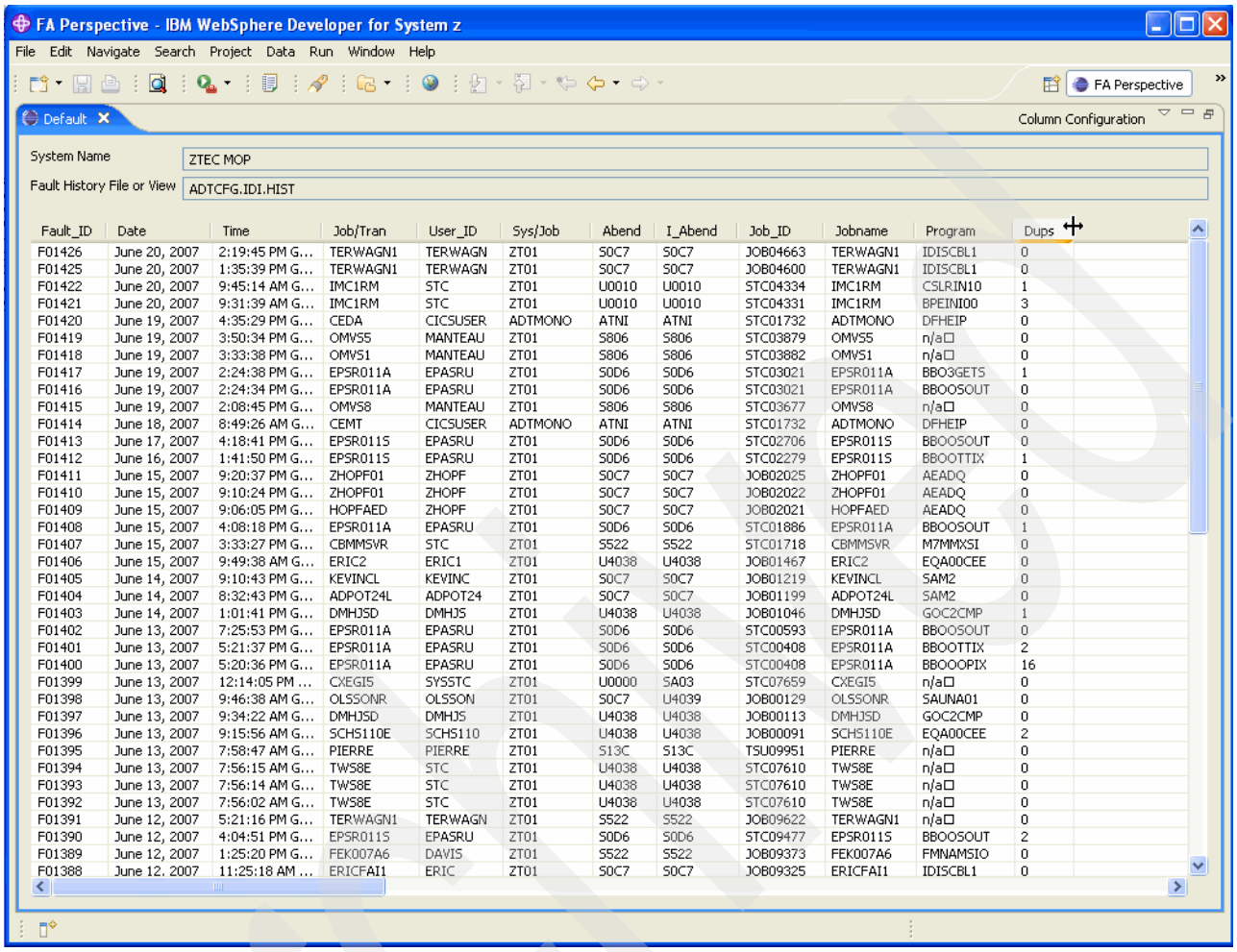


Figure 10-198 Fault Analyzer: history file list

20. By right-clicking the Fault ID row, you can choose to browse the report, as shown in Figure 10-199 on page 379.

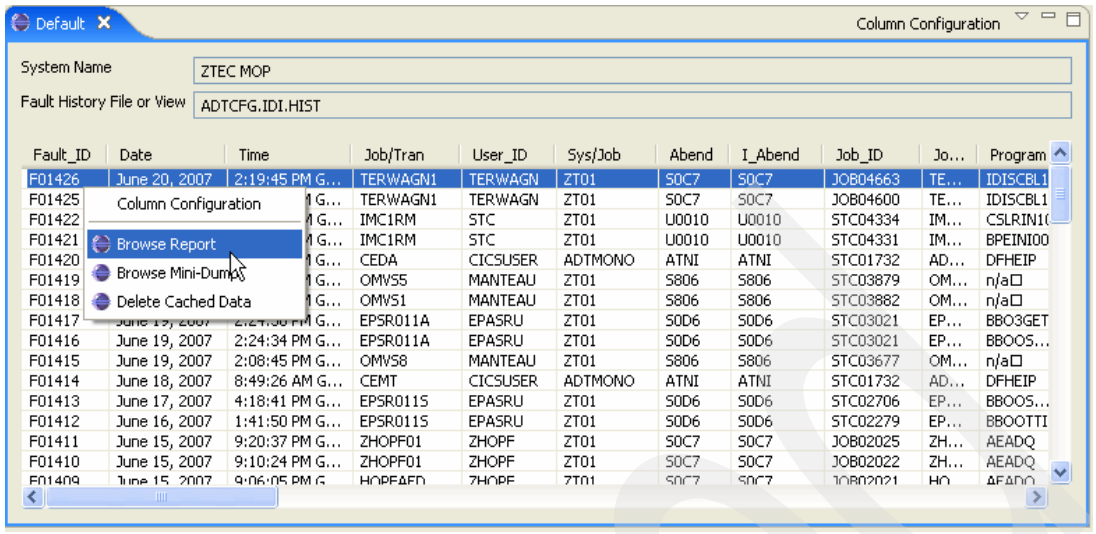


Figure 10-199 Fault Analyzer: browse the report

21. When viewing the report, you can navigate it by using the hyperlinks (in blue). There are several tabs at the bottom of the report as well. Clicking the program line number where the abend occurred (line 36 in the example) shows the program listing, as shown in Figure 10-200 and Figure 10-201 on page 380.

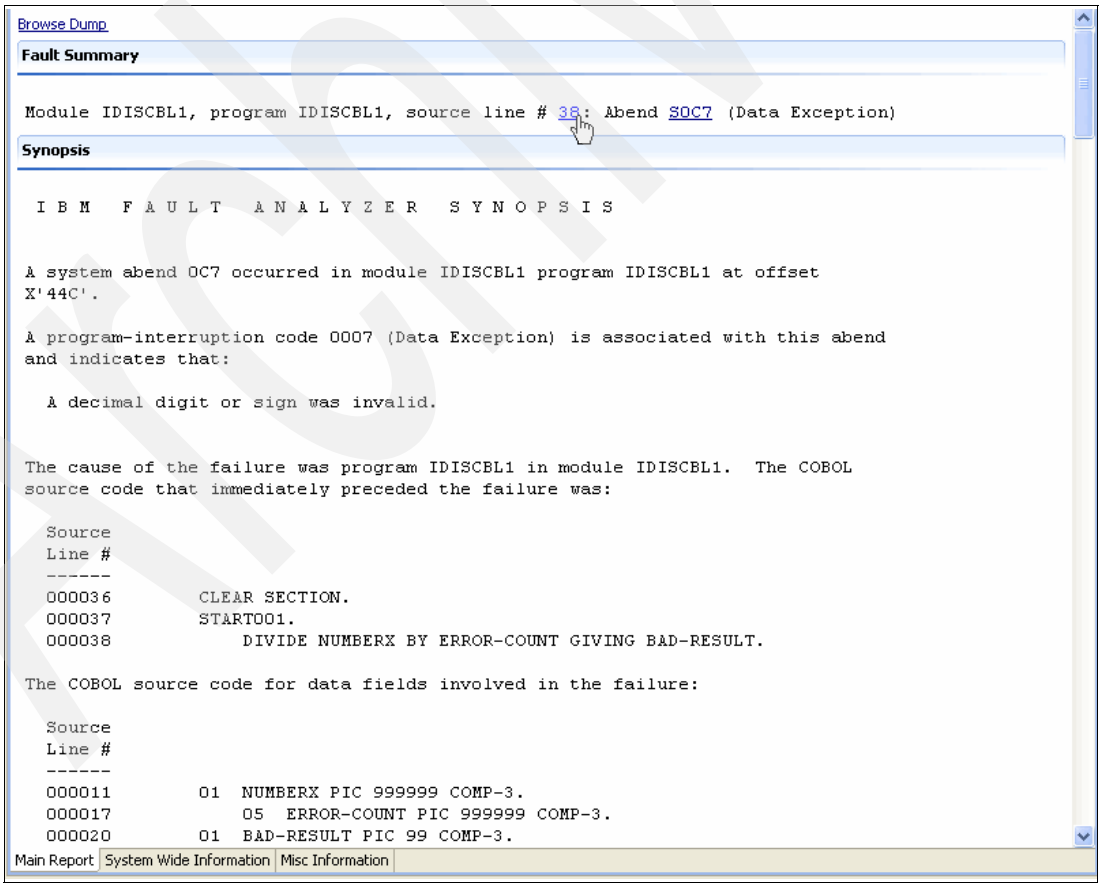


Figure 10-200 Fault Analyzer: browse report

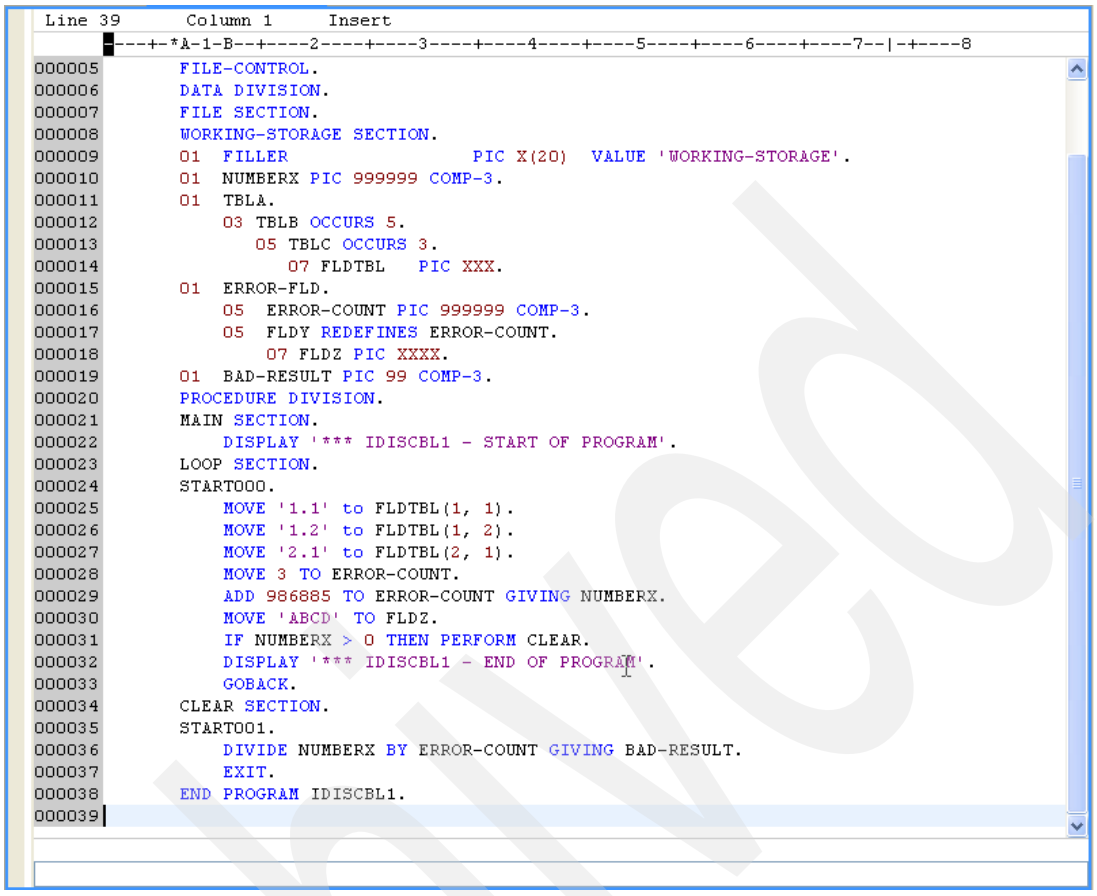


Figure 10-201 Fault Analyzer: program listing

10.8 File Manager plug-in for WD/z

This section describes how to use the File Manager plug-in for WebSphere Developer for System z. This plug-in, a Formatted Data Editor, provides editing access to sequential data sets, PDS members, and VSAM data sets of type KSDS and ESDS based on a data structure stored within a COBOL copybook or an IBM File Manager Template. The Formatted Data Editor takes the data structure(s) available in the copybook, include, or template, and applies them to the record of the data sets, providing a field type-sensitive edit session.

The functionality added by WebSphere Developer File Manager has been made available through an integration effort with IBM File Manager. IBM File Manager provides the data to WebSphere Developer File Manager. The Formatted Data Editor is closely based on the data set editor available in IBM File Manager.

Attention: This plug-in must be installed on the host and on the workstation. We assume that these installations are already done. We highly recommend that you download the *IBM File Manager for z/OS User's Guide and Reference*, SC19-1239 and familiarize yourself with the IBM File Manager's editor and template concepts.

Archived

10.8.1 IBM File Manager concepts

Many concepts from IBM File Manager have been reused for WebSphere Developer File Manager and understanding them is crucial to properly use COBOL copybooks and particularly IBM File Manager templates to properly format your data. The following items are areas that are especially important to understand or are areas where WebSphere Developer File Manager slightly deviates from IBM File Manager's implementation of the concepts.

Record layout A record layout defines the set of fields contained within a particular record set. All records matching this layout will be formatted with its set of fields at the offsets it describes. Each field is treated as a column in a table in the primary view, which is the table view. The record layout is built directly from data structure definitions located in COBOL copybooks.

Templates Record layouts are also obtained from IBM File Manager templates. In addition to describing record layouts, templates can also describe identification criteria and selection criteria, which can further alter the view of the data provided by the Formatted Data Editor. Templates obtain their data structures directly from COBOL copybooks. These types may be compiled into templates using IBM File Manager under ISPF. There is currently no support for requesting this compilation from within WebSphere Developer File Manager. For more information about templates, selection criteria, and identification criteria, refer to the *IBM File Manager for z/OS User's Guide and Reference*, SC19-1239.

Unselected records When a formatted data edit session is requested on a data set with a given template, WebSphere Developer File Manager will contact IBM File Manager, which will use the template to apply any selection criteria or identification criteria that exists in the template. Any records not meeting the selection criteria will be marked as unselected and will be represented as a grayed out line similar to the shadow line shown in IBM File Manager under ISPF. Selection criteria and identification criteria are not editable using WebSphere Developer File Manager, so these records will be unselected and therefore not editable until the template is modified so that they meet the selection criteria and the Formatted Data Editor is reopened.

Suppressed sets Templates can describe multiple record layouts. If they do, IBM File Manager will use record length or identification criteria to match the available record layouts to the data set's records (see the I*BM File Manager for z/OS User's Guide and Reference*, SC19-1239 for more information). Only one record layout may be displayed at once. Any records not matching the current record layout are grouped into suppressed sets. Unlike unselected record sets, suppressed record sets may be edited during a formatted data edit session without template modification. Editing suppressed record sets is described in 10.9, "Formatted editing" on page 384.

Excluded records Excluded records are one or more records that have been temporarily removed from the editor's view. Excluded records, unlike suppressed sets, are removed manually at the user's discretion and not by record layout constraints. This particular method of "data hiding" can be helpful when performing searches on data sets where the user knows that specific records can be

excluded. Record exclusion is similar to record selection, except exclusion is performed only during a specific editor session instead of for all editor sessions with the template, as is the case with selection criteria. Excluded records may easily be included back into the visible, editable set of records during the edit session. The process of excluding and including records is covered in 10.9, “Formatted editing” on page 384.

Shadow Line A shadow line is a spacer displayed within the editor that represents a group of adjacent records that have been unselected, suppressed, or excluded from the view by the user.

10.8.2 New file mappings

WebSphere Developer File Manager adds two new data set extension mappings to the WD/z Projects perspective: .tmpl and .tdat. .tmpl is used to define a data set as an IBM File Manager template. .tdat is used to set a data set as targeted for editing with the Formatted Data Editor. When the .tdat extension is added to a data set, the Formatted Data Editor will be defined as its default editor. All data set extensions can be set manually in the Remote Systems view in the z/OS Projects perspective to force them to be .tmpl or .tdat by right-clicking the data set and selecting **Properties**. Once the data set has the .tdat extension, it may be opened using the Formatted Data Editor with the data set’s Open With or Open options, or simply by double-clicking it, as shown in Figure 10-202 on page 383 and Figure 10-203 on page 383.

Note: By default, all data sets matching the pattern **.TEMPLATE and all of their members will have the extension .tmpl appended. Also by default, all data sets matching the pattern **.DATA will have the .tdat extension appended. If you wish for more patterns to be associated with these extensions, go to z/OS File System Mapping View to add them. VSAM clusters and sequential data sets must also have this extension to open them in the Formatted Data Editor even though the extension does not show up after being added using the Mappings pane in the Preferences window.

Archive

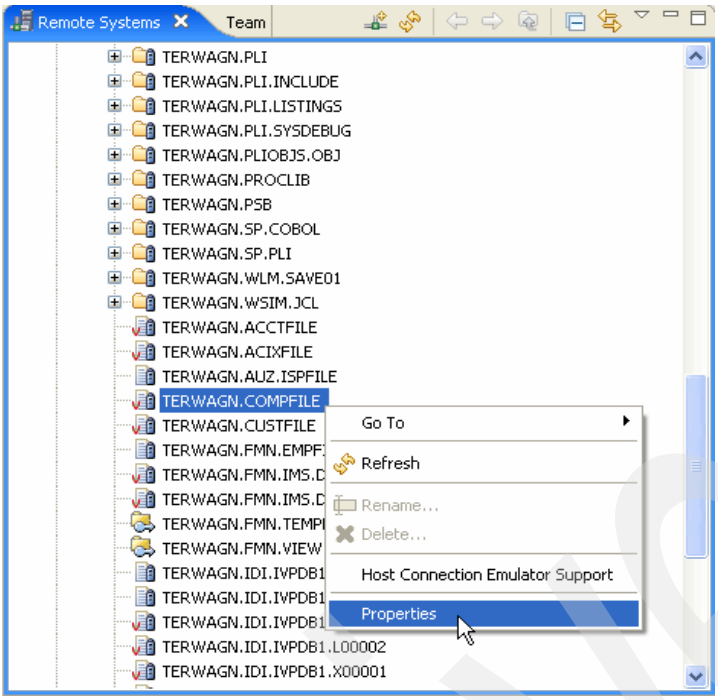


Figure 10-202 Data set properties

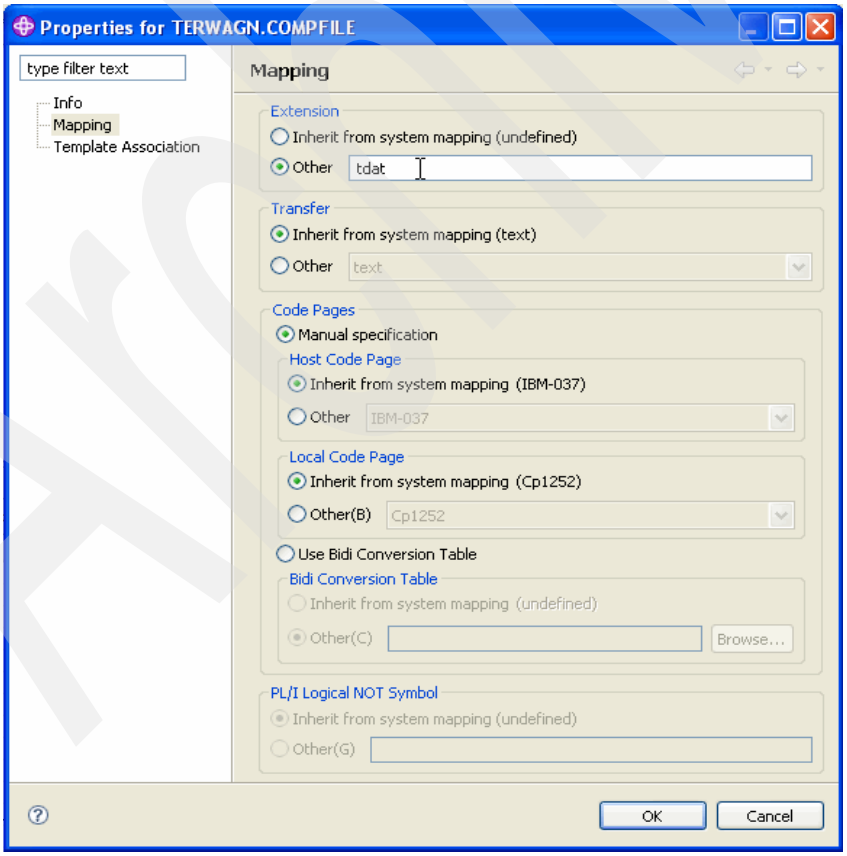


Figure 10-203 Data set mapping

10.8.3 Template association

In order to start the Formatted Data Editor with a template (or copybook) associated against a data set, you must go into the data set's properties and set the desired template to be associated. The template association properties window is only available for sequential data sets, VSAM clusters, and PDS members with the .tdat extension. To open the template association properties window, right click the target data set, select **Properties**, and select **Template Association** on the left sidebar of the window. Browse for the template or copybook in your data sets and select the desired one. Your can directly enter the name if you wish, as shown in Figure 10-204.

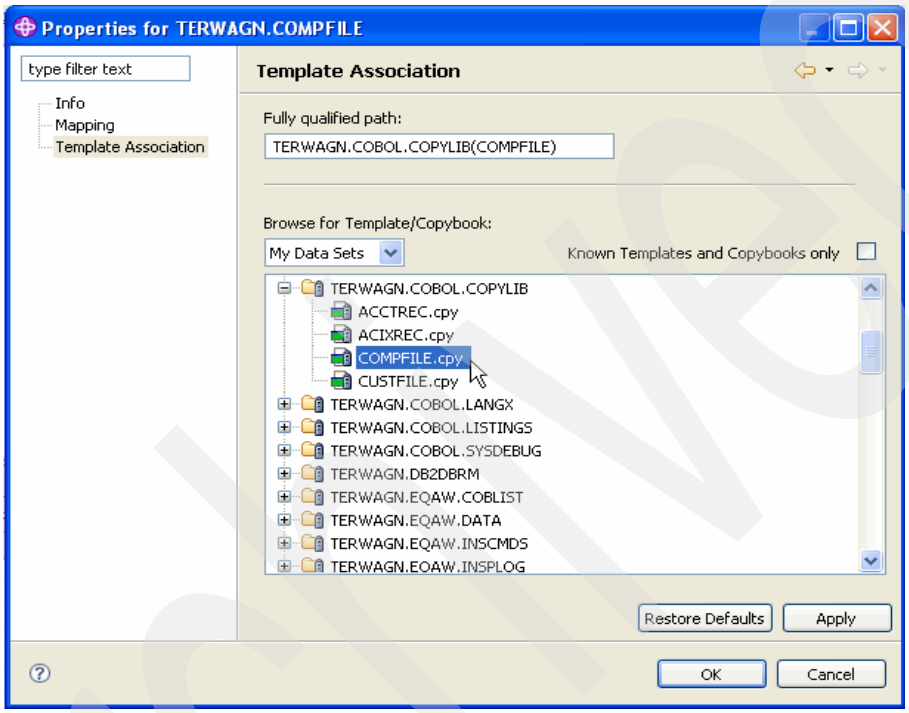


Figure 10-204 Template Association window

10.9 Formatted editing

Formatted editing refers to using a template while editing data within WebSphere Developer File Manager's Formatted Data Editor. Data sets can be opened in the editor with a template applied against them or with no template at all. If no template is specified, all records are treated as having single alphanumeric fields for the entire width of the record, as shown in Figure 10-205 on page 385.

When performing formatted edits using one of the previously mentioned template formats, the editor will display the following information for data sets:

- Field ID
- Field Widths
- Field Types
- Field Names

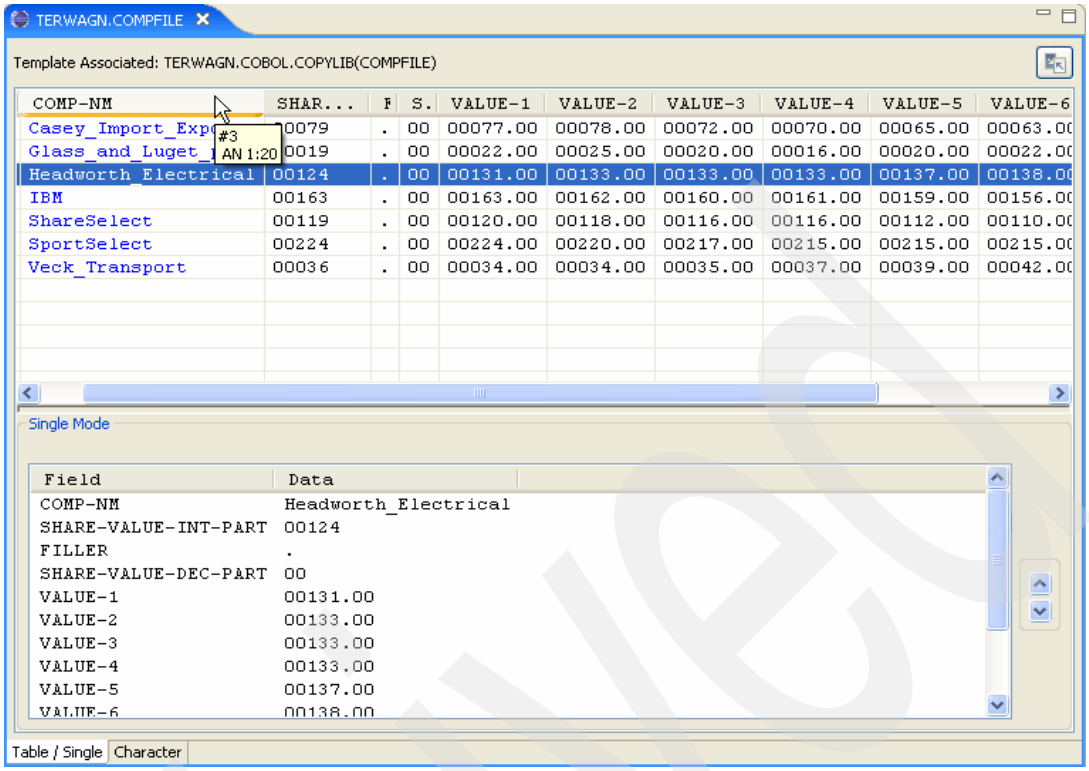


Figure 10-205 Formatted editing

10.9.1 Standard editing capabilities

When the Table and Character mode is being used, the following basic editing capabilities and functions are provided by the Formatted Data Editor:

- Find/Replace
- Cut/Copy/Paste (within the same data set)
- Row selection within the table used to perform a Cut, Copy, Delete, or Exclude operation on the selected group
- Add new records before or after any selected row

Archive

10.9.2 Find/Replace

While editing a file, right-click and select **Find/Replace**, as shown in Figure 10-206.

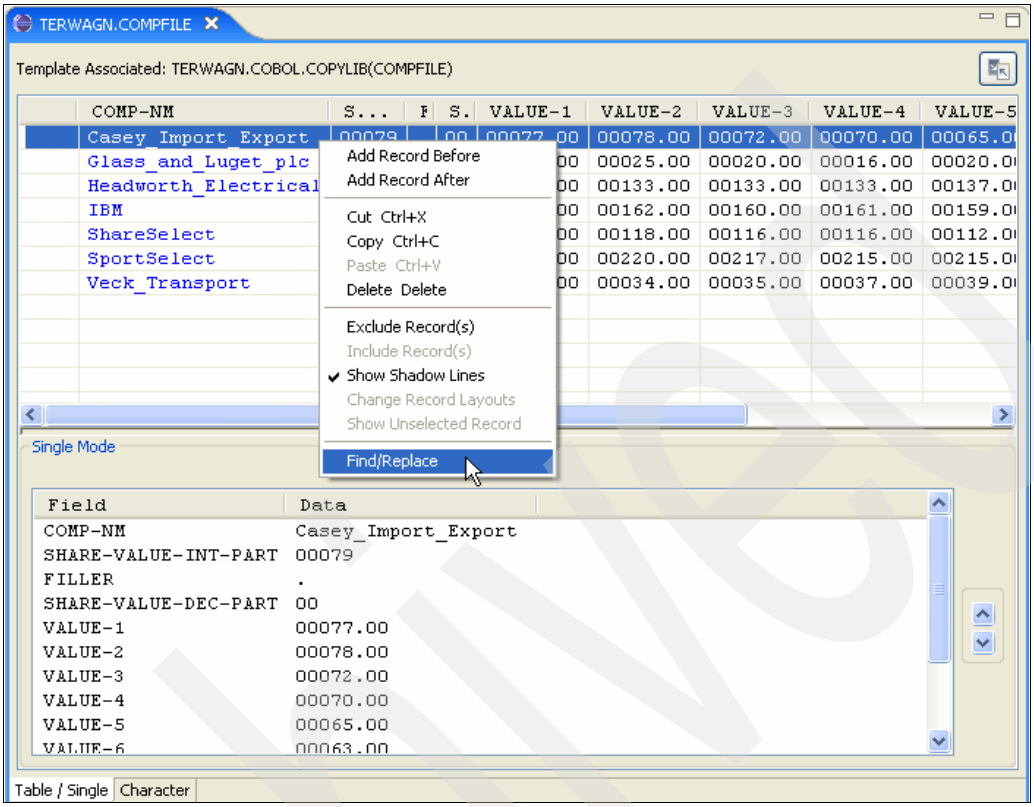


Figure 10-206 Find/Replace

The Find/Replace window gives you the ability to find and to replace values in selected columns, as shown in Figure 10-207 on page 387 and Figure 10-208 on page 387.

Archive

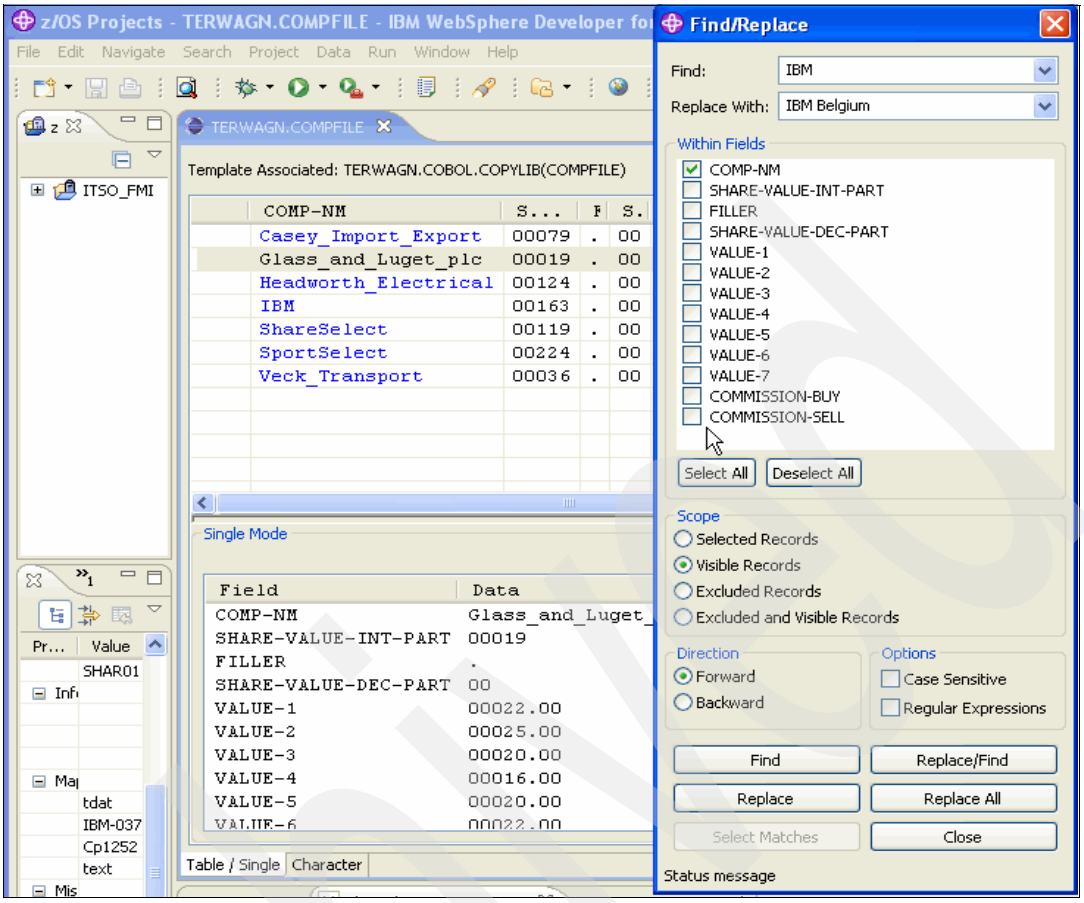


Figure 10-207 Find/Replace window

Headworth_Electrical	00124	.	00	00131.00	00133.00	00133.00	00133.00	00137.0
IBM Belgium	00163	.	00	00163.00	00162.00	00160.00	00161.00	00159.0
ShareSelect	00119	.	00	00120.00	00118.00	00116.00	00116.00	00112.0
SportSelect	00224	.	00	00224.00	00220.00	00217.00	00215.00	00215.0

Figure 10-208 Replace result

Archived

11

Software Configuration Management support in WebSphere Developer for System z V7

In this chapter, we present an overview of Software Configuration Management (SCM) features available in WebSphere Developer for System z (WD/z) V7.0 and provide guidelines for choosing the right solution for your requirements.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

11.1 Overview of WD/z SCM features

A good source code management system is an integral part of a software development life cycle for many years, and application development on z/OS platforms is no exception, regardless of the type of applications you are developing, whether more mature applications or newer applications based on newer languages such as Java and J2EE.

WebSphere Developer for System z V7.0 includes two distinctive SCM features designed for different users:

- Common Access Repository Manager (CARMA)
- Software Configuration Library Manager (SCLM) Developer Toolkit

If you have an existing repository for your project on the host and are looking for a way to access the repository from a WD/z environment, then CARMA provides a framework where you can extend the framework to access your repository as well as implement actions that are unique to your repository requirements. Section 11.2, “Overview of CARMA” on page 391 provides an overview of the CARMA framework as well as how to extend the framework to meet the requirements to access your repository.

If you are looking for an out-of-box SCM solution for your new project, are willing to migrate to SCLM from your current SCM system, or using SCLM in the ISPF environment and looking for ways to access the repository from WD/z, then SCLM Developer Toolkit is a suitable solution. SCLM DT allows you to access the SCLM repository from a WD/z environment and manage host-based resources (such as COBOL source files) as well as resources from a distributed environment (such as Java source files and Word documents). Section 11.3, “Overview of SCLM Developer Toolkit V2” on page 402 presents an overview of SCLM DT, including the management of resources in your workspace projects and the usage of SCLM view to manage SCLM resources remotely.

Archived

11.2 Overview of CARMA

Common Access Repository Manager (CARMA) is a library that provides a generic interface to z/OS software configuration managers (SCMs). Developers can build on top of CARMA by developing Repository Access Managers (RAMs) that plug into the CARMA environment. RAMs define how CARMA should communicate with various SCMs, that is, CARMA is an extensible framework that allows you to integrate any SCM into WD/z by developing RAMs for your repositories. Figure 11-1 shows the typical configuration areas of the z/OS Projects perspective with active CARMA RAMs.

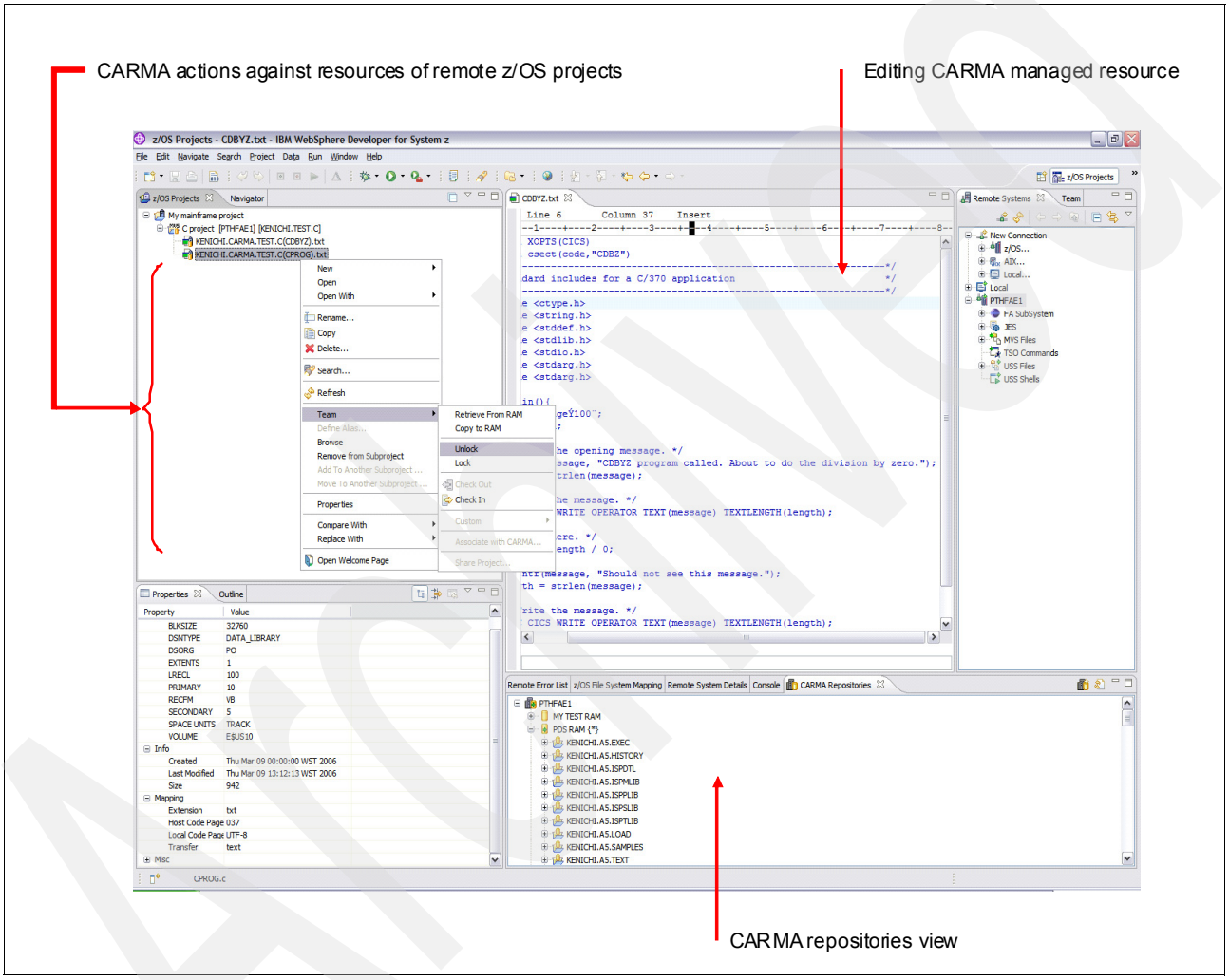


Figure 11-1 Overview of CARMA managed resource integration with WD/z

All active RAMs are shown in the CARMA repositories view (bottom right in the view) and actions against CARMA managed resources start from the view. Once you choose to work with a CARMA managed resource (for example, check out a member for modification), you will typically have an option to check out to a local project or a remote z/OS project (top left in the view). Once CARMA managed resources are checked out to a project, they are annotated with icons to indicate that they are CARMA managed resources. Actions implemented by the corresponding RAM are made available through the context menu of the Team menu. For example, the Unlock action is selected in the figure.

11.2.1 CARMA basics

Figure 11-2 illustrates a typical overview of a CARMA environment. In this diagram, four RAMs are installed in the CARMA environment, which allow you to access four different SCMs on the host from the client.

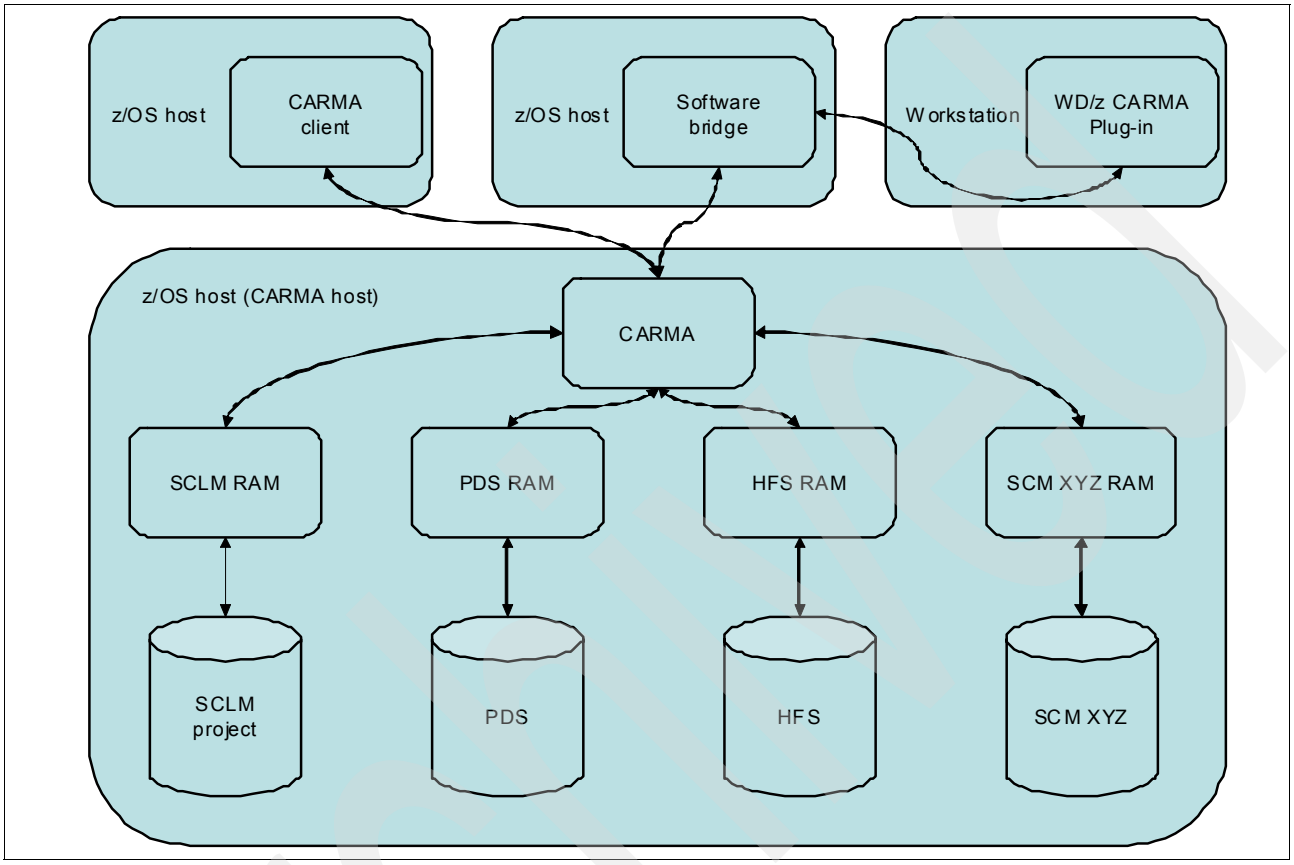


Figure 11-2 CARMA environment

CARMA currently supports the following sets of generic actions:

- Browse an SCM.
- Extract an SCM member.
- Create an SCM member.
- Update an SCM member.
- Get SCM member meta data.
- Update SCM member meta data.
- Copy a member to a PDS or SDS.
- Copy a member from a PDS or SDS.
- Delete a member or container.
- Lock, unlock, check in, or check out a member.

Although CARMA supports all of these actions, it is quite possible that a given SCM may not support one or more of these actions. CARMA provides a framework called the Custom Action Framework (CAF) for customizing the actions RAMs can perform. Using the CAF, you

are able to describe the differences of your RAM to the CARMA's standard APIs. You are able to:

- Define additional actions (custom actions)
- Disable standard actions
- Add additional parameters to standard actions
- Add additional return values to standard actions

You should refer to Chapter 4, "Customizing a RAM API using the CAF" in *Common Access Repository Manager Developer's Guide*, SC31-6914 for a detailed discussion of this topic.

Figure 11-3 shows the CARMA client (CARMA repositories view) in WD/z V7. The view can be opened by selecting **Window** → **Show View** → **Other** → **CARMA** → **CARMA views**. Descriptions of the items in this view are presented below. Most actions are made available to the users through this view. A typical usage of the view involves invoking actions against one of the RAMs installed on the host, which typically include an action to check out and make the resources available for modification in a local or remote project. Once resources are checked out or copied to a project, additional actions can be invoked against the resources through the Team action menu, which is available in the context menu.

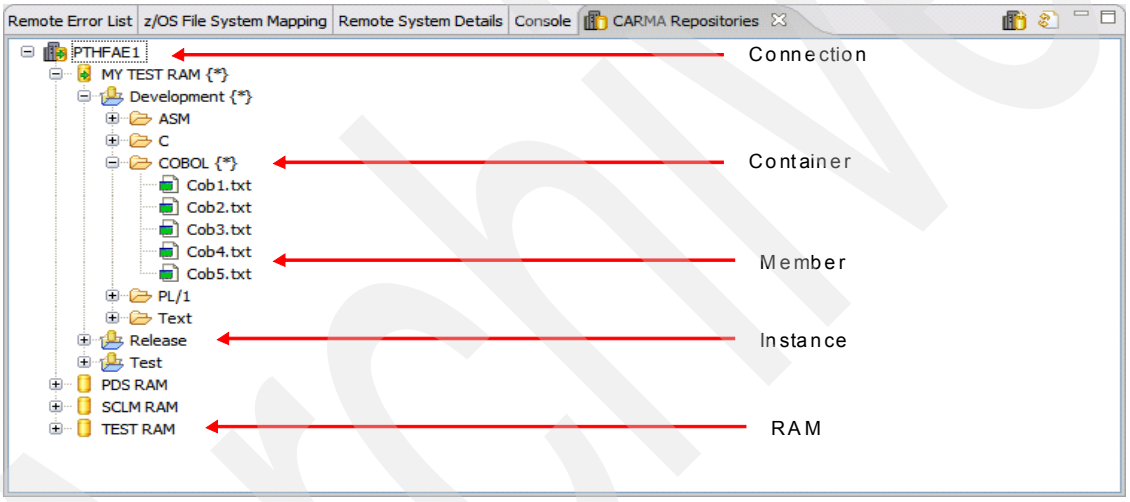


Figure 11-3 CARMA client (CARMA repositories view) in WD/z

- Connections

  Connections are the root elements in the CARMA repositories view. They display all the RAMS installed on the host. A new connection can be established through the Create a New CARMA Connection wizard, which allows you to create a new connection from an existing RSE connection (that is, a connection to a host is already established), or to establish a new RSE connection to a host and connect it to CARMA.

- RAMs

  RAMs encapsulate logics required to communicate with a SCM system on the host. All requests from clients are propagated to the repository through RAMs. It is possible to have multiple RAMs coexist on a host.

Archive

- Repository instances

  CARMA uses a repository instance to define a *project*, that is, a grouping of distinct programs and other content. It is appropriate to view instances as static entities on the client. In Figure 11-3 on page 393, MY TEST RAM has three instances, which represent different development stages during the project life cycle.

- Containers

  CARMA uses a container as an item that can hold a list of other CARMA resources. An equivalent of a container in CARMA would be a directory in HFS. Containers do not have to map to anything physically on the host, and can simply be used as a means of dividing up information into separate groups (for example, containers divide members by their language types in our example in Figure 11-3 on page 393).

- Members

  Members are the default resource type in CARMA. CARMA assumes members work in a similar manner to standard files, such as data set members and sequential files. Members hold content, such as a COBOL program or assembler routine.

  The standard client allows the members' contents to be extracted to local workspace projects where developers can then alter the members content and upload the changes back into CARMA.

CARMA out of box solutions

CARMA distribution includes a number of sample RAMs to help you develop your own RAM to meet your repository requirements and facilitates the communication between the client and the repository. However, these RAMs are provided as samples to help you develop your own RAM and they are not intended to be used in a production environment. A description of two sample RAMs are presented below.

- PDS RAM: The PDS RAM allows you to access a PDS associated with TSO users by leveraging ISPF services. In this case, the TSO/ISPF services are the SCM and the repository is the user's PDS.
- SCLM RAM: The Software Configuration Library Manager (SCLM) RAM allows you to access and edit software assets stored in SCLM projects. If the SCLM RAM is available on your CARMA host, then you can connect to an SCLM project using the CARMA plug-in.

11.2.2 Extending CARMA

A good starting to point to develop your own RAM for your repository is the sample RAM provided with the CARMA distribution. Figure 11-4 on page 395 summarizes all components related to the sample RAM. The source for the sample RAM can be found in WD4Z.V700.SCRASAMP(CRARAMSA) and a sample JCL to create the DLL is provided in WD4Z.V700.SCRASAMP(CRARAMCM).

Archived

The Custom Action Framework (CAF) allows you (the RAM developer) to describe your RAMs to CARMA. In particular, it allows you to describe the differences between your RAM and the standard RAM. For example, additional actions can be defined using the framework. CRADEF and CRASTRS VSAM files are the integral part of this framework and their purpose are described as follows (see Chapter 4, “Customizing a RAM API using the CAF” in *Common Access Repository Manager Developer’s Guide*, SC31-6914 for a more detailed description:

- CRADEF

  CRADEF stores all the language independent CAF data (that is, data that does not need to be translated from one locale to another). It contains records for each of the CAF object types, which include RAMs, actions, parameters, and return values. CARMA discovers the RAMs available by reading CRADEF.

  The following is a section in CRADEF where the sample RAM is defined to CARMA (TESTRAM with ID of R02000). If you are interested in developing your own RAM, define your RAM to CARMA here (for example, R03000 for MYTESTRAM):

```
<===+==>-1----+----2----+----3----+----4----+----5----+----6----+----7---
R00000  1.1     C       Z/OSV1R41.0     CRARPDS
R01000  1.0     C       1.0     1.0     CRARSCLM
R02000  1.0     C       Z/OSV1R41.0     TESTRAM
R03000  1.0     C       Z/OSV1R41.0     MYTESTRAM
```

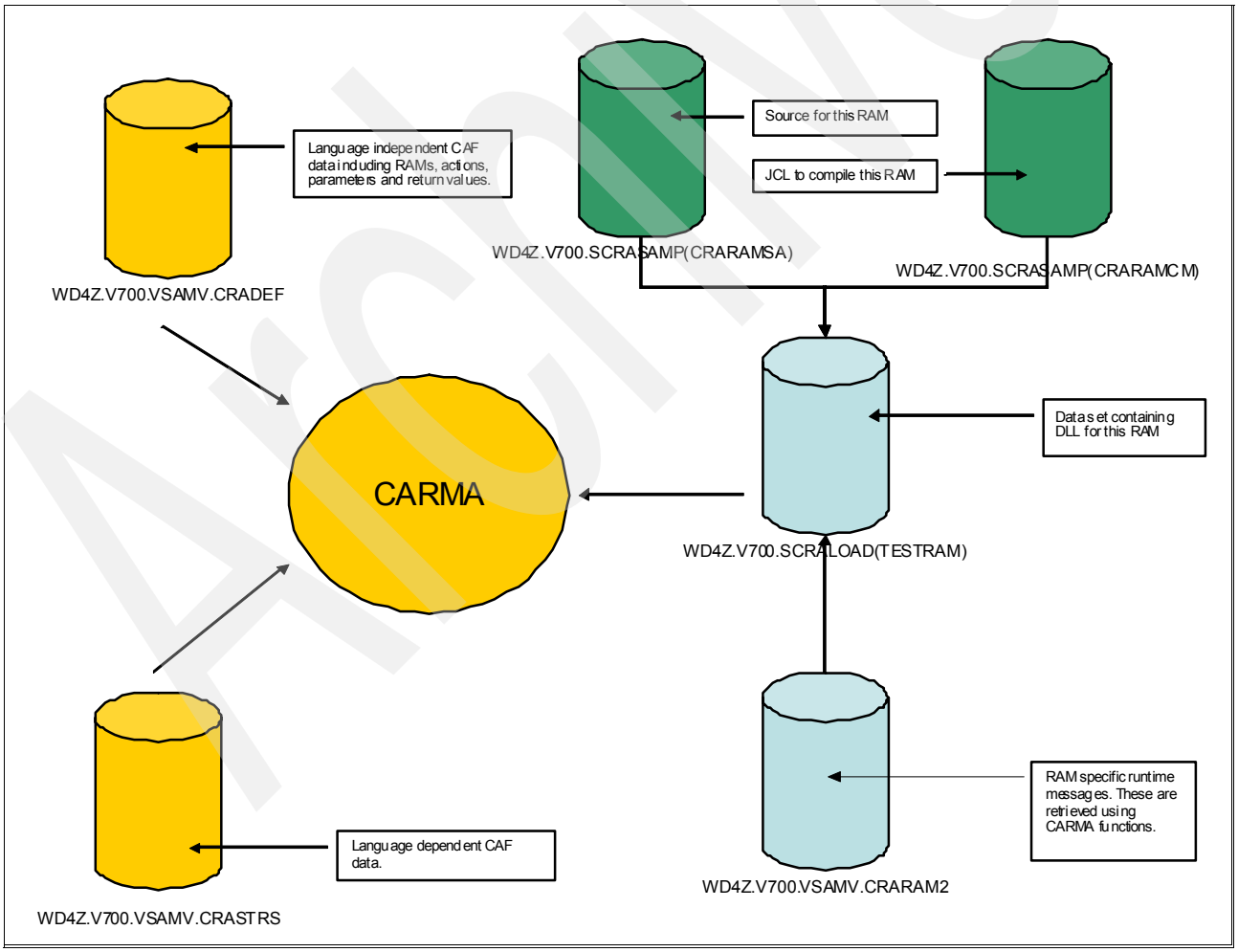


Figure 11-4 Components of the sample RAM

Archived

- CRASTRS

  CRASTRS stores all the language dependent CAF data. The languages are indexed within the VSAM cluster based on an eight character local and a five character code page. CARMA tries to determine if a user's preferred language is available for a given RAM by checking CRASTRS, which contains locale-specific information for the RAMs. When a client requests the list of available RAMs, CARMA will reference CRASTRS to attempt to compose a list of the RAMs that are available in the client's requested locale and code page.

  The following is a section in CRASTRS where the sample RAM description is defined to CARMA in English (locale of EN_US and code page of 0037). You will see similar entries for all languages supported:

```
<===+====1====+====2>---+----3----+----4----+----5----+----6----+----7----
EN_US   00037R00000  PDS RAM         Allows browsing of PDSs and browsing/
EN_US   00037R01000  SCLM RAM        Capable of performing all CARMA API a
EN_US   00037R02000  TEST RAM        Sample RAM to be used as a starting p
EN_US   00037R03000  MY TEST RAM     My RAM based on the sample provided.
```

Implementations of RAMs are presented to CARMA as a dynamic link library that exports entry points for all API functions that are implemented. Sample JCLs are provided to compile your RAM appropriately. WD4Z.V700.SCRASAMP(CRARAMCM) should be sufficient to compile the sample as provided (WD4Z.V700.SCRASAMP(CRARAMSA)). However, if you are intending to use CARMA provided APIs to implement your RAMs, you may need to use WD4Z.V700.SCRASAMP(CRARAMCS) instead. For example, if you are interested in using CARMA APIs to access language dependent messages stored in a VSAM file, then you need to use WD4Z.V700.SCRASAMP(CRARAMCS). See Chapter 3, "Developing a RAM', in *Common Access Repository Manager Developer's Guide*, SC31-6914 for a more detailed description.

Once you compile your RAM successfully, the data set containing your library must be added to STEPLIB DD in WD4Z.V700.SCRACLST(CRASUBMT) so that it can be discovered by the CARMA run time.

The sample RAM uses the WD4ZV700.VSAMV.CRARAM2 VSAM file to store its language dependent messages. You can modify the contents of this VSAM file, or define a new VSAM file and add appropriate DD statements, in WD4Z.V700.SCRACLST(CRASUBMT).

Sample code walkthrough

The sample RAM source provided in WD4Z.V700.SCRASAMP(CRARAMSA) defines the minimum set of functions that RAMs should implement. The description of functions are summarized in Table 11-1 through Table 11-16 on page 401. For a more comprehensive list of functions, refer to *Common Access Repository Manager Developer's Guide*, SC31-6914.

Table 11-1 initRAM: Initializes a RAM

Parameter	Input/Output	Description
Log_Func logFunc	Input	Pointer to log function.
FILE* log	Input	Pointer to log file.
int traceLev	Input	The trace level for the current session.
char locale[8]	Input	Tells CARMA the locale of the strings that should be returned to the client.
char codepage[5]	Input	Tells CARMA the code page of the strings that should be returned to the client.

Parameter	Input/Output	Description
char error[256]	Output	The error message to return if an error occurred.

Table 11-2 reset: Tells the RAM to reset itself to its initial state

Parameter	Input/Output	Description
char buffer[256]	Output	The error message to return if an error occurred.

Table 11-3 terminateRAM: Tells the RAM to clean up its environment

Parameter	Input/Output	Description
char error[256]	Output	The error message to return if an error occurred.

Table 11-4 getInstances: Retrieves the list of instances available in the SCM

Parameter	Input/Output	Description
Descriptor** records	Output	This will be allocated and filled with the IDs and names of instances.
int* numRecords	Output	The number of records that have been allocated and returned.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char filter[256]	Input	This can be passed from the client to filter out sets of instances.
char error[256]	Output	The error message to return if an error occurred.

Table 11-5 getMembers: Retrieves the list of members available within the specified instance

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance for which the members should be retrieved.
Descriptor** members	Output	This will be allocated and filled with the IDs and names of instances.
int* numRecords	Output	The number of records that have been allocated and returned in the array.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char filter[256]	Input	This can be passed from the client to filter out sets of members.
char error[256]	Output	The error message to return if an error occurred.

Archived

Table 11-6 isMemberContainer: Sets isContainer to true if the member is a container

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance the member is in.
char memberID[256]	Input	The member that may be a container.
int* isContainer	Output	Sets this to 1 if the member is a container, otherwise, is 0.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Table 11-7 getContainerContents: Retrieves the list of members within a container

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance the member is in.
char memberID[256]	Input	The container for which the members are being retrieved.
Descriptor** contents	Output	This will be allocated and filled with the IDs and names of the members within the container.
int* numMembers	Output	The number of member records that have been allocated and returned in the array.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char filter[256]	Input	This can be passed from the client to filter out sets of members.
char error[256]	Output	The error message to return if an error occurred.

Table 11-8 getAllMemberInfo: Retrieves all of a member's meta data

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance containing the member.
char memberID[256]	Input	The ID of the member whose meta data is being retrieved.
KeyValPair** meta data	Output	This should be allocated and filled with all the meta data key-value pairs for the specified member.
int* num	Output	The number of key-value pairs.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Archived

Table 11-9 getMemberInfo: Retrieves a specific piece of a member's meta data

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance containing the member.
char memberID[256]	Input	The ID of the member whose meta data is being retrieved.
char key[64]	Input	The key for the value to be returned.
char value[256]	Output	The requested value.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Table 11-10 updateMemberInfo: Updates a specific piece of a member's meta data

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance containing the member.
char memberID[256]	Input	The ID of the member whose meta data is being updated.
char key[64]	Input	The key for the value to be updated.
char value[256]	Input	The value to be set.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn,	Output	Used to reference an array of custom return values.
char error[256]	Output	Error message to return if error occurred.

Table 11-11 extractMember: Retrieves a member's contents

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance containing the member.
char memberID[256]	Input	The ID of the member being extracted.
char*** contents	Output	Will be allocated as a two-dimensional array to contain the member's contents.
int* lrecl	Output	The number of columns in the data set and array.
int* numRecords	Output	The number of records in the data set or the number of rows in the array.
char recFM[4]	Output	Will contain the data set's record format (FB or VB).
int* moreData	Output	Sets the value of the variable to which this points as 1 if the extraction should be called again because there is still more data to be extracted; otherwise, it should be 0.
int* nextRec	Input/output	Input: The member record where the RAM should begin extracting. Output: The first record in the data set that was not extracted.
void** params	Input	Pointer to an array of custom parameters.

Parameter	Input/Output	Description
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Table 11-12 putMember: Updates a member's contents or creates a new member if the specified member ID does not exist within the instance

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance containing the member.
char memberID[256]	Input	The ID of the member being updated/created.
char** contents	Input	Contains the new member contents.
int lrecl,	Input	The number of columns in the data set and array.
int* numRecords	Input/output	The number of records in the data set or the number of rows in the array.
char recFM[4]	Input	Contains the data set's record format (FB or VB).
int moreData	Input	Will be 1 if the client has more chunks of data to send; otherwise, it will be 0.
int nextRec	Input	The record in the data set to which the 0th record of the contents array maps.
int eof	Input	If 1, denotes that the last row of the away should mark the last row in the data set; otherwise, it should be 0.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Table 11-13 lock: Locks the member.

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance the member is in.
char memberID[256]	Input	The member to be locked.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Table 11-14 unlock: Unlocks the member

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance the member is in.
char memberID[256]	Input	The member to be unlocked.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Archive

Table 11-15 checkin: Check in the member. This only sets a flag. A putMember call is expected immediately after this call.

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance the member is in.
char memberID[256]	Input	The member to be checked in.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	The error message to return if an error occurred.

Table 11-16 checkout: Check out the member. This only sets a flag. A extractMember call is expected immediately after this call.

Parameter	Input/Output	Description
char instanceID[256]	Input	The instance the member is within.
char memberID[256]	Input	Member to be checked out.
void** params	Input	Pointer to an array of custom parameters.
void*** customReturn	Output	Used to reference an array of custom return values.
char error[256]	Output	Error message to return if error occurred.

Using CARMA APIs to manage runtime messages

CARMA provides a set of APIs to manage a set of language-dependent messages for your RAM. These messages are stored in a VSAM file and accessed using the APIs described in this section. The APIs makes your RAM ready to handle multiple locale and language settings.

These APIs are not used in the sample provided (WD4Z.V700.SCRASAMP(CRARAMSA)). To compile your RAM with these APIs, you need to use the sample JCL provided in WD4Z.V700.SCRASAMP(CRARAMCS). WD4Z.V700.SCRASAMP(CRAMSGH) is the header file containing the declarations of VSAM file interaction APIs.

Example 11-1 shows the steps required to initialize the CARMA message retrieval framework and its usage. You need to call `initializeVsamVsgs` to initialize to the user's preferred locale and language. Also, it takes the name of VSAM file containing the messages. The VSAM file name should be specified through the DD card (*`message_file`* with the value of "DD:MYMSG" in the example below), which should be specified through WD4Z.V700.SCRACLST(CRASUBMT). After the call, you can call `getErrorString`, `getLogString`, and `getFunctionString` to retrieve messages from the VSAM file.

Example 11-1 Example of using CARMA APIs to access language dependent messages

```
#define message_file       "DD:MYMSG"
int initRAM(Log_Func logFunc, FILE* log, int traceLev, char locale[8],
            char codepage[5], char error[256]){
   char locale_str[13];
   int rc;
   char buffer[256];

   ...

   log_wrapper(0, "Entering initRAM", 16);
```

```
   /* Setup local and language setting for message VSAM */
   memcpy(locale_str, locale, 8);
   memcpy(&(locale_str[8]), codepage, 5);

   /* set up VSAM message retrieval facility */
   rc = initializeVsamMsgs(locale_str, message_file);

   ...

   /* Retrieve messages using CARMA APIs */
   getErrorString(buffer, 615, 256);
   getLogString(buffer,  616, 256);
   getFunctionString(buffer,  617, 256);
}
```

11.3 Overview of SCLM Developer Toolkit V2

SCLM Developer Toolkit V2 is a feature of WD/z V7.1[1], which allows you to use Software Configuration Library Manager (SCLM) as a repository to manage resources associated with a project. It provides SCM services required for the host based development, such as COBOL application development, as well as application development in the distributed environments using Java and J2EE.

This section assumes that you are familiar with the basic SCLM terminology. If you are not familiar with SCLM, we highly recommend *Getting Started with SCLM: A Practical Guide to Setting Up and Using SCLM* or refer to the product Web site at:

http://www-306.ibm.com/software/awdtools/sclmsuite/sclm/

Figure 11-5 on page 403 shows the SCLM Developer Toolkit areas of interest.

[1] SCLM Developer Toolkit V2 is also available as part of SCLM Advanced Edition for z/OS.

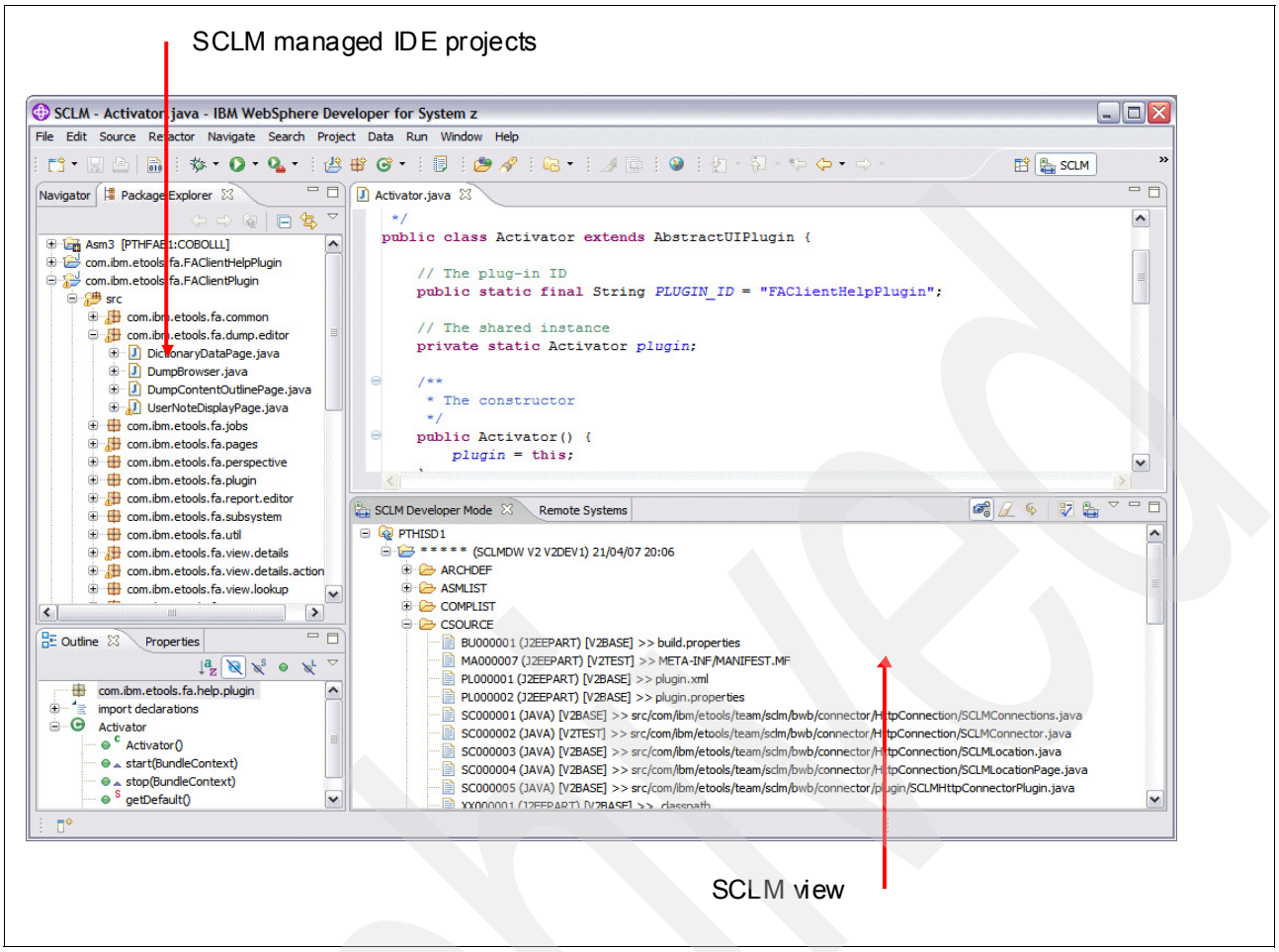


Figure 11-5 Overview of SCLM DT in WD/z

In summary, SCLM Developer Toolkit provides an out-of-box solution that allows you to manage resources for your host-based applications (such as COBOL programs) as well as resources of distributed applications (such as Java/J2EE projects) in a single repository using SCLM. In addition to providing basic support for managing resources of distributed applications, it also provides a framework for building and deploying J2EE applications.

Figure 11-5 shows a typical configuration a WD/z perspective, where SCLM managed projects are shown in the Package Explorer view (on the top left), and SCLM resources are shown in the SCLM view (on the bottom right).

11.3.1 IDE based Java/J2EE application development

Refer to Chapter 21, “SCLM Developer Toolkit for Java applications”, in *Getting Started with SCLM: A Practical Guide to Setting Up and Using SCLM* for a detailed discussion of this section’s topic.

SCLM DT provides the source code management capability when you are working with a project in your workspace based on the Eclipse team framework. In addition to providing the common set of SCM functions, such as check in and check out and synchronize, it provides an advanced build and deployment framework, which allows you to build and deploy your Java/J2EE applications under the control of SCLM.

Once you register SCLM as the repository provider of your project in your workspace (select **Team** → **Share Project** to invoke the share project wizard, select **SCLM**, and specify your information), most SCLM functions are made available for the resources of your project from the context menu under the Team menu (right-click a resource and select the items under the Team menu). Table 11-17 summarizes a set of actions that are available for your resources.

Table 11-17 Summary of actions available for the SCLM managed project

Action	Description
Logon	Authenticates with the server. Presents a dialog to specify the login information as well as provides an opportunity to change the password if required.
Register project to SCLM	Register and select SCLM to manage resources in a project in an Eclipse environment. Once the project is registered, all the action described here become available for resources in the project.
Un-map project from SCLM	When you finished working with a project being managed by SCLM, you can un-map the project (or unregister SCLM as a repository provider) so that all SCLM specific actions will not be available against the resource of the project.
SCLM Import	Imports members in SCLM to the selected project. You can specify the members to import from SCLM using various filters or ARCHDEFs. By importing members from SCLM, you get local copies of the members in the selected project. However, members are not locked on the host.
Add to SCLM	Adds additional resources to SCLM.
Check out Member	Places an exclusive lock against the selected member and makes it available for editing in the local project. The latest copy of the member is copied from the repository by default.
Check in Member	Once you are finished making all the required changes, the contents of the member should be saved back in the repository. This action saves the contents of a local resource in the repository and removes the exclusive lock placed on the member through the check out action.
Cancel Edit / Un-lock	This action allows you to remove the exclusive lock being placed for the selected member.
Delete Member from SCLM	This action deletes the member from the repository.
Get SCLM Status	Retrieves the latest information from the repository for the selected member.
Synchronize Project with SCLM	Compares the status of all members in the selected project and the members in an SCLM repository. If status differences exist, then they are reported in the synchronize view.
Version Information	Displays version information.
Operations Log	Allows you to browse and save an operational log of all actions being performed.
Local Info	Displays the cached SCLM information about the selected resources.
Update Authorization Code	Allows you to update the authorization code of the selected member.
Update Member Details	Allows you to change the language type and change code for the selected member.

Action	Description
Upload Jar Files	Allows you to upload local files to a directory under UNIX System Services. This feature can be useful when moving all dependent JAR files from PC to the host.
View Cached Project Information	DT caches the project information. This action allows you to view the cached project information.
Update Project Information	Retrieves the latest project information from the host and refreshes the cached project information.
Generate Database Content Reports	This function allows you to select various criteria in order to extract information from SCLM's meta data tables.
Compare with latest version	Allows you to compare the contents of the selected local file and the latest version in the repository.
Compare with different version	Allows you to compare the contents of the selected local file and different versions of the member in the repository (the versioning feature must be turned on for this to work correctly).
Replace with latest from SCLM	Replaces the contents of selected resource with the contents of the corresponding resource in the repository.

Most of the common SCM functions are provided by SCLM DT as well as SCLM specific actions, such as the Promote action through the context menu. A detailed explanation of each action is presented in the online help, which is accessible by pressing the F1 key while selecting an item in the context menu.

Managing workspace projects using SCLM DT

One of the key features of SCLM DT is its ability to manage and build Java/J2EE projects using SCLM as a repository on a z/OS host. It provides a framework to manage the long name / short name translation[2], as well as managing the code page differences.

Figure 11-6 on page 406 illustrates a typical configuration of Java/J2EE projects that are managed by SCLM. Each member in a SCLM repository has a language type (for example, Java, COBOL, and Text) so that SCLM knows how to build them. A set of related members in a SCLM repository (that is, resources of a project in your workspace) are described by ARCHDEF[3] files. When you are migrating the resources of your project to a SCLM repository, you need to specify a name of ARCHDEF file so that SCLM is able to keep track of related resources for your project. Using the ARCHDEF file for your project, you are able to build your project.

IBM makes the following recommendations for storing your Java/J2EE projects in SCLM:

- All related Java/J2EE projects are stored in SCLM projects, so that resources of dependent projects for a project can be found during the build process.
- Resources of different Java/J2EE projects should be stored under different SCLM types to avoid the naming conflicts during the long name / short name translation, and the ease of management.

[2] File names of distributed platforms must be translated because data sets and their member names are more restrictive on z/OS platforms.
[3] Architectural definition files.

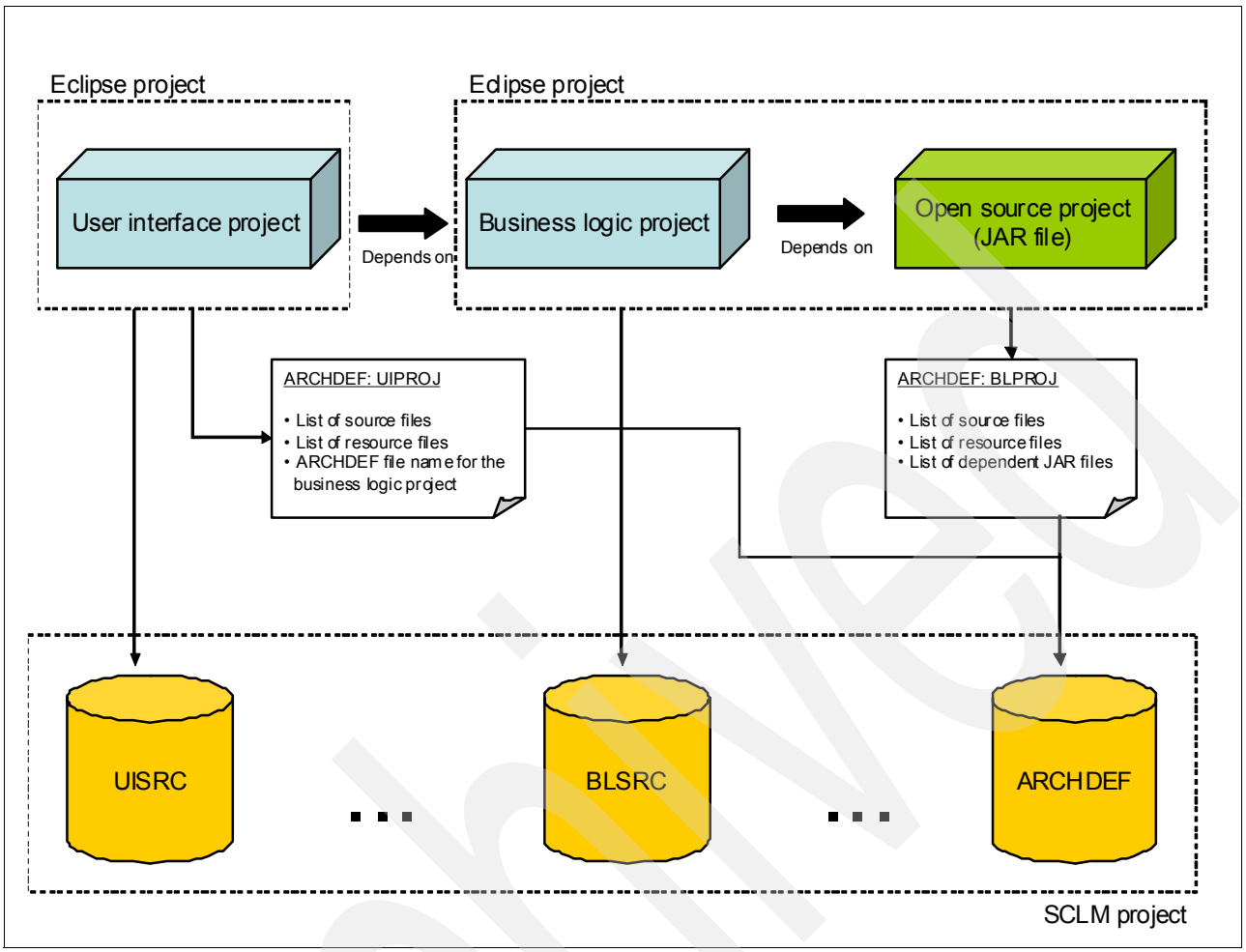


Figure 11-6 Typical configuration of Java projects managed by SCLM

Figure 11-7 on page 407 illustrates an overview of SCLM build process for Java/J2EE projects. The build framework of SCLM DT is built using ANT[4], which is highly customized to meet your requirements, and SCLM DT enables seamless integration of SCLM and the ANT build process.

An ARCHDEF file for your project describes all the resources of your project and tells SCLM what members should be considered for building when requested. SCLM build process compares the build date and the modified date of your resources to decide which members of your project should be built. SCLM only builds members when necessary unless it is told to build the entire project (*forced build mode*, which can be specified through the client).

A special entry (using the SYNC keyword; refer to *Getting Started with SCLM: A Practical Guide to Setting Up and Using SCLM* for further details) in your ARCHDEF file identifies an ANT build script as well as parameters required by the build script. Identified script is copied to a directory under UNIX System Services along with additional parameters used by the script[5]. The build process also copies the resources of your project to a directory under UNIX System Services to invoke the build process. Once the build process is complete and successful, accounting information of your resources are updated, and build process

[4] The APACHE ANT is an open source framework for building projects. Refer to the Web site found at http://ant.apache.org/ for further details about ANT.

[5] Additional parameters are overlaid at the beginning of the build script.

generated artifacts are copied back to the repository (for example, CLASS files generated by compiling Java source files).

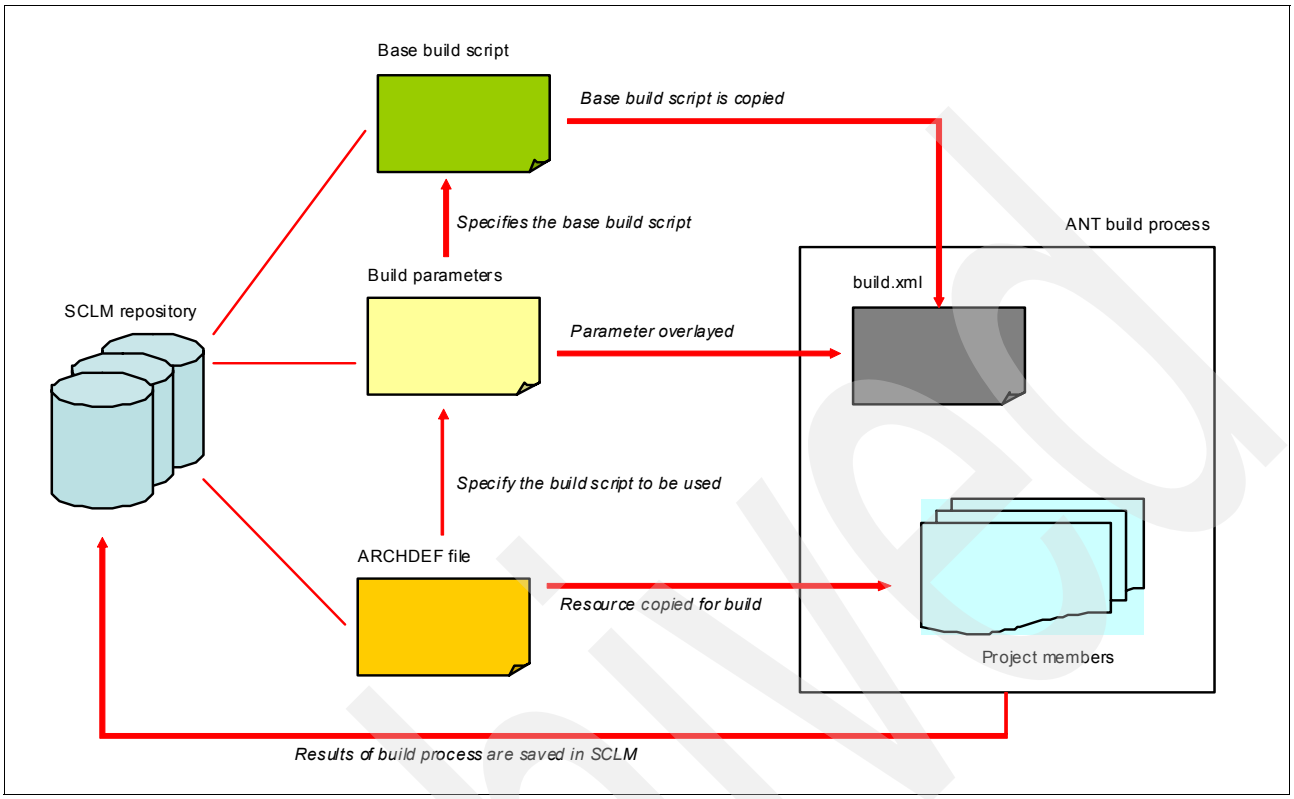


Figure 11-7 Overview of the SCLM build process

11.3.2 Using SCLM view

Refer to Chapter 20, “SCLM DT for mainframe development”, in *Getting Started with SCLM: A Practical Guide to Setting Up and Using SCLM* for a detailed discussion about this topic.

The SCLM view can be considered a portal to SCLM repositories (You can open this view by selecting **Window** → **Open View** → **Others** → **IBM SCLM Developer Toolkit** → **SCLM Repository view**). It allows you to browse and work with resources in SCLM repositories.

Archive

Most of actions summarized in Table 11-17 on page 404 are also available in the view through the context menu, as shown in Figure 11-8.

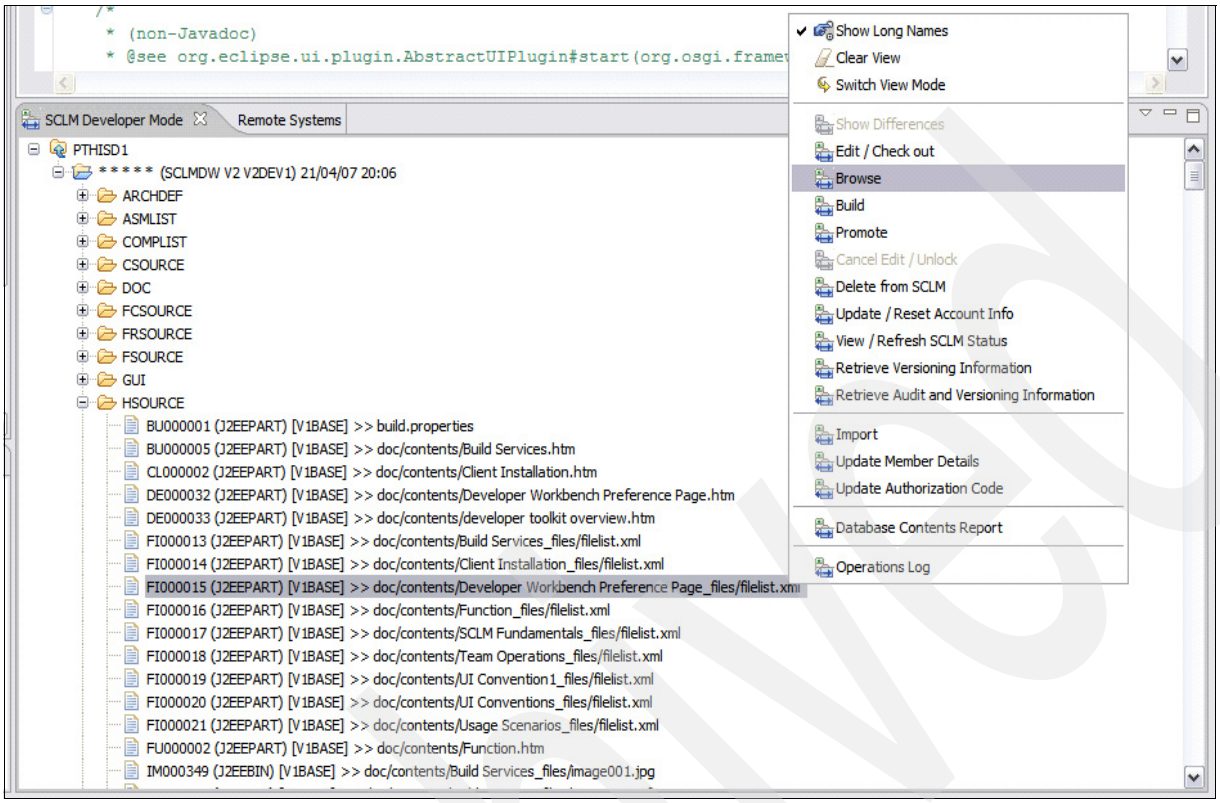


Figure 11-8 SCLM view overview

In addition to making these actions available from the view, additional actions are made available through the view, which include:

- Database content report
- Extensive audit and versioning support

As well as providing extensive support for invoking SCLM specific actions, this view allows you to edit members without copying them to a project in your workspace.[6] Using this feature and other actions, you are able to browse and make changes to the members in your repository as you would in the SCLM interface in the ISPF environment.

When you are working with resources in the repository through the SCLM view, it is important to specify your preferred file extensions for the language types supported in your repository. For example, members with the language type of JAVA should have an extension of .java, where members with the language type of COBOL should have an extension of .cbl so that SCLM DT is able to invoke your favorite editors (for example, invoke the Java editor for Java source files and the ELPEX editor for COBOL source files). Your language-extension preferences can be set in the SCLM DT preference page by selecting **Window** → **Preferences** → **Team** → **SCLM Preferences** → **Language Extensions**).

[6] SCLM DT copies the contents of a selected member to a local project and manages it there.

Since the view allows you to browse repositories efficiently and be able to specify different filters to populate the view, it allows you to work with your SCLM repository efficiently. When you are working with the view, you are able to filter the view using the following filters:

- Filter options: Allows you to specify the combination of following filters to select a set of members.
  - Group: Filters members by their development groups (for example, DEV1, TEST, and RELEASE).
  - Type: Filters members by their types (for example, SOURCE and JAVASRC).
  - Language: Filters members by their language types (for example, JAVA and COBOL).
  - Member: Filters members by their member name patterns (for example, A* for members starting with ‘A’).
  - Authorization code: Filters members by their authorization codes.
  - Change code: Filters members by their change codes.
- Filter by ARCHDEF options: Allows you to specify an ARCHDEF file that contains a set of members.
  - ARCHDEF name: Name of ARCHDEF file to use.
  - ARCHDEF group: Development group where the ARCHDEF file should be found.
  - ARCHDEF type: Type of ARCHDEF where it should be found.

Archived

Archived

12

Sample implementation of a CARMA RAM

In this chapter, we demonstrate, with a simple example, the steps you take to implement a CARMA RAM.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

12.1 Motivation

As we explained before, SCLM DT is a good decision if you want an out of the box SCM solution. But when you look at today's SCM host installations, in most cases you do not find an out of the box installation. In many cases, the SCM customer has implemented a specific user interface and often a repository system has been installed to manage the artifacts of the application development process. This repository system may be an IBM product, a third-party product, or the result of in-house development.

The repository is the central access system of the artifacts in the application development process. This usually means that the repository also controls the access to an SCM system (for example, SCLM) so that the application developers do not use the user interface of the SCM system.

There are various reasons for controlling the application development process with a repository:

- A repository can register artifact meta data information in the application development process. You can define which meta data you want to register in your repository.
- A repository provides better information retrieval facilities than any SCM system. Moreover, a repository is best suited to managing and reproducing the relationships between the artifacts of your application development process.
- Based on the relationship information, a repository usually provides a very comfortable navigation facility for all your members. Access to the artifacts of the application development process through the repository navigation facility is more natural and corresponds more to practical requests than access through data sets, such as the native SCLM installation implies.

We think WD/z users have a considerable need for a CARMA RAM implementation based on a host repository. Therefore, we want to illustrate the RAM development based on such a structure. For reasons of simplification, we decide to implement a very simple application development model based on DB2 tables. In our example, the underlying SCM system will be SCLM. Of course, you could adapt another SCM system in the same way.

12.2 Modeling considerations

One of the first steps you have to take when implementing a CARMA RAM is to map your SCM structure to CARMA concepts. Normally, there are several alternative ways to present your SCM structure to the users. You have to decide which representation best fits the needs of the SCM users and then discuss this representation with your users before you implement the CARMA RAM.

12.2.1 SCM data model

In our case, the SCM structure is project-based. The simplified SCM data model looks like Figure 12-1 on page 413.

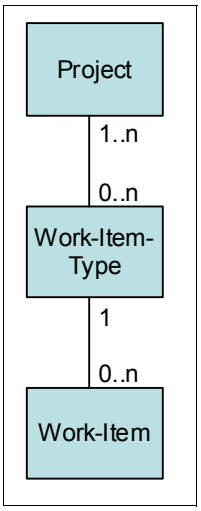


Figure 12-1 ER model of the CARMA example

The user works within a certain development project. Within this project he can select several work item types (for example, project protocol, COBOL program, copybook, and so on). A work item type groups the respective work items of the same type, that is, the documents or artifacts on which the user is actually working.

The SCM model is technically implemented by DB2 tables. The work items reside in TSO PO data sets, which are SCLM controlled. The access from a DB2 table member entry to a physical PO member is done based on certain attributes (SCLM_TYPE and SCLM_NAME) in the DB2 work item type table.

The structure of the SCM DB2 tables and attributes are shown in Table 12-1 through Table 12-4 on page 414.

Table 12-1 Structure of SCM DB2 tables

Project		
Attribute	**Data type**	**Key**
ID	integer	primary
Name	char(8)	
Short_Desc	varchar(50)	
Manager	char(16)	
State	char(16)	
SCLM_Group	char(8)	

Table 12-2 Attributes of the DB2 table PROJECT_Work_ITEM_TYPE

Project_Work_Item_Type		
Attribute	**Data type**	**Key**
ID	integer	primary
Project_ID	integer	foreign
Work_Item_Type_ID	integer	foreign

Table 12-3 Attributes of the DB2 table WORK_ITEM_TYPE

Work_Item_Type		
Attribute	**Data type**	**Key**
ID	integer	primary
Name	char(8)	
Short_Desc	varchar(50)	
Format	char(8)	
RecordLength	smallinit	
SCLM_Type	char(8)	

Table 12-4 Attributes of the DB2 table WORK_ITEM

Work_Item		
Attribute	**Data type**	**Key**
ID	integer	primary
Project_Work_Item_Type_ID	integer	foreign
Name	char(8)	
Short_Desc	varchar(50)	
SCLM_Name	char(8)	
SCLM_Lang	char(8)	
Create_User	char(8)	
Create_Date	timestamp	
Change_User	char(8)	
Change_Date	timestamp	
Build_User	char(8)	
Build_Date	timestamp	
Release_User	char(8)	
Release_Date	timestamp	
Release_State	smallinit	
Version_Number	smallinit	

A special feature of the SCM model is that SCLM is hidden from the user. The user gets access to his members through the hierarchical SCM model project “work item type” work item. An essential goal of our CARMA implementation is to adhere to this access model in future distributed software development environments under the CARMA view.

Thus, this implementation is an example of an SCM requirement that could not be solved by the SCLM DT plug-in, even though the SCM system in the background is SCLM. SCLM DT gives a data set view to the user, while we want to adhere to a project based view of the SCM model. This can be done by using the CARMA SCM framework, which, unlike the SCLM DT plug-in, can be adapted to the users' requests.

The mapping of our SCM model to the CARMA concepts is quite simple, as shown in Table 12-5.

Table 12-5 Mapping between the SCM model and the CARMA concept

SCM Model concept	CARMA concept
Project	Instance
Work Item Type	Container
Work Item	Member

12.2.2 Underlying SCLM project hierarchy

The SCLM project hierarchy in our sample CARMA model reflects the project structure of an insurance company.

There are five SCLM development groups:

- CUST: Customer
- PRAC: Property and Casualty
- LIFE: Life Insurance
- ENCA: Encashment (collection procedure)
- LIAB: Liability

The SCLM hierarchy consists of two consecutive promotion groups:

- INTTEST: Integration Test
- RELEASE: Release

Figure 12-2 shows the SCLM project hierarchy of our sample CARMA implementation.

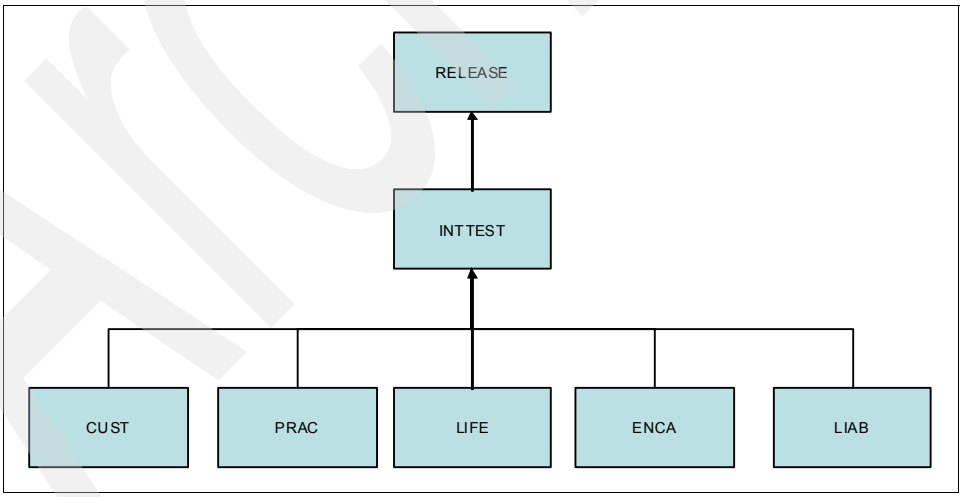


Figure 12-2 Underling SCLM project hierarchy

Archived

12.2.3 Functional SCM model

A major motivation of our sample CARMA implementation is to implement as many of the CARMA RAM interface functions as possible. Therefore, the functional SCM model we implemented may seem somewhat artificial or sophisticated to you. Try to ignore this as much as possible.

The simplified functional model is represented by Figure 12-3.

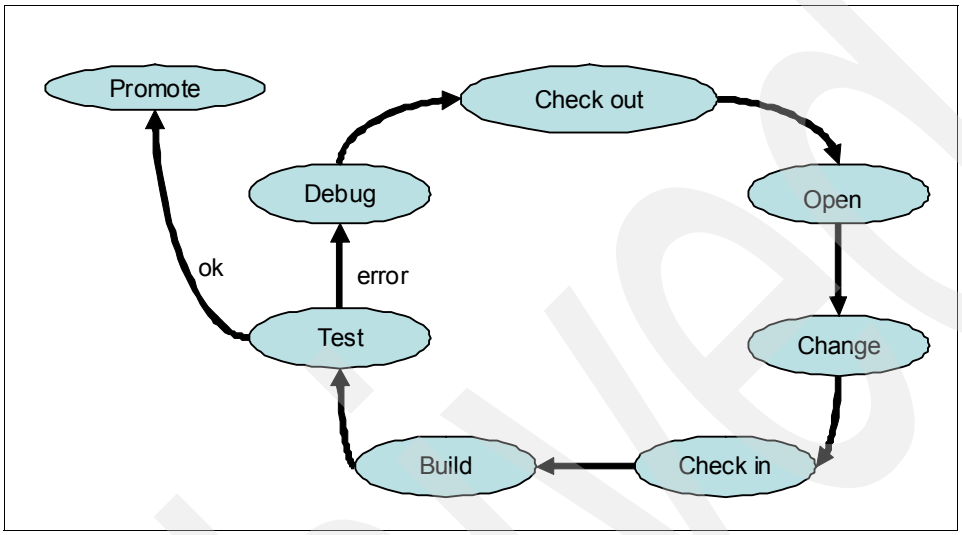


Figure 12-3 Functional model of the CARMA example

The user who wants to change a work item has to check out the item from the SCM system. This check-out draws the work item down within the SCLM hierarchy if necessary and provides it with the SCLM access key of the user to protect it against mutual changes.

Then the user has to open the work item. Each time the user saves the work item, its actual contents is transferred to the host and saved into a temporary file.

Before the user can build the work item he has to do a check-in. The check-in transfers the contents from the corresponding temporary file to the appropriate SCLM file and does an SCLM save. The temporary file is deleted and the user's access key is removed from the file.

Then the user can call the build function, which does an SCLM build. In this example RAM, we do not implement the SCLM build. The build merely consists of a compile/link job for compilable programs.

After the build finishes, the corresponding program can be tested. If the test ends successfully, the work item can be promoted within the SCLM hierarchy.

If the test is not successful, the program has to be debugged. Afterwards, the development life cycle starts again.

In addition to the described function of the standard development life cycle, our CARMA implementation contains the following actions:

- Create a new work item.

  The new work item is inserted into the DB2 work item table and its relation to the work item type and project is maintained. In addition, the corresponding member is added to the appropriate SCLM group.

Archived

- Delete a work item.

  The release status of the work item is determined. If it is not the SCLM status RELEASE and there is no other occurrence of the work item in the SCLM hierarchy, the work item entry is deleted from DB2 and from the corresponding SCLM data set.

  If it is not the SCLM status RELEASE and there is still another occurrence of the work item in the SCLM hierarchy, the release status of the work item is updated in the DB2 work item table accordingly and the member is deleted from the corresponding SCLM data set.

  If it is the SCLM status RELEASE, the user gets the message that a released work item cannot be deleted and no action takes place.

- Lock a work item.

  The work item is SCLM locked with the access key of the user to protect it against mutual changes.

- Unlock a work item.

  The work item is SCLM unlocked, that is, the user's access key is removed from the work item.

12.3 Implementing RAM

Now that we have identified and mapped the SCM components and presented the functional model, we can start with the implementation.

Your system programmer should have prepared the CARMA server job. This CARMA server job is started for each user when he calls a CARMA function for the first time.

The CRASRV.properties file in your UNIX System Services CARMA installation directory defines the location of the job under TSO.

The library to which your RAM is linked must be added to the steplib of this job so that the job will find your CARMA implementation.

In our case, we have to add the proper SDSNLOAD, which contains the DSNREXX module, to the ISPLLIB DD statement, because we will access DB2 from our REXX procedures through DSNREXX.

You, as the SCM responsible person, have basically two tasks:

- Configure the CARMA RAM functionality.
- Implement the CARMA RAM functions.

Both tasks are correlated to each other. In the configuring step, you define your extension to the standard CARMA model. That means you tell CARMA:

- The name of your RAM
- The name and parameters of your user defined functions
- The data format of your user defined parameters

In the implementation step, you realize your extensions to the standard CARMA model in accordance with your configuration information.

Archived

12.3.1 Configuring CARMA RAM functionality

Configuring CARMA is done by adapting the two VSAM files:

- CRADEF
- CRASTRS

There is a job for each of these VSAM files to load them from a PO member under the TSO CARMA installation directory (ask your system programmer for the SCRASAMP file). In the SCRAVSAY file, you can find two PO members with the configuring definitions for the sample RAMs that are provided with the installation of WD/z CARMA. In order to add your own definitions, you must understand the structure of these entries.

The CRADEF VSAM file is the language-independent part of the configuration. For each entry in the CRADEF VSAM file, there must exist a corresponding entry in the CRASTRS VSAM file. If you want to support multiple languages, you must provide an entry in the CRASTRS VSAM file for each language.

There are basically four different types of entries (marked by the first letter):

- R: RAM
- A: Action
- P: Parameter
- T: Return value

RAM record

The RAM record describes your own RAM. It must be present so that CARMA recognizes that there is another RAM to display.

For example:

```
R04000  1.0     C       Z/OSV1R41.0     REDBRAM
```

This is RAM 04 Version 1.0 of our CARMA implementation. The DLL is implemented in the programming language C. It is running under z/OS V1.4 and the name is REDBRAM.

Action record

You must add an action record for your RAM if you have your own user defined functions or if you extend a given CARMA function with additional custom parameters.

For example:

```
A04017  001|
```

The function 017 (that is, createMember) of RAM 04 is extended by the user defined input parameter 001. The | sign is the separator between input and output parameters (the German separator sign is !).

We extend the parameter list of the createMember CARMA function in our example by the parameter SCLM language.

For example:

```
A04100  000|
```

There is a user defined function 100 in the RAM 04 with the input parameter 000. All functions higher than 099 are user defined.

The user defined function 100 in our example is the build function.

Parameter record

For each custom input parameter that you address in an action record of your RAM, you must add a parameter record.

For example:

```
P04001  STRING          8               N
```

The parameter 001 of RAM 01 has the data format String(8). N means the parameter can contain different values.

Return value

For each custom output parameter that you address in an action record of your RAM, you must add a return value record.

In the CRASTRS VSAM file, you specify short and long descriptions for the entries in the CRADEF VSAM file. You can specify descriptions in each language you want to support.

The text that you specify here will occur in the CARMA view at the appropriate place.

For example:

```
EN_US   00037R04000  REDB RAM        Sample RAM For The book
```

The RAM 04 gets the display name REDB RAM on the CARMA view.

For example:

```
EN_US   00037P04001  LANGUAGE                        What is the language?
```

In the panel that is displayed when creating a new member, the language parameter is asked for by the prompt “What is the language?”.

Note: The EN_US entry must always exist, even if you do not want to work with English language support.

12.3.2 Implementing CARMA RAM functions

The easiest way to implement your own RAM is to take one of the sample RAMs provided with the WD/z host installation as a basic copy. The sample RAMs reside in the SCRASAMP data set (ask your system programmer for the exact name).

Within this data set, you can also find a job (CRARAMCS) to compile/link a RAM.

General considerations

For our example, we use one of the RAMs that has been implemented in the programming language C. There is also a COBOL example, but for our needs the C examples are the better choice. Within the C source, the CARMA interface functions are published by the pragma export keyword.

Our goal is to implement the actual functionality of the interface functions in TSO REXX procedures. The main argument for this decision is that there is a lot of SCM functionality in the z/OS environment implemented in REXX procedures. Thus, in this way, it is possible to reuse existing code.

You can use Example 12-1 as the coding used to call a REXX procedure from a C program.

Example 12-1 Coding used to call a REXX procedure from a C program

```
char *execREXXProcedure(char *procname, int narg, ... ) {
  va_list argptr;

  OSFUNC *fptr;
  int rc;
  int p0 = 0;
  static char result[LBUF+1];
int n;
  EXECBLK execblk;
  EXECBLK* pexecblk = &execblk;

  /* execution code */
  fptr = (OSFUNC *)fetch("IRXEXEC ");
  if (fptr == NULL) {
    puts("IRXEXEC fetch error");
    return "";
  }

  char acryn[8] = "IRXEXECB";
  memcpy(execblk.exec_blk_acryn,acryn,8);
  execblk.exec_blk_length = 48;
  memset(execblk.exec_blk_reserv,0,4);
  memset(execblk.exec_member,' ',8);
  n = MIN(strlen(procname),8);
  memcpy(execblk.exec_member,procname,n);
  char ddname[8] = "SYSPROC ";
  memcpy(execblk.exec_ddname,ddname,8);
  char subcom[8] = "        ";
  memcpy(execblk.exec_subcom,subcom,8);
  execblk.exec_dsnptr = 0;
  execblk.exec_dsnlen = 0;

  va_start(argptr, narg); /* get first argument */
  for (n=0; n < narg; n++) {
    char *pc;

    pc = va_arg(argptr, char *);
    arglist_entry[n].argstring_ptr = pc;
    arglist_entry[n].argstring_length = strlen(pc);
  }
  va_end(argptr);

  arglist_entry[n+1].argstring_ptr = (void*)-1;
  arglist_entry[n+1].argstring_length = -1;

  EVALBLOCK *pevalblk = malloc(LBUF + 16);
  pevalblk->evalblock_evpad1 = 0;
  pevalblk->evalblock_evsize = LBUF/8 + 2;
  pevalblk->evalblock_evlen = 0;
  pevalblk->evalblock_evpad2 = 0;
rc = (*fptr)(&pexecblk,&parglist,SUBROUTINE,p0,p0,&pevalblk,p0,p0);
  if (rc != 0) {
```

Archived

```
    printf("IRXEXEC RC:%d\n",rc);
    free(pevalblk);
    return "";
  }

  pevalblk->evalblock_evlen = strlen(pevalblk-> evalblock_evdata);
  /* make returned result C-processable */
  memcpy(result,pevalblk->evalblock_evdata,pevalblk-> evalblock_evlen);
  result[pevalblk->evalblock_evlen] = '\0';
  free(pevalblk);
  return result;
}
```

Example 12-2 is an example of a call to a REXX procedure from our sample RAM REDBRAM.

Example 12-2 Coding to call to a REXX procedure from our sample RAM REDBRAM

```
int getInstances(Descriptor** RIrecords, int* numRecords,
                 void** params, void*** customReturn, char filter[256],
                 char error[256]) {

   int MAXRECS = 10000;
   char buf[300];
   memset(buf, ' ', 300);
   *RIrecords = (Descriptor*)
               malloc(sizeof(Descriptor) * MAXRECS);
   sprintf(buf, "%i %i %.256s", MAXRECS, *RIrecords, filter);

   char *rexxRes = execREXXProcedure("REDBCAGI", 1, buf);

   rexxRes[3] = '\0';
   rexxRes[9] = '\0';
   int retc = atoi(rexxRes);
   *numRecords = atoi(rexxRes+4);
   memcpy(error, rexxRes+10, 256);

   return retc;
}
```

In this example, the REXX procedure REDBCAGI is called. This is the REXX procedure that implements the CARMA getInstances interface function. The main difficulty is the correct handling of the input and output parameters between the C program and the REXX procedure.

In this case, the REXX procedure is called with three input parameters:

- MAXRECS: The maximum record count that can be processed by this call.
- *RIrecords: A pointer to the storage area where CARMA expects placement of the resulting records.
- filter: A filter description by which you can restrict the selected records.

MAXRECS is a parameter that is defined within the body of the getInstances function. *RIrecords and filter are parameters of the getInstances function itself that are passed to the REXX procedure.

Archived

The storage that is addressed by the pointer *RIrecords must be allocated before we call the REXX procedure. This can be done with intrinsic C function malloc. We allocate a storage area of the size MAXRECS multiplied by the size of the Descriptor record. The Descriptor record is a CARMA predefined data structure to pass information back to CARMA.

The structure of the Descriptor record is shown in Table 12-6.

Table 12-6 Descriptor record

Field	Description
char id[256]	Unique ID to describe the entity
char name[64]	Display name

We use the C intrinsic function sprintf to format the input parameters in a way that can be understood by the called REXX procedure. Using the sprintf function, the three parameters are written in a formatted way to the “C char array buf”, which has been previously allocated.

The result of our REXX procedure is returned to the storage area in the C program pointed to by rexxRes. The REXX procedure returns its parameters through the command **exit res**, where res contains the concatenated output parameters separated by blanks.

The calling C RAM has to separate the returned values before it passes them back to CARMA without separators. In addition, we have to convert numeric values by the C intrinsic function atoi. String results are copied by the function memcpy into the corresponding char array, as in our error string example.

Sample function implementations

Let us take a look at the REXX implementation of some CARMA RAM interface functions. We selected the function getInstances as an example of the CARMA browsing functions, getAllMemberInfo as an example of the CARMA meta data functions, and check-in as an example of other functions.

For simplification reasons, we have given the following aliases to our DB2 tables, as shown in Table 12-7.

Table 12-7 Aliases for the DB2 tables

DB2 type	Aliases name
PRJ_WI_TYPE	PROJECT_WORK_ITEM_TYPE
WI_TYPE	WORK_ITEM_TYPE
WI	WORK_ITEM

Implementation of getInstances

The getInstances function has to return to CARMA the entities of the highest level of the CARMA hierarchy. In our case, this means a list of projects. As we have seen before, the name of the REXX procedure is REDBCAGI.

Example 12-3 shows the coding of the core functionality of REDBCAGI.

Example 12-3 Coding for REDBCAGI

```
Say $n "started..."
Arg maxrecs adr filter
```

```
anzRecs = 0
retID. = ''
retID.0 = 0
filter = Strip(filter)
filter = Translate(filter,'%','*')
ADDRESS TSO "SUBCOM DSNREXX"
subcomrc = rc
if subcomrc then
   s_rc = RXSUBCOM('ADD','DSNREXX','DSNREXX')

ADDRESS DSNREXX
"CONNECT DB8G"
if rc <> 0 then do
   errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
   retc = 8
   call Exit retc 0 errmsg
end

/* determining the projects considering the filter information */
sqlstmt = "SELECT NAME FROM PROJECT WHERE NAME LIKE '"filter"'"
"EXECSQL DECLARE C1 CURSOR FOR S1 "
"EXECSQL PREPARE S1 FROM :sqlstmt"
if rc <> 0 then do
   errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
   retc = 8
   call Exit retc retID.0 errmsg
end
"EXECSQL OPEN C1"

j1 = 0
retID. = ''
retID.0 = 0
fetchrc = 0
Do While fetchrc = 0
  "EXECSQL FETCH C1 INTO :prj"
  fetchrc = rc
  If fetchrc = 0 Then Do
     j1 = j1 + 1
     retID.j1 = 'PJ-'prj
     retName.j1 = "Projekt" prj
     End
  End
"EXECSQL CLOSE C1"

ADDRESS ISPEXEC
retID.0 = j1
retc = 0
errmsg = ""

/* transfer the projects names to the storage area declared by CARMA  */
Do i = 1 To retID.0
  anzRecs = anzRecs + 1
  x = Storage(D2X(adr), 256, Left(retID.i, 256))
  x = Storage(D2X(adr+256), 64, Left(retName.i, 64))
  adr = adr + 320
```

Archived

```
  If anzRecs > maxrecs Then Do
    errmsg = "Not enough storage reserved! "!!,
             "(max records possible: "maxrecs")"
    retc = 8
    Leave
    End
  End

if subcomrc then
   s_rc = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
Say $n "exit "retc

/*return res */
call Exit retc retID.0 errmsg

EXIT:
Parse Arg exitrc retid exitstring

If "SYMBOL"('$T') ^= 'VAR' Then
  Parse Source . $t $n .

res = Right(exitrc, 3, '0')
res = res Right(retID, 5, '0')
res = res exitstring!!d2c(0)

If $t = 'SUBROUTINE' Then
   Exit res
Else
   Exit exitrc
```

The REXX procedure has three parameters:

- maxrecs: The maximum records that can be retrieved by an getInstances call.
- adr: An address pointer to the storage area where CARMA expects the result of the getInstances call.
- filter: A filter attribute by which the selected records can be restricted.

The REXX procedure connects to the DB2 subsystem. After that, it reads the project information from the DB2 project table according to the filter given by an SQL select statement. To do this task, the filter has to be formatted for DB2, that is, the sign * has to be changed to the sign %. In the SQL fetch loop, the contents from the database selection is transferred into two stem variables whose ID and name correspond to the entries of the CARMA descriptor record.

After the SQL fetch loop, the contents of the two stem variables are transferred in a formatted way to the storage area that was provided by the calling CARMA getInstances interface function. You can use the REXX intrinsic function storage to place the contents into a certain storage address.

Note: You should think about naming conventions for the IDs of your CARMA entries so that you can easily distinguish between the CARMA instances, containers, and members. In our example, the instance entries (that is, projects) get the prefix PJ-.

Archived

Implementation of getAllMemberInfo

The function getAllMemberInfo is called when the properties function is clicked in the context menu of a CARMA container or member. It fills the property/value table of the displayed properties view. You can decide which attributes of the work item types and work items you want to display in this property/value table.

The name of the REXX procedure implementing the CARMA interface function getAllMemberInfo is REDBCAGA.

Example 12-4 shows the coding of the core functionality of REDBCAGA.

Example 12-4 Coding of REDBCAGA

```
Say $n "started..."
Parse Arg instanceId memberId adr maxAttr

anzAttr = 0
instanceId = Strip(instanceId)
memberId = Strip(memberId)
prj = Substr(instanceID,4)
If Substr(memberID,1,3) = 'WI-' Then Do
   lp_strich = Lastpos('-',memberID)
   wi_type = Substr(memberID,4,lp_strich-4)
   wi = Substr(memberID,lp_strich+1)
End
Else Do
   wi_type = Substr(memberID,4)
End

ADDRESS TSO "SUBCOM DSNREXX"
subcomrc = rc
if subcomrc then
   s_rc = RXSUBCOM('ADD','DSNREXX','DSNREXX')

ADDRESS DSNREXX
"CONNECT DB8G"
if rc <> 0 then do
   errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
   retc = 8
   call Exit retc retID.0 errmsg
end

j1 = 0
/* read the attributes from DB2 and pass it to the key-value area   */
/* of CARMA                                                         */
If Substr(memberId,1,3) = 'WI-' Then Do
   sqlstmt = "SELECT SCLM_NAME, SCLM_LANG ",
             "     , CREATE_USER, CREATE_DATE, CHANGE_USER ",
             "     , CHANGE_DATE, BUILD_USER, BUILD_DATE ",
             "     , RELEASE_USER, RELEASE_DATE ",
             "     , RELEASE_STATE, VERSION_NUMBER ",
             "FROM PROJECT T1, PRJ_WI_TYPE T2, WI_TYPE T3, WI T4 ",
             "WHERE T1.NAME = '"prj"' " ,
             "AND   T1.ID = T2.PROJECT_ID ",
             "AND   T2.WORK_ITEM_TYPE_ID = T3.ID ",
             "AND   T3.NAME = '"wi_type"' ",
```

Archived

```
            "AND   T2.ID = T4.PROJECT_WORK_ITEM_TYPE_ID ",
            "AND   T4.NAME = '"wi"'"
   "EXECSQL DECLARE C1 CURSOR FOR S1 "
   "EXECSQL PREPARE S1 FROM :sqlstmt"
   if rc <> 0 then do
      errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
      retc = 8
call Exit retc retID.0 errmsg
   end
   "EXECSQL OPEN C1"
j1 = 0
   retID. = ''
   retID.0 = 0
   fetchrc = 0
   Do While fetchrc = 0
     "EXECSQL FETCH C1 INTO :name, :lang, :cr_user, :cr_date "!!,
                ", :chg_user, :chg_date, :build_user, :build_date "!!,
                ", :rel_user, :rel_date, :rel_state, :vers "
     fetchrc = rc
     End
   "EXECSQL CLOSE C1"

   ADDRESS ISPEXEC
   j1 = j1 + 1
   retKey.j1 = 'SCLM-Name'
   retValue.j1 = name
   j1 = j1 + 1
   retKey.j1 = 'SCLM-Language'
   retValue.j1 = lang
   If lang = 'COBOL' Then Do
      j1 = j1 + 1
      retKey.j1 = 'carma.file-extension'
      retValue.j1 = 'cbl'
   End
   j1 = j1 + 1
   retKey.j1 = 'Creation User'
   retValue.j1 = cr_user
   j1 = j1 + 1
   retKey.j1 = 'Creation Date'
   retValue.j1 = Substr(cr_date,1,10) Substr(cr_date,12,8)
   j1 = j1 + 1
   retKey.j1 = 'Change User'
   retValue.j1 = chg_user
   j1 = j1 + 1
   retKey.j1 = 'Change Date'
   retValue.j1 = Substr(chg_date,1,10) Substr(chg_date,12,8)
   j1 = j1 + 1
   retKey.j1 = 'Build User'
   retValue.j1 = build_user
   j1 = j1 + 1
   retKey.j1 = 'Build Date'
   retValue.j1 = Substr(build_date,1,10) Substr(build_date,12,8)
   j1 = j1 + 1
   retKey.j1 = 'Release User'
retValue.j1 = rel_user
```

Archived

```
   j1 = j1 + 1
   retKey.j1 = 'Release Date'
   retValue.j1 = Substr(rel_date,1,10) Substr(rel_date,12,8)
   j1 = j1 + 1
   retKey.j1 = 'Release State'
   retValue.j1 = rel_state
   j1 = j1 + 1
   retKey.j1 = 'Version'
   retValue.j1 = vers
End

If Substr(memberId,1,3) = 'WT-' Then Do
   sqlstmt = "SELECT FORMAT, RECORDLENGTH ",
             "     , SCLM_TYPE ",
             "FROM PROJECT T1, PRJ_WI_TYPE T2, WI_TYPE T3 ",
             "WHERE T1.NAME = '"prj"' ",
             "AND   T1.ID = T2.PROJECT_ID ",
"AND   T2.WORK_ITEM_TYPE_ID = T3.ID ",
             "AND   T3.NAME = '"wi_type"'"
   "EXECSQL DECLARE C1 CURSOR FOR S1 "
   "EXECSQL PREPARE S1 FROM :sqlstmt"
   if rc <> 0 then do
      errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
      retc = 8
      call Exit retc retID.0 errmsg
   end
   "EXECSQL OPEN C1"

   j1 = 0
   retID. = ''
   retID.0 = 0
   fetchrc = 0
   Do While fetchrc = 0
     "EXECSQL FETCH C1 INTO :recfm, :lrecl, :sclmtype "
     fetchrc = rc
     End
   "EXECSQL CLOSE C1"

   ADDRESS ISPEXEC
   j1 = j1 + 1
   retKey.j1 = 'Record Format'
   retValue.j1 = recfm
   j1 = j1 + 1
   retKey.j1 = 'Logical Record Length'
   retValue.j1 = lrecl
   j1 = j1 + 1
   retKey.j1 = 'SCLM Type'
   retValue.j1 = sclmtype
End


retKey.0 = j1
retc = 0
errmsg = ""
```

Archived

```
/* put the key-value information into the storage area provided    */
/* by CARMA                                                        */
Do i = 1 To retKey.0
  anzAttr = anzAttr + 1
  x = Storage(D2X(adr), 64, Left(retKey.i, 64))
  x = Storage(D2X(adr+64), 256, Left(retValue.i, 256))
  adr = adr + 320
  If anzAttr > maxAttr Then Do
errmsg = "Not enough storage reserved! "!!,
             "(max. "maxAttr" attributes are possible)"
    retc = 8
    Leave
    End
  End

Say $n "exit "retc

/*return res */
call Exit retc retID.0 errmsg

EXIT:
Parse Arg exitrc retid exitstring

If "SYMBOL"('$T') ^= 'VAR' Then
  Parse Source . $t $n .

res = Right(exitrc, 3, '0')
res = res Right(retID, 5, '0')
res = res exitstring!!d2c(0)

If $t = 'SUBROUTINE' Then
   Exit res
Else
   Exit exitrc
```

The REXX procedure has four parameters:

- instanceID; The CARMA specific ID of the current instance entry. This ID was defined in the getInstance call.
- memberID; The CARMA specific ID of the current container or member entry. The ID was defined in the getContainerContents or, respectively, in the getMembers call.
- adr: The address pointer to the storage area where CARMA expects the result of the getAllMemberInfo call.
- maxAttr: The maximum count of attributes that can be processed by a getAllMemberInfo call.

The REXX procedure connects to the DB2 subsystem. Depending on the type of the memberID (work item type or work item), the corresponding attributes are selected from the DB2 table WI_TYPE or, respectively, WI. In the SQL fetch loop, the attributes are fetched into REXX host variables. After the SQL fetch loop, the fetched attributes are assigned to a key/value REXX stem variable pair. The key stem variable contains a name of the attribute of your choice. The value stem variable contains the value of the attribute from the DB2 table.

Archived

The contents of the stem variables is transferred to the storage area that is provided by the calling CARMA interface function getAllMemberInfo. The structure of this storage area is shown in Table 12-8.

Table 12-8 Key Val pair record

Field	Description
char key[64]	Attribute's name
char value[256]	Attributes's name

implementation of checkIn

The CARMA interface function checkIn is called when **Check In** is selected in the context menu of a CARMA container or member. In our sample, we only implement the checkIn function for a single member. The name of the REXX implementing the CARMA interface function checkIn is REDBCACI.

Example 12-5 shows the coding of the core functionality of REDBCACI.

Example 12-5 Coding for REDBCACI

```
Say $n "started..."
Arg instanceID memberID

instanceID = Strip(instanceID)
memberID = Strip(memberID)
user = USERID()
If Substr(memberID,1,3) <> 'WI-' Then Do
   retc = 8
   msg = 'Check In is only realized for a single member selection.'
   call Exit retc 0 msg
End

/* determining SCLM group and type */
prj = Substr(instanceID,4)
lp_strich = Lastpos('-',memberID)
wi_type = Substr(memberID,4,lp_strich-4)
wi = Substr(memberID,lp_strich+1)
ADDRESS TSO "SUBCOM DSNREXX"
subcomrc = rc
if subcomrc then
   s_rc = RXSUBCOM('ADD','DSNREXX','DSNREXX')

ADDRESS DSNREXX
"CONNECT DB8G"
if rc <> 0 then do
   errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
   retc = 8
   call Exit retc retID.0 errmsg
end
sqlstmt = "SELECT SCLM_GROUP, SCLM_TYPE, T4.ID ",
          "FROM PROJECT T1, PRJ_WI_TYPE T2, WI_TYPE T3, WI T4 ",
          "WHERE T1.NAME = '"prj"' ",
          "AND   T1.ID = T2.PROJECT_ID ",
          "AND   T2.WORK_ITEM_TYPE_ID = T3.ID ",
          "AND   T3.NAME = '"wi_type"' ",
```

```
          "AND   T2.ID = T4.PROJECT_WORK_ITEM_TYPE_ID ",
          "AND   T4.NAME = '"wi"'"

"EXECSQL DECLARE C1 CURSOR FOR S1 "
"EXECSQL PREPARE S1 FROM :sqlstmt"
if rc <> 0 then do
   errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
   retc = 8
   call Exit retc retID.0 errmsg
end
"EXECSQL OPEN C1"

j1 = 0
retID. = ''
retID.0 = 0
fetchrc = 0
Do While fetchrc = 0
  "EXECSQL FETCH C1 INTO :grp, :dstyp, :wi_id"
  fetchrc = rc
  End
"EXECSQL CLOSE C1"

grp = Strip(grp)
dstyp = Strip(dstyp)
ADDRESS ISPEXEC
Call ISPF '8 SELECT CMD(FLMCMD LOCK,'SCLMProj',,'grp','dstyp','wi,
             ',,'user',)'
If rc = 8 Then Do
   retc = 4
   say 'Member 'wi' is  locked by another user.'
   msg = 'Member 'wi' is locked by another user.'
   call Exit retc 0 msg
End

/* Copy the member from the temporary data set to the corresponding */
/* SCLM data set                                                    */
tmpdsn = "'"user".CARMA."wi"'"
dsn = "'"SCLMProj"."grp"."dstyp"("wi")'"
Call TSO '12 ALLOC FI(MIN) DA('tmpdsn') SHR REUSE'
If rc = 12 Then Do
   retc = 4
   say 'Member 'wi' was not changed and saved before.'
   msg = 'Member 'wi' was not changed and saved before.'
   call Exit retc 0 msg
End
Call TSO '0 ALLOC FI(MOUT) DA('dsn') SHR REUSE'
Call TSO "0 EXECIO * DISKR MIN (STEM MEMCONT. FINIS)"
Call TSO "0 EXECIO * DISKW MOUT (STEM MEMCONT. FINIS)"
Call TSO '0 FREE FI(MIN)'
Call TSO '0 FREE FI(MOUT)'
Call TSO '0 DELETE 'tmpdsn
/* execute SCLM SAVE                                            */
Call ISPF '0 SELECT CMD(FLMCMD SAVE,'SCLMProj',,'grp','!!,
           dstyp','wi',,'user',,,,,,,)'
/* execute SCLM UNLOCK                                          */
```

Archived

```
Call ISPF '4 SELECT CMD(FLMCMD UNLOCK,'SCLMProj',,'grp','dstyp','wi,
                        ','user')'

h_date = date()
If Length(h_date) = 10 Then h_date = '0'h_date
month = Substr(h_date,4,3)
select
  When month = 'Jan' Then month = '01'
  When month = 'Feb' Then month = '02'
  When month = 'Mar' Then month = '03'
  When month = 'Apr' Then month = '04'
  When month = 'May' Then month = '05'
  When month = 'Jun' Then month = '06'
  When month = 'Jul' Then month = '07'
  When month = 'Aug' Then month = '08'
  When month = 'Sep' Then month = '09'
  When month = 'Oct' Then month = '10'
  When month = 'Nov' Then month = '11'
  When month = 'Dec' Then month = '12'
  Otherwise Nop
End
h_time = Time()
h_time = Translate(h_time,'.',':')
h_date = Substr(h_date,8,4)'-'month'-'Substr(h_date,1,2)'-'h_time
ADDRESS DSNREXX
"EXECSQL UPDATE WI ",
"    SET CHANGE_USER = '"user"'",
"      , CHANGE_DATE = '"h_date"'",
"    WHERE ID = "wi_id
if rc <> 0 then do
   errmsg = "SQL-RC: "sqlcode"   SQL error: "sqlerrmsg
   retc = 8
   call Exit retc retID.0 errmsg
end
"EXECSQL COMMIT"
ADDRESS ISPEXEC
if subcomrc then
   s_rc = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
Say $n "exit "retc
/*return res */
call Exit retc 0 msg

TSO:
  Arg tsomax tsocmd
  If debug ^= 1 Then
    msgmode = OUTTRAP('tsomsg.')
  tsomsg.0 = 0
  Address TSO tsocmd
  rctso = rc
  msgmode = OUTTRAP('OFF')
  If tsomax ^= '*' Then
    If rctso < 0 ! rctso > tsomax Then Do
       tsorc = rctso
       cmd = tsocmd
       Say $n': RC:' tsorc 'in' tsocmd
```

Archived

```
       If Datatype(tsomsg.0, 'W') = 1 Then
         Do tsocnt = 1 To tsomsg.0
           Say Strip(tsomsg.tsocnt, 'T')
           End
       msgtext = 'rc='tsorc' in 'cmd
       Say msgtext
        Call EXIT 11 0 msgtext
       End
  rc = rctso
  Return rc

ISPF:
  Arg ispfmax ispfcmd
  Drop zerrmsg
  Address ISPEXEC ispfcmd                    /* execute ISPF-Command */
  ispfrc = rc
  If ispfmax ^= '*' Then
    If ispfrc<0 ! ispfrc > ispfmax Then Do    /* bad returncode       */
      If "SYMBOL"('ZERRMSG') = 'VAR' Then Do
        intvlm = '"'ispfcmd'" result: 'ispfrc' "'zerrlm'"'
        msgtext = intvlm
        Say msgtext
        Call EXIT 11 0 msgtext
        End
      Else Do
        intvrc = ispfrc
        intvcmd = ispfcmd
        msgtext = 'RC' intvrc 'in' intvcmd
        Say msgtext
        Call EXIT 11 0 msgtext
        End
      End
  rc = ispfrc
  Return ispfrc

EXIT:
Parse Arg exitrc retid exitstring

If "SYMBOL"('$T') ^= 'VAR' Then
  Parse Source . $t $n .

res = Right(exitrc, 3, '0')
res = res Right(retID, 5, '0')
res = res exitstring!!d2c(0)

If $t = 'SUBROUTINE' Then
   Exit res
Else
   Exit exitrc
```

Archived

The REXX procedure has two parameters:

- instanceID: The CARMA specific ID of the current instance entry. This ID is defined in the getInstance call.
- memberID: The CARMA specific ID of the current container or member entry. The ID is defined in the getMembers or, respectively, getContainerContents call.

The REXX procedure determines the name of the work item type or, respectively, the work item from the given memberID according to our naming conventions. After that, it connects to DB2 to determine the SCLM group and the type of the work item. In the next step, we try to execute an SCLM lock on the work item with the access key of the current user to see if the work item has already been locked by another user. If not, the work item content is copied from the appropriate temporary data set to the corresponding SCLM data set and an SCLM save and unlock is performed. The temporary data set is deleted.

Finally, we update the attributes of the change user and change date of the work item in the DB2 table WI.

Archived

12.4 Sample CARMA session

As you can see in the CARMA view, as shown in Figure 12-4, we have implemented five RAMs on our host. Three of them (PDS RAM, SCLM RAM, and TEST RAM) are the sample RAMs delivered by IBM with the installation. The INTER RAM is a prototype solution for a customer. The REDB RAM is the RAM we are discussing here.

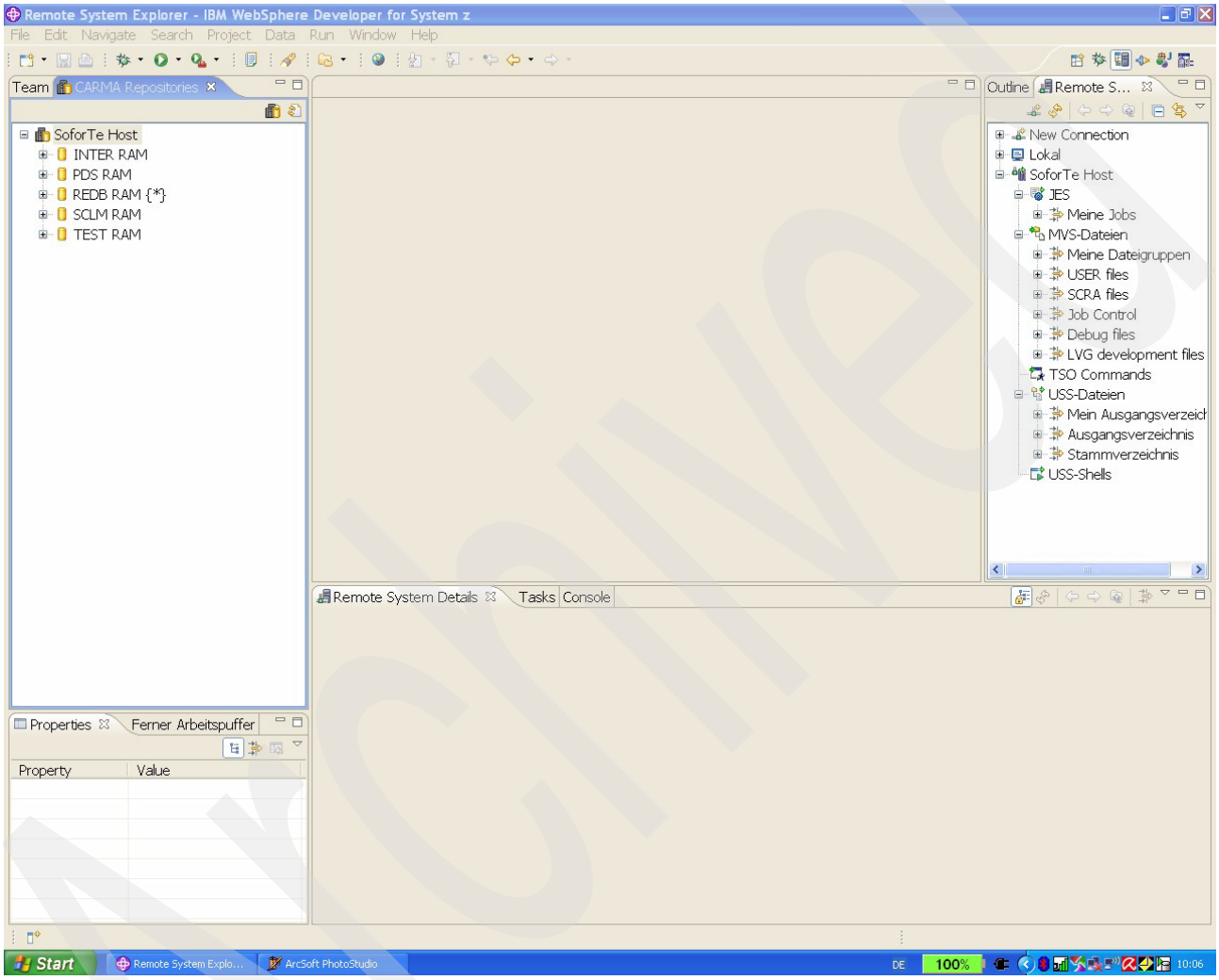


Figure 12-4 RAMs on the CARMA view

After opening the RAM, you see the projects that are defined in the DB2 table project, as shown in Figure 12-5 on page 435.

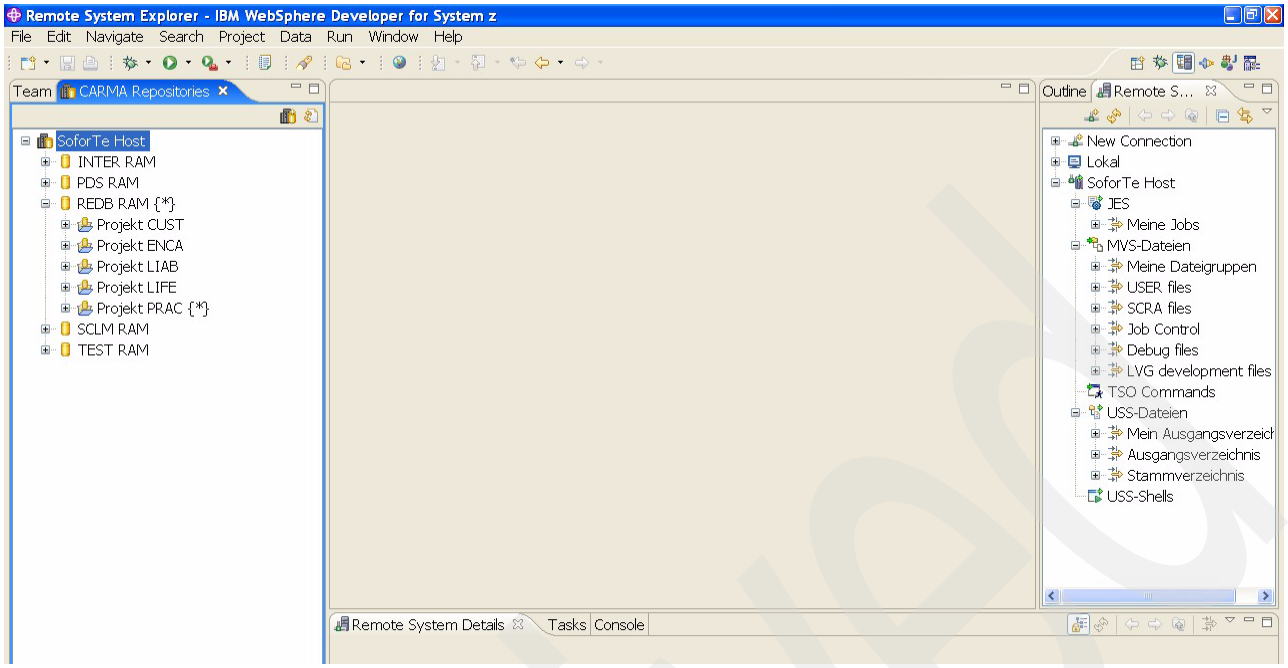


Figure 12-5 Projects of RAM REDB RAM

We look at the project PRAC (Property and Casualty). After opening the project, you see the work item types that are defined for this project, as shown in Figure 12-6.

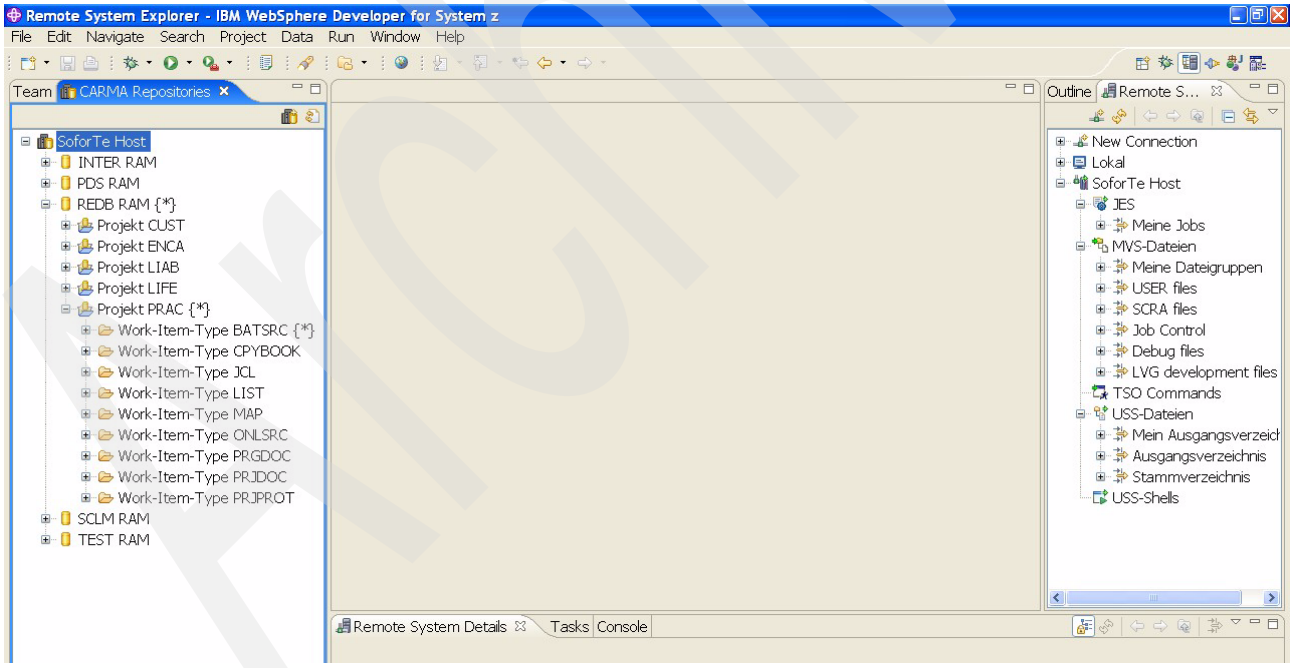


Figure 12-6 Work item types of project PRAC

Next, we look at the property page of the work item type BATSRC, as shown in Figure 12-7.

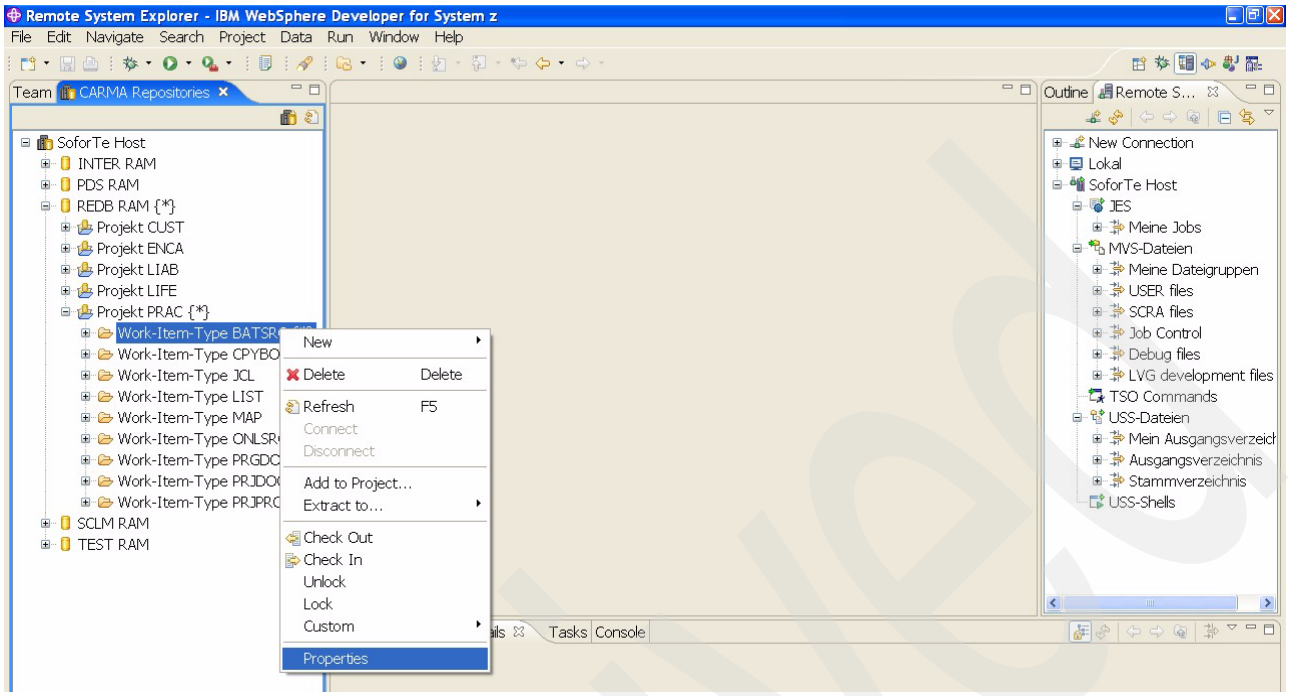


Figure 12-7 Choosing properties of work item type BATSRC

There is a property/value table at the bottom of the property page where your self-defined properties are displayed, as shown in Figure 12-8.

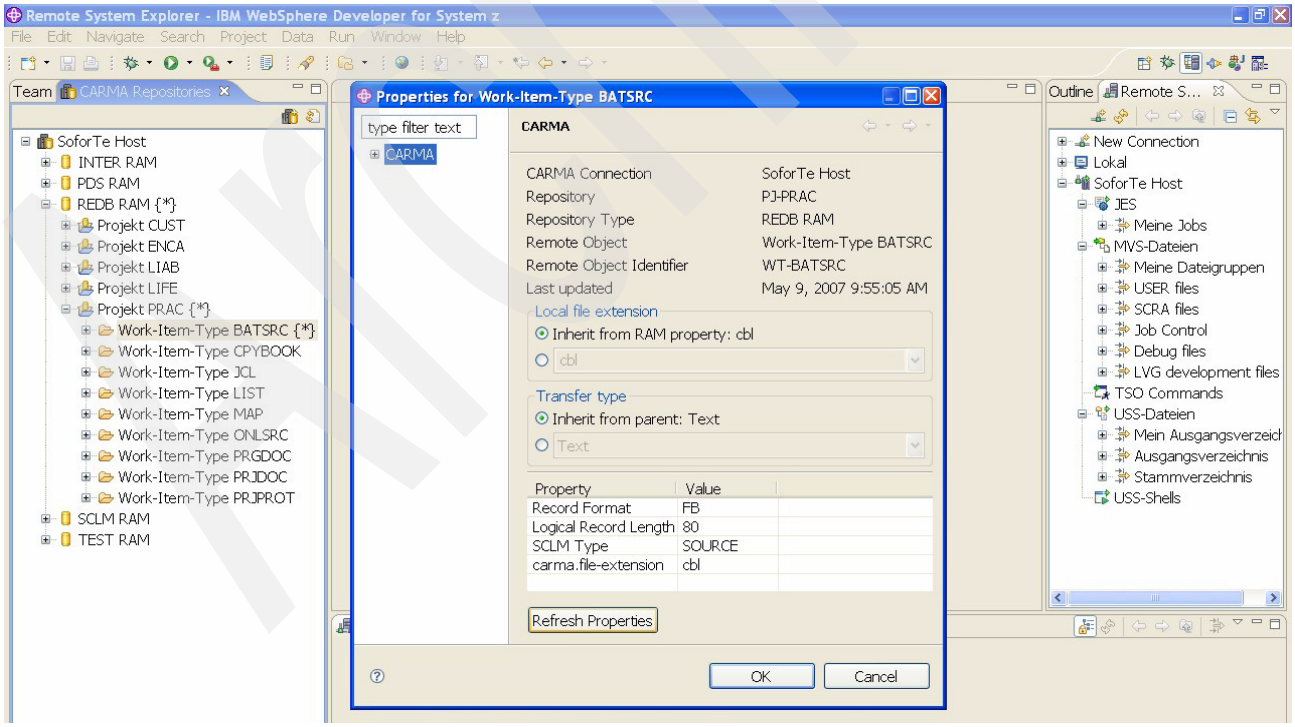


Figure 12-8 Property page of work item type BATSRC

Now we completely walk through the application life cycle. The first step is to create a new member, as shown in Figure 12-9.

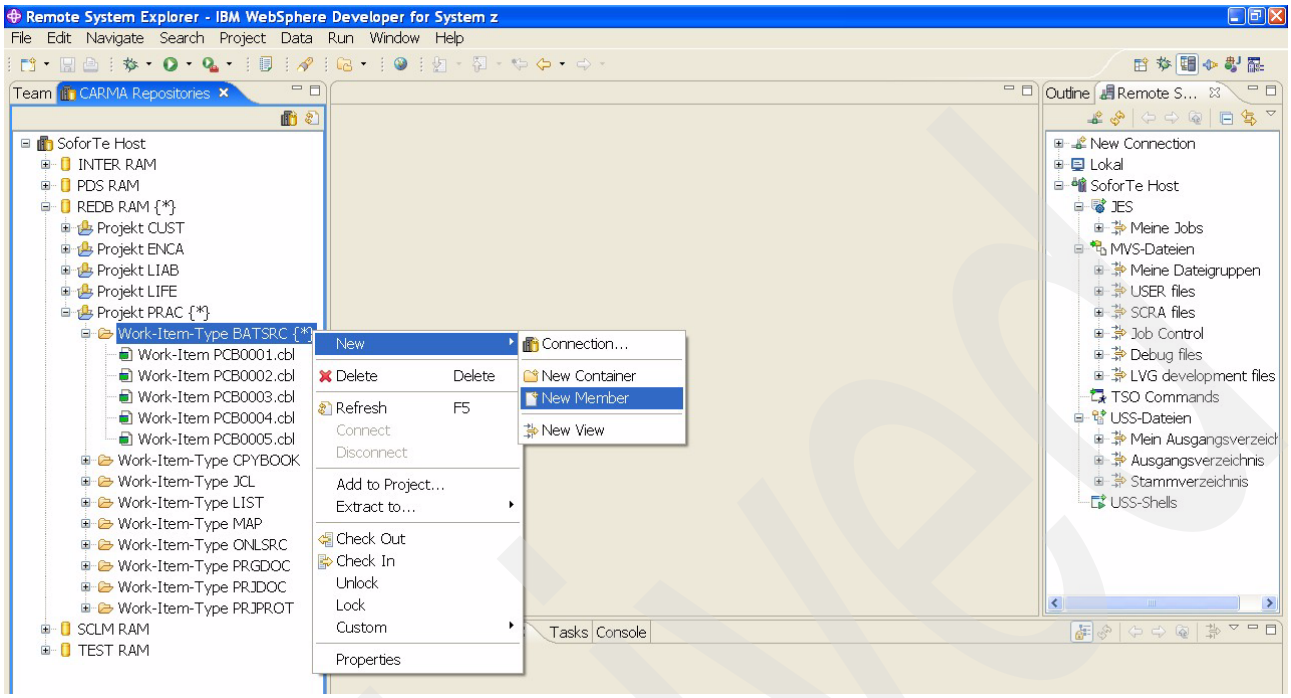


Figure 12-9 Creating a new member

We get a window where we have to enter the name of the new member. If desired, the container for a member can be changed, as shown in Figure 12-10.

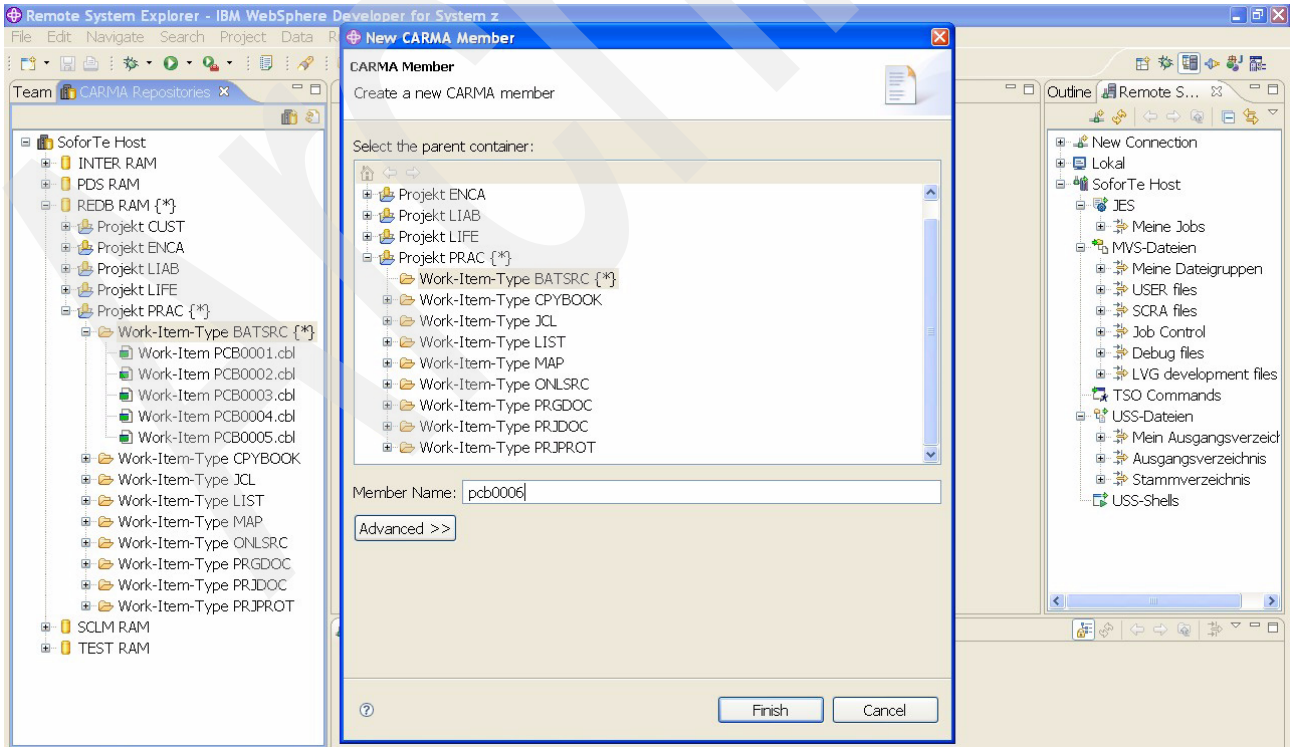


Figure 12-10 Request for the name of the new member

As this is the first action for which CARMA needs more information than is stored in the workspace, the client is now connecting to the CARMA task on the host. This means a CARMA server job assigned to the current user is started on the host. This job serves the CARMA requests of this connected user, as shown in Figure 12-11.

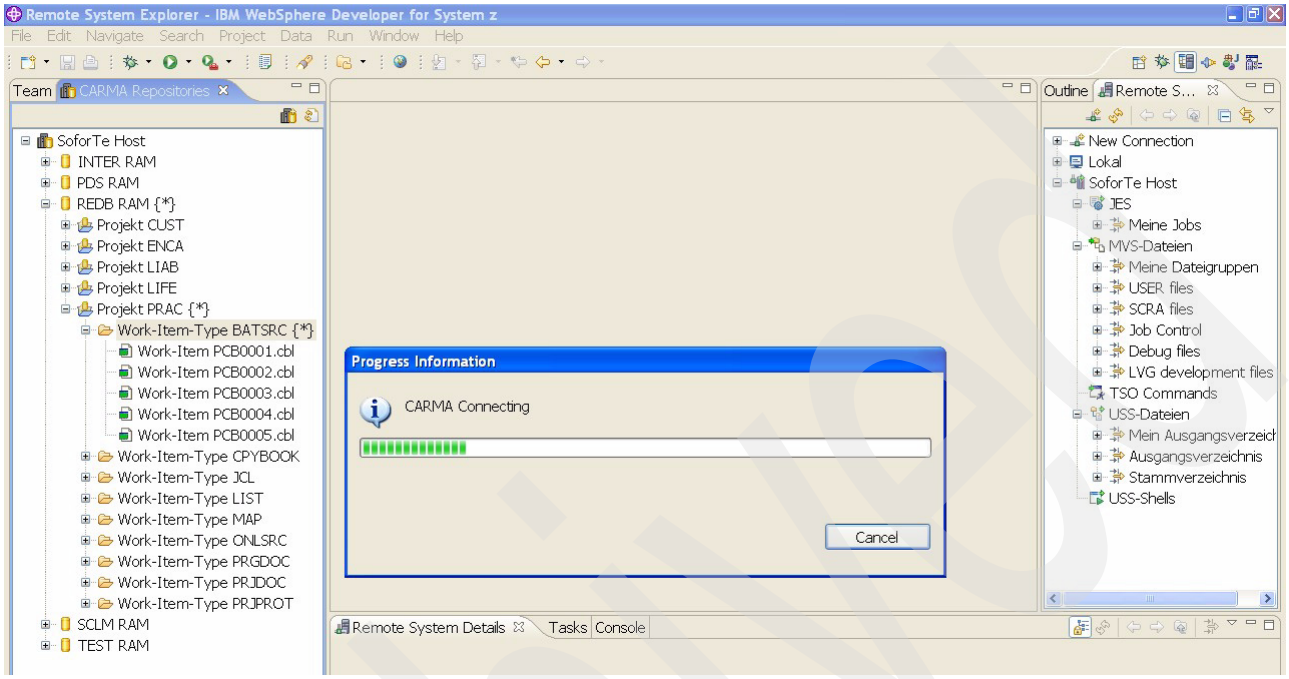


Figure 12-11 Connecting to CARMA

When we want to save the member in SCLM, we need the appropriate SCLM language. The SCLM language is a user defined parameter of the CARMA createMember action, as shown in Figure 12-12.

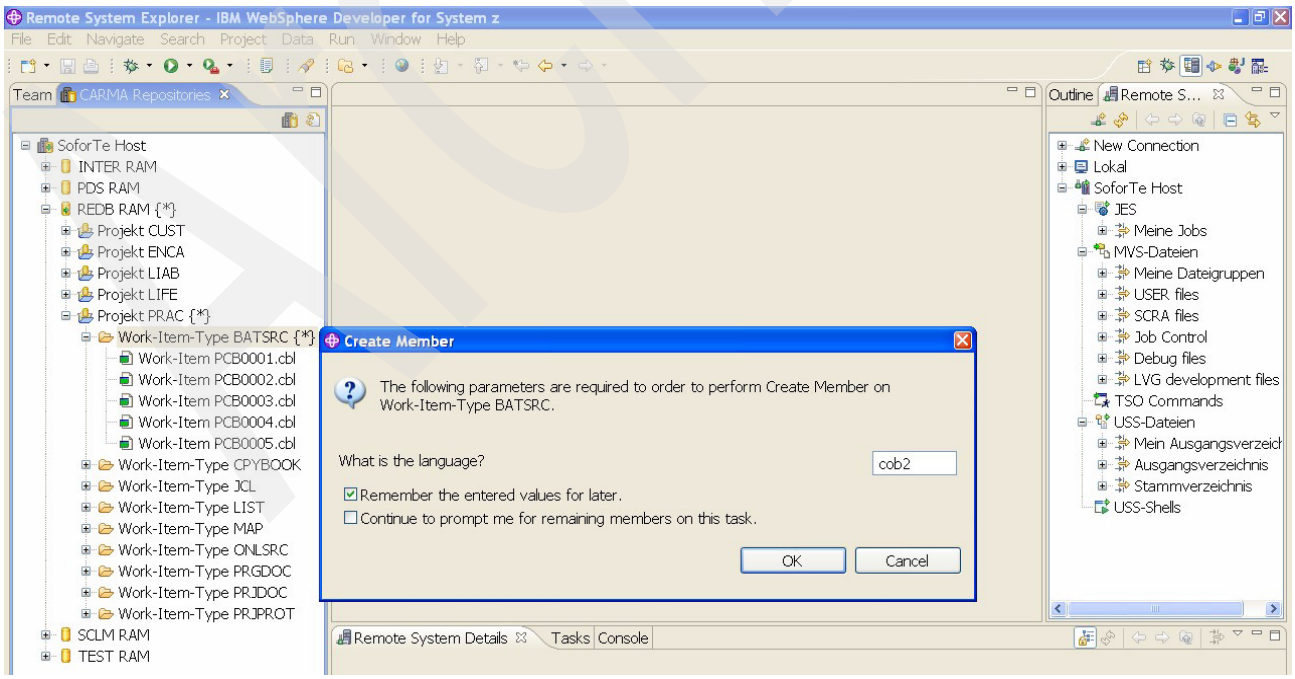


Figure 12-12 Request for the SCLM language

The new member pcb0006 appears in the CARMA view under the BATSRC work item type, as shown in Figure 12-13.

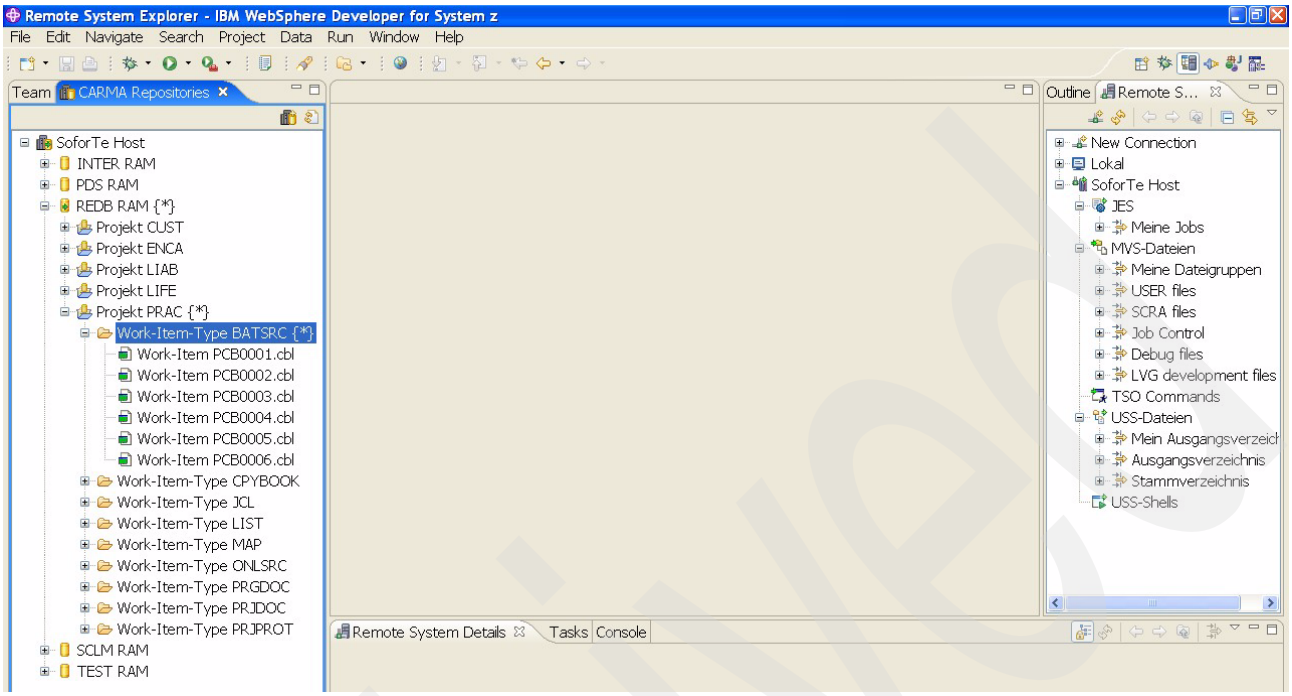


Figure 12-13 Display of the new member in the CARMA view

Archive

In order to change our newly created member, we have to perform a check-out. This will execute an SCLM lock on the member with the access key of the user, as shown in Figure 12-14.

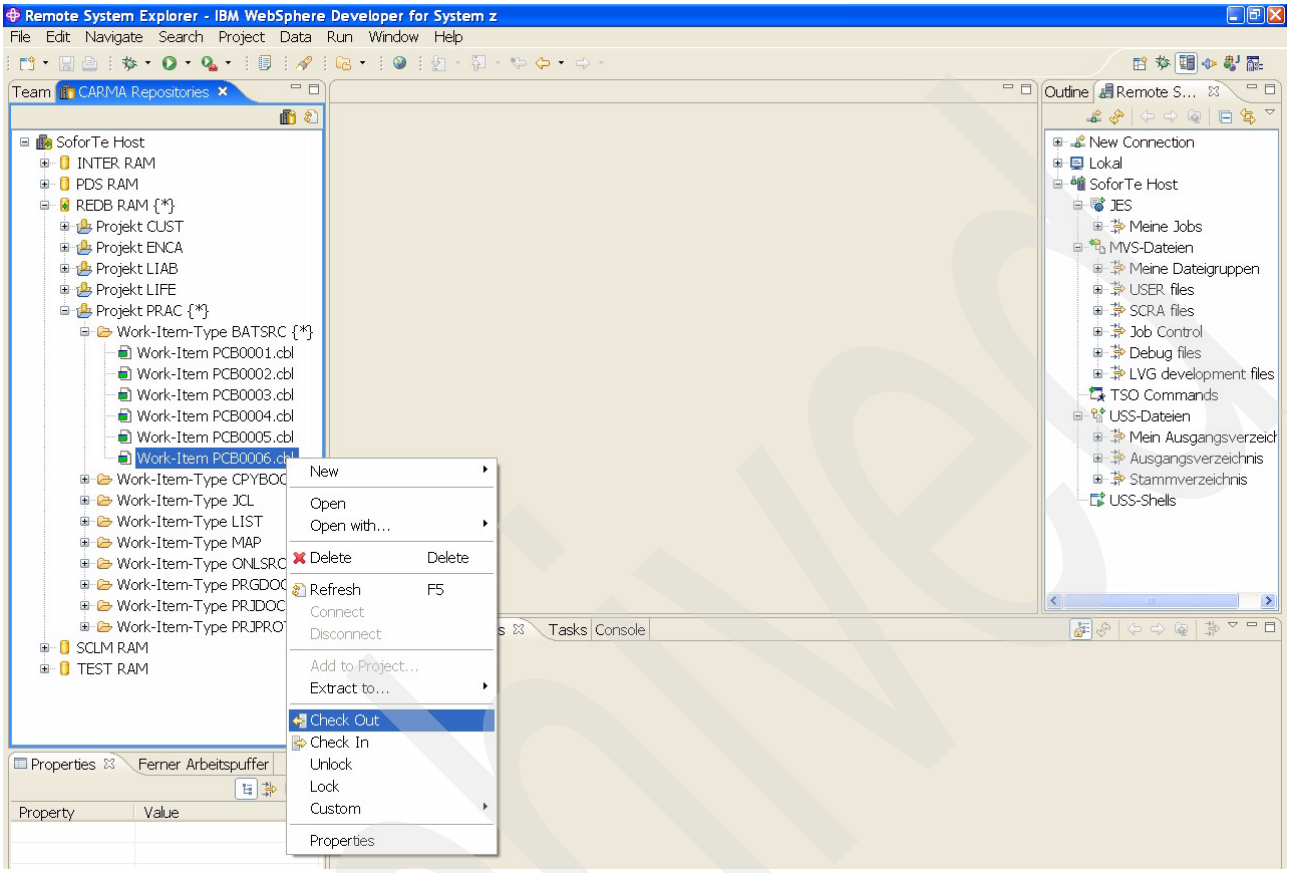


Figure 12-14 Check out the new member

Archive

We receive a message that the operation was successful, as shown in Figure 12-15.

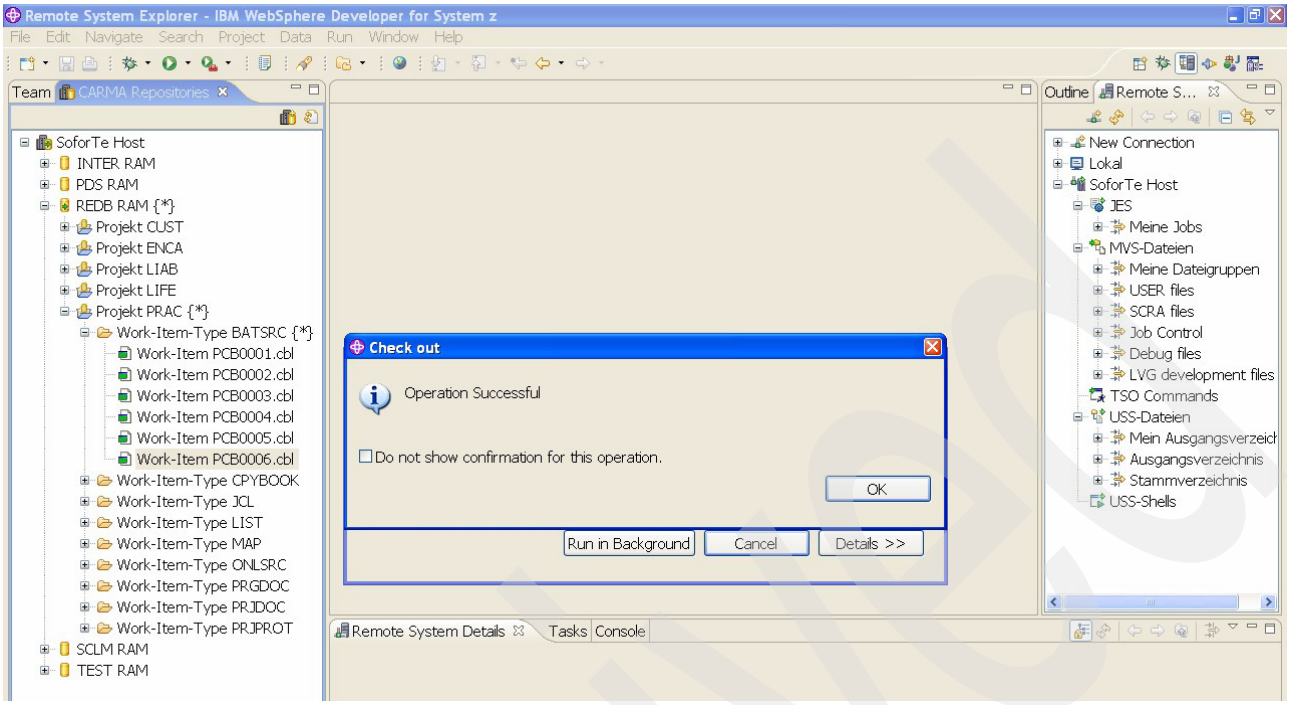


Figure 12-15 Message of success

Archive

Now we can open the member. This can be done by double-clicking the member or by selecting the **Open** action from the context menu of the member, as shown in Figure 12-16.

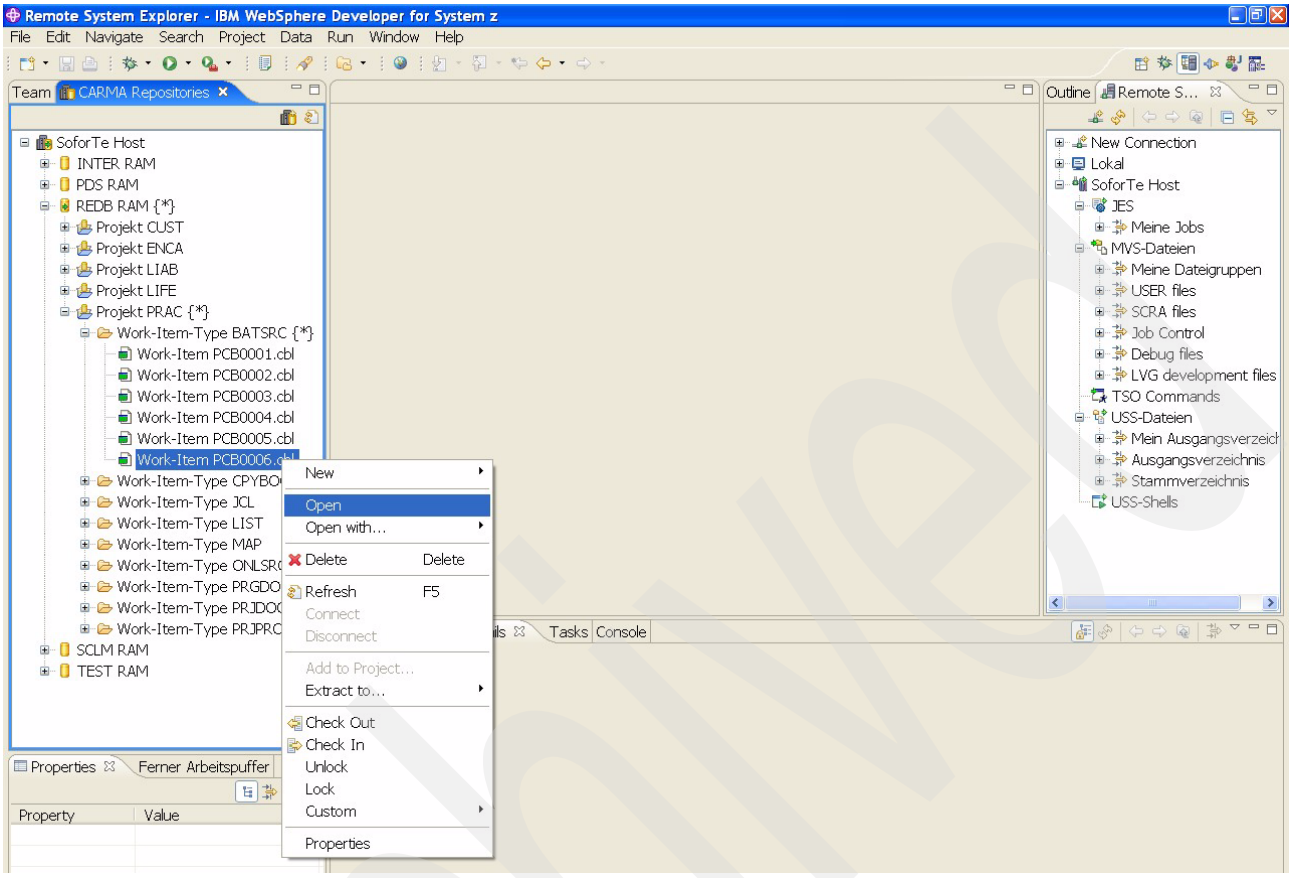


Figure 12-16 Open the number

The source of PCB0006 is fetched from the host and put into the LPEX editor. As you can see, we have defined a simple COBOL frame when creating a COBOL member, as shown in Figure 12-17 on page 443.

Archive

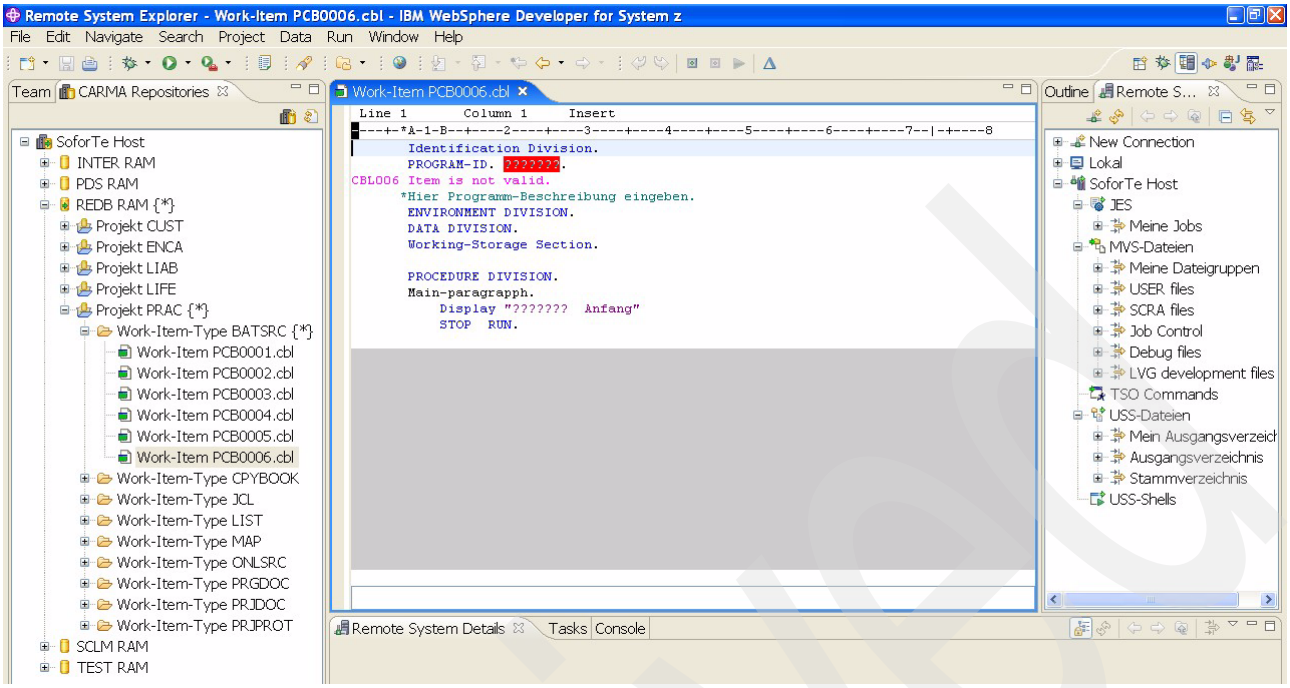


Figure 12-17 COBOL frame of the new member in the LPEX editor

We do some changes to the member, as shown in Figure 12-18.

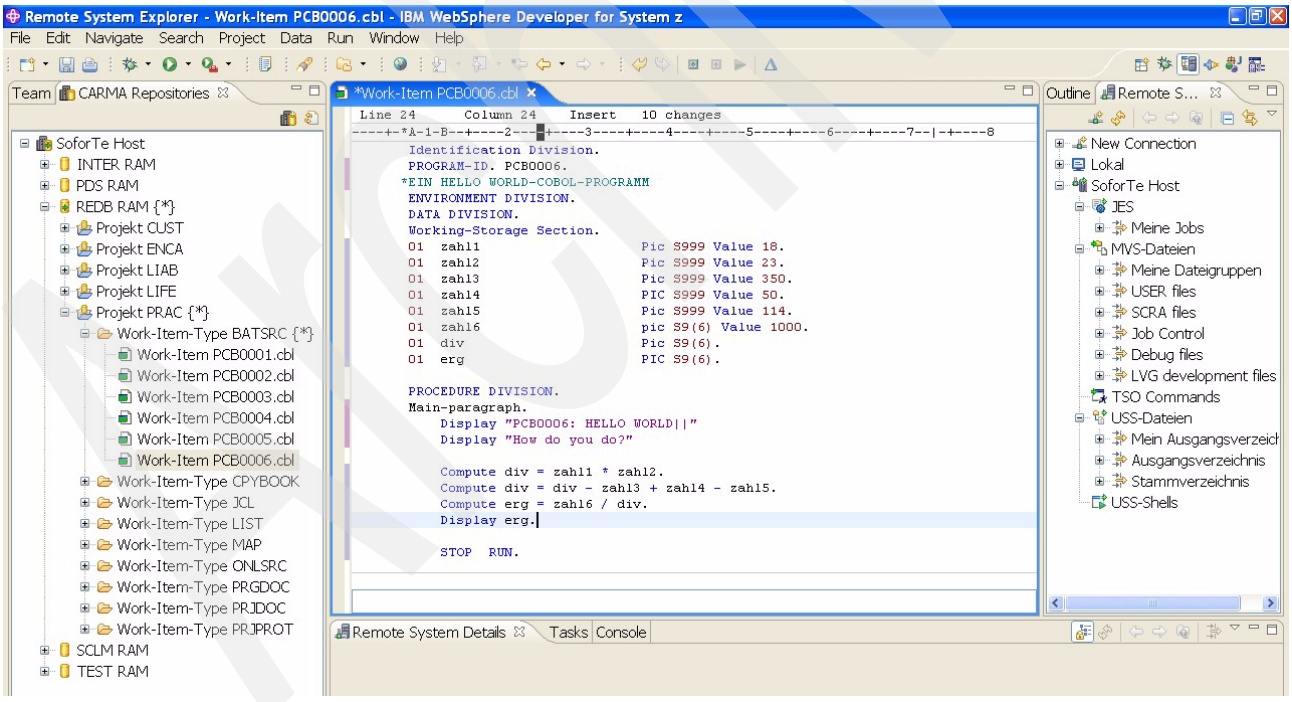


Figure 12-18 Changed member in the LPEX editor

We save the member by pressing Ctrl+S or clicking **Save** in the context menu of the member, as shown in Figure 12-19.

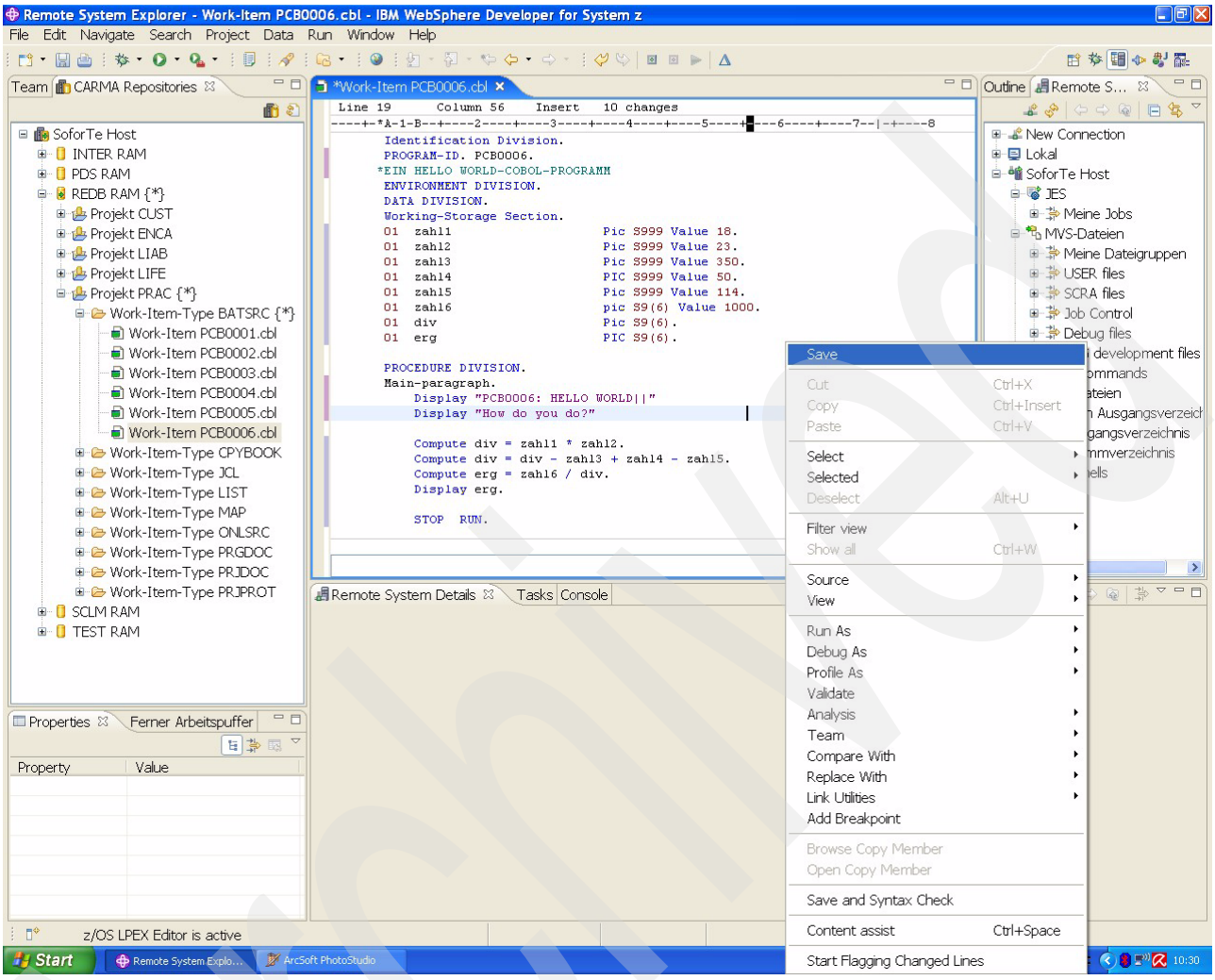


Figure 12-19 Save the changed member

The member is transferred to the host and saved in a temporary user data set. We get a message that the execution of the operation was successful, as shown in Figure 12-20.

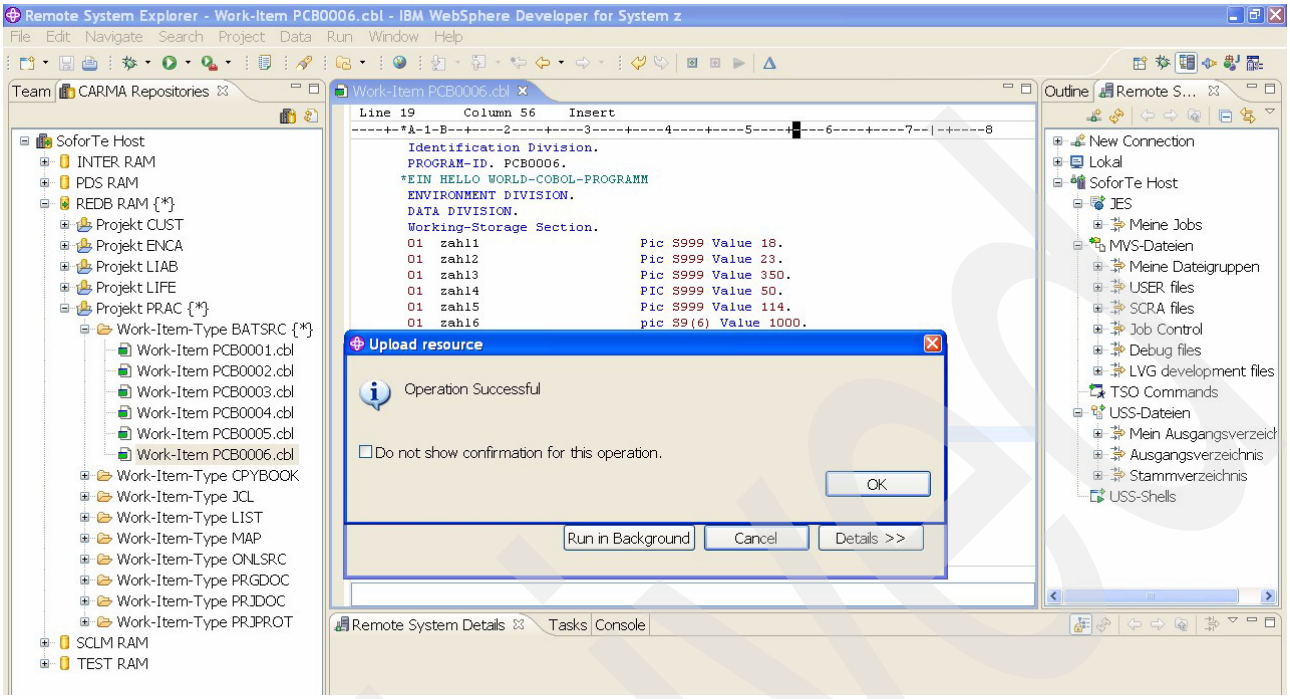


Figure 12-20 Message of success

Archive

When we decide that our changes are complete, we have to transfer the member back into SCLM. This is done with the CARMA check-in function. This function copies the member from the temporary user data set to the appropriate SCLM development data set and does an SCLM save and an unlock of the member, as shown in Figure 12-21.

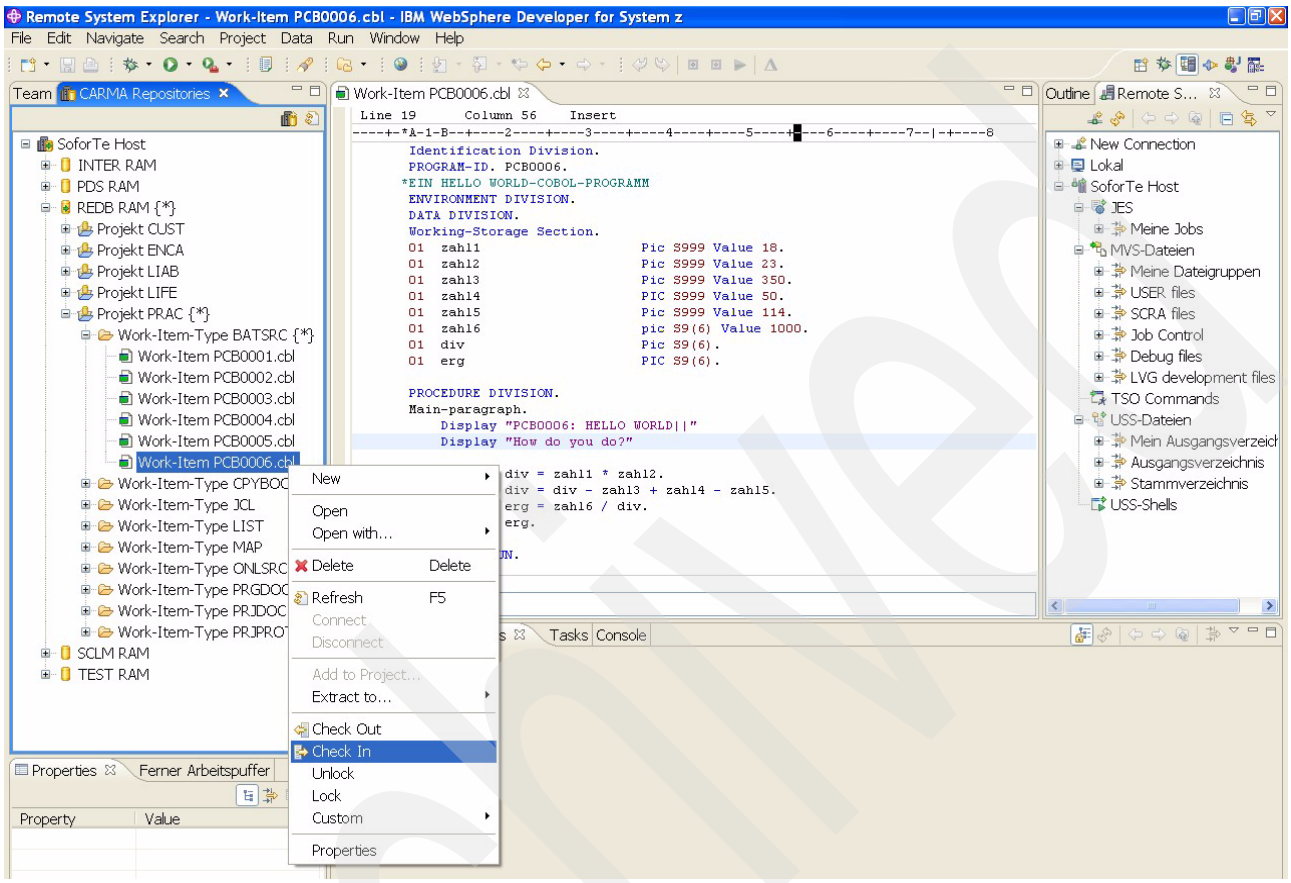


Figure 12-21 Check in the changed member

Archive

Again, we receive a message of success, as shown in Figure 12-22.

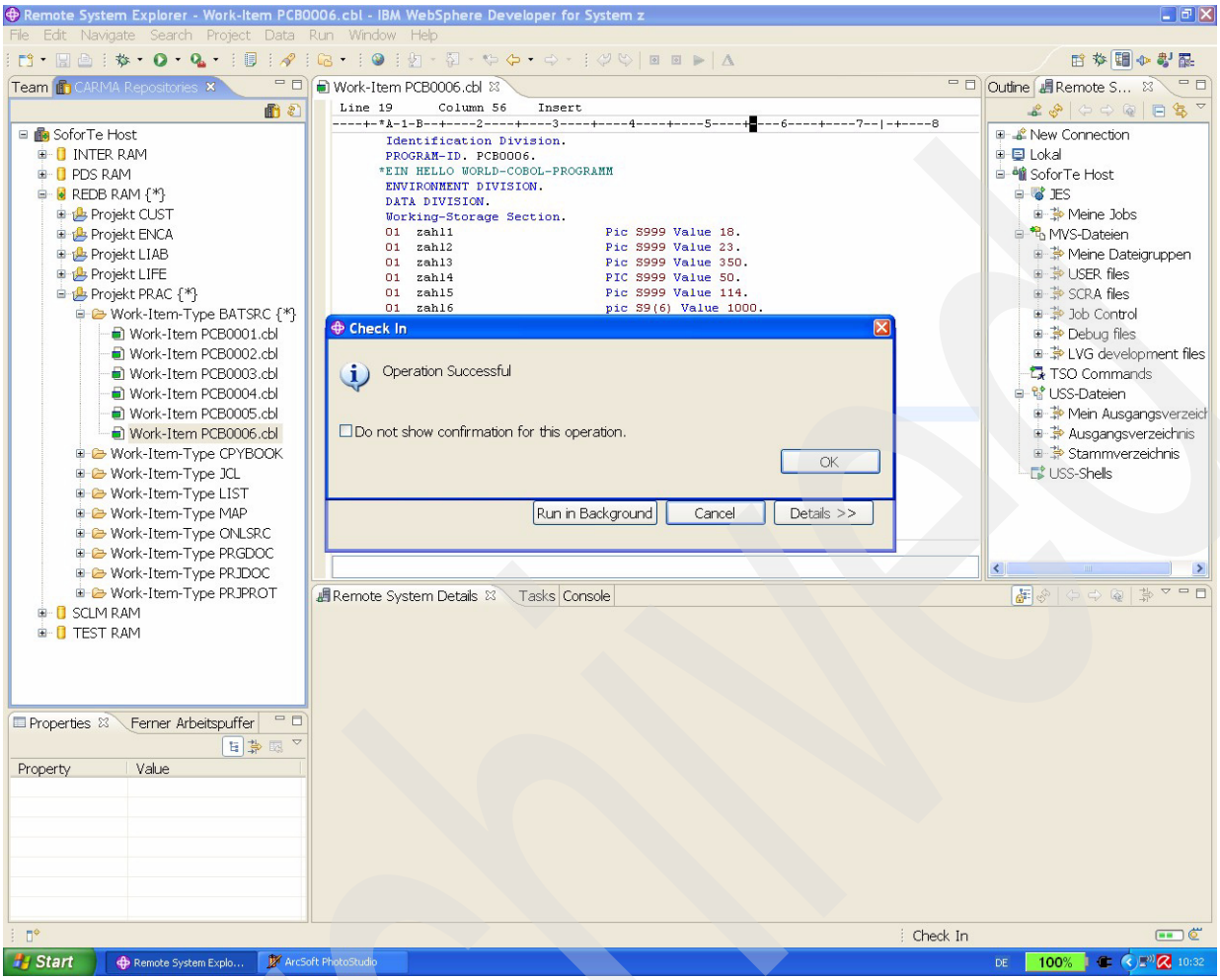


Figure 12-22 Message of success

Archive

Now we can build the member. The build action is our sole custom action. For simplification reasons, the build is not an SCLM build. We just submit a batch job which compiles and links our member, as shown in Figure 12-23.

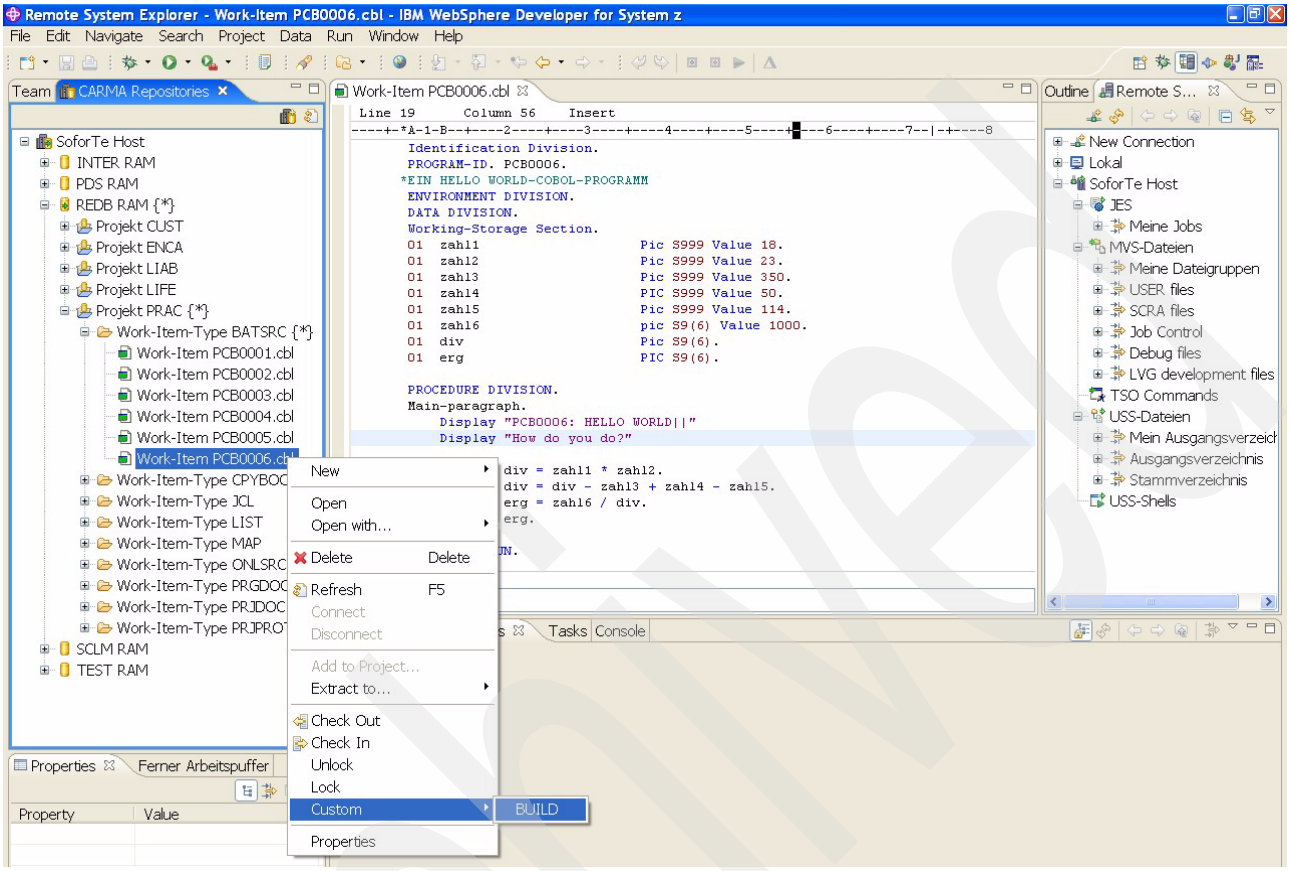


Figure 12-23 User defined action build

In order to incorporate the definition of a custom parameter into a custom function, we have defined the parameter “SCLM language”, which is requested in the next window. Of course, as we do not do an SCLM build, this parameter is meaningless in our RAM implementation, as shown in Figure 12-24 on page 449.

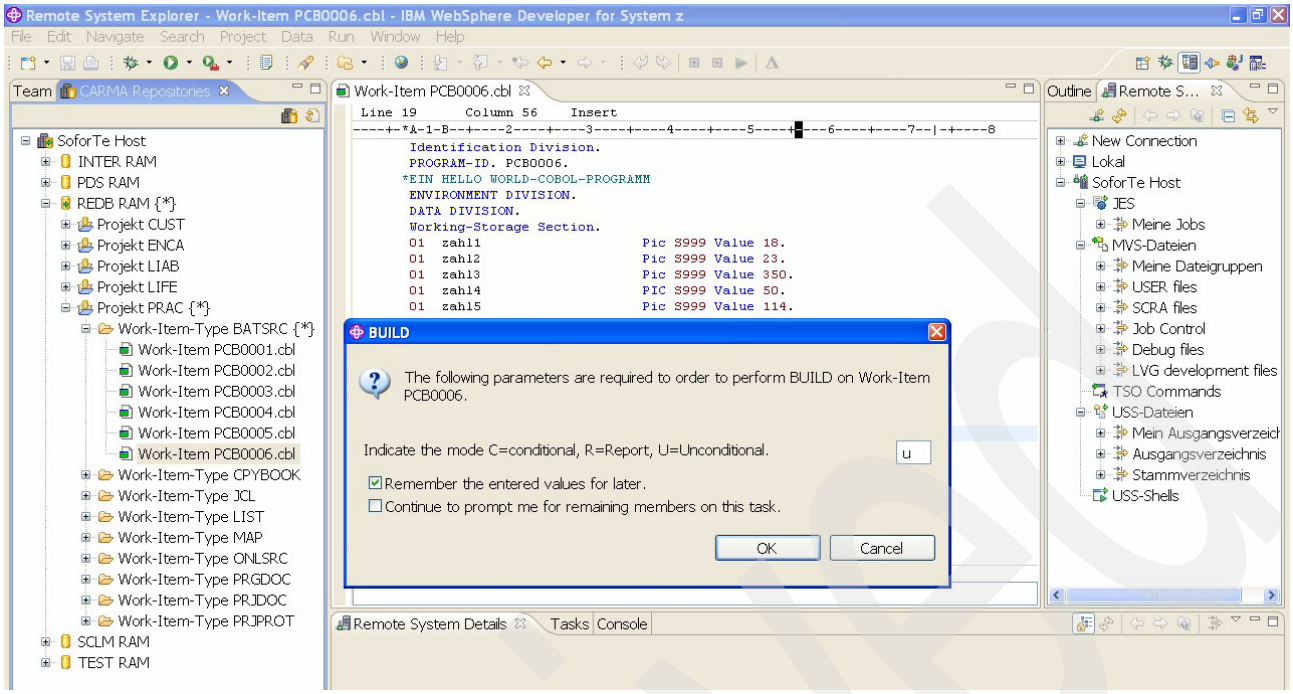


Figure 12-24 Request for the build mode

We receive another success message, as shown in Figure 12-25.

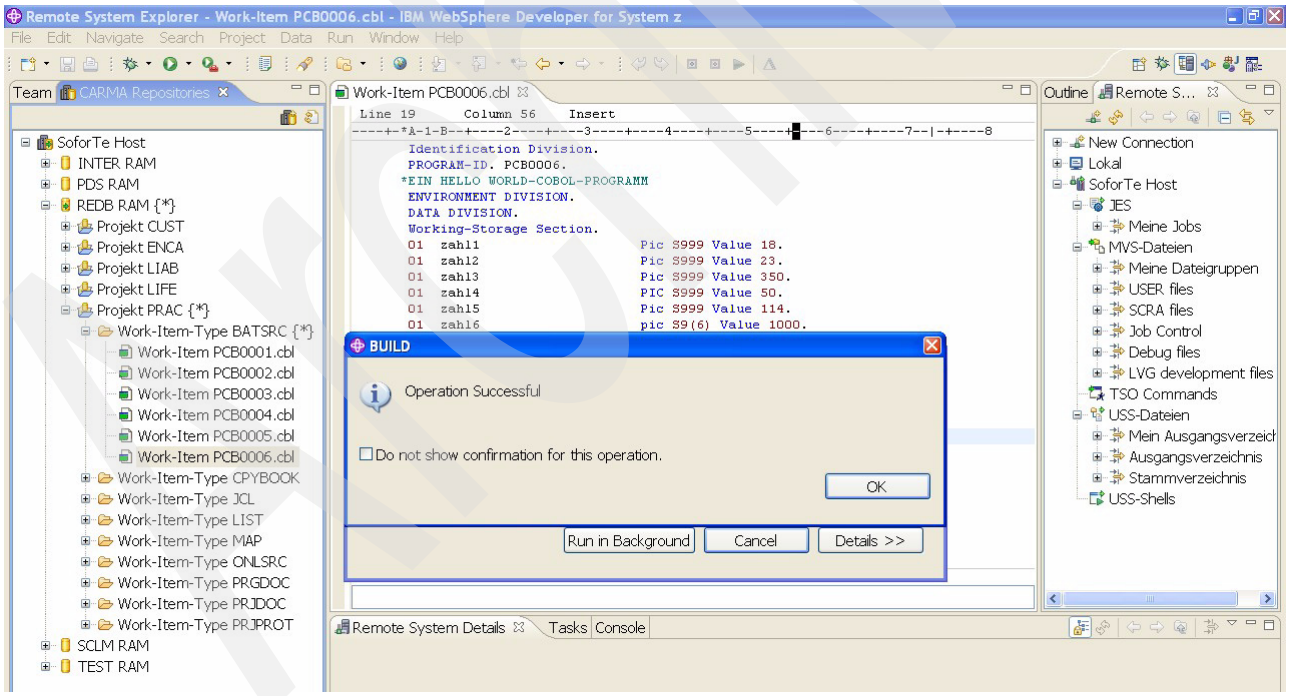


Figure 12-25 Message of success

The job output is found in the JES display of the Remote Systems view. Before we can see it, we have to refresh the JES display, as shown in Figure 12-26.

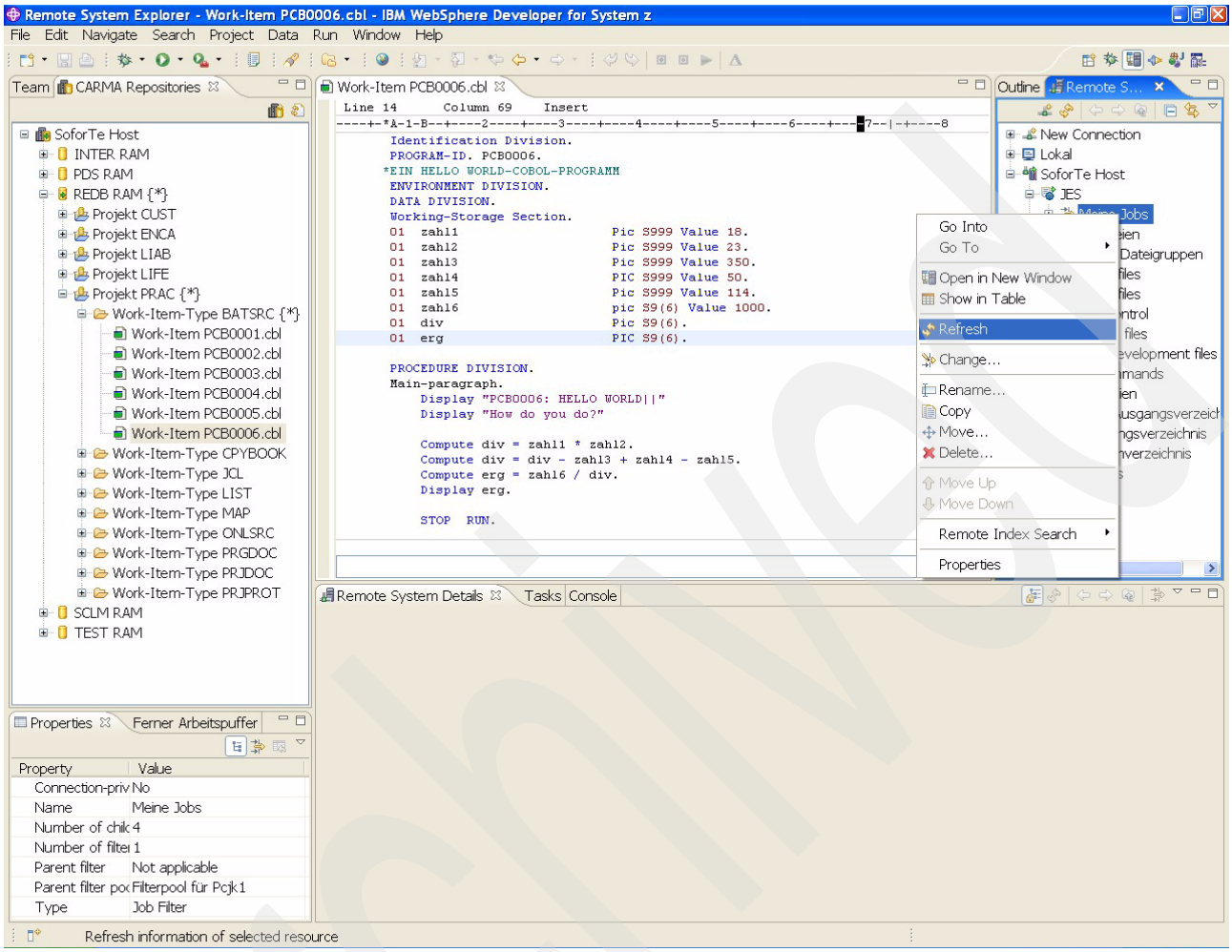


Figure 12-26 Refresh the JES display

Archive

There is an IBMUSERC job that contains the compile/link output. We expand it to see the output of the numerous job steps. We take a look at the compile output, as shown in Figure 12-27.

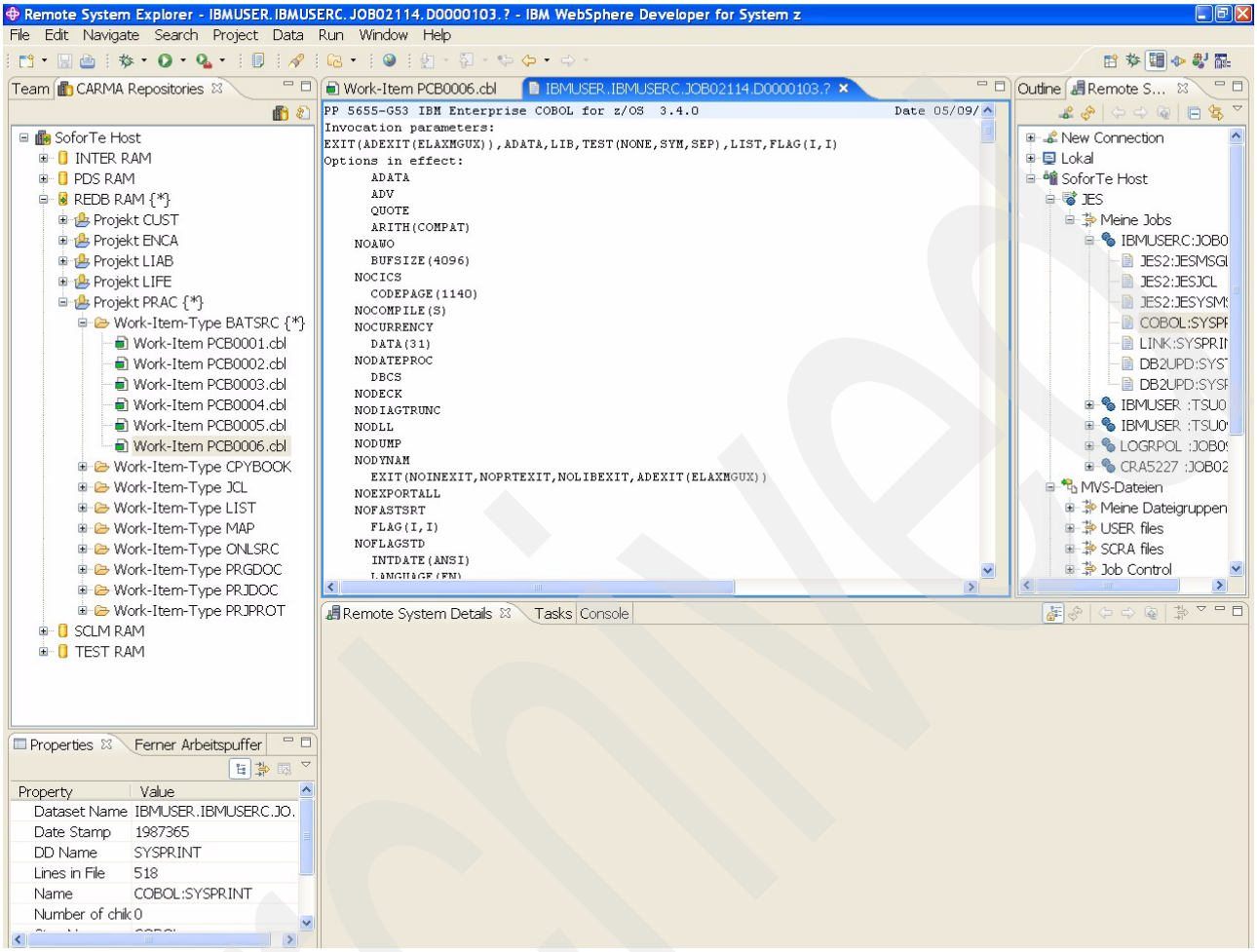


Figure 12-27 Compile output

Archive

At the end of the output, we see the compilation run with a return code of 0, as shown in Figure 12-28.

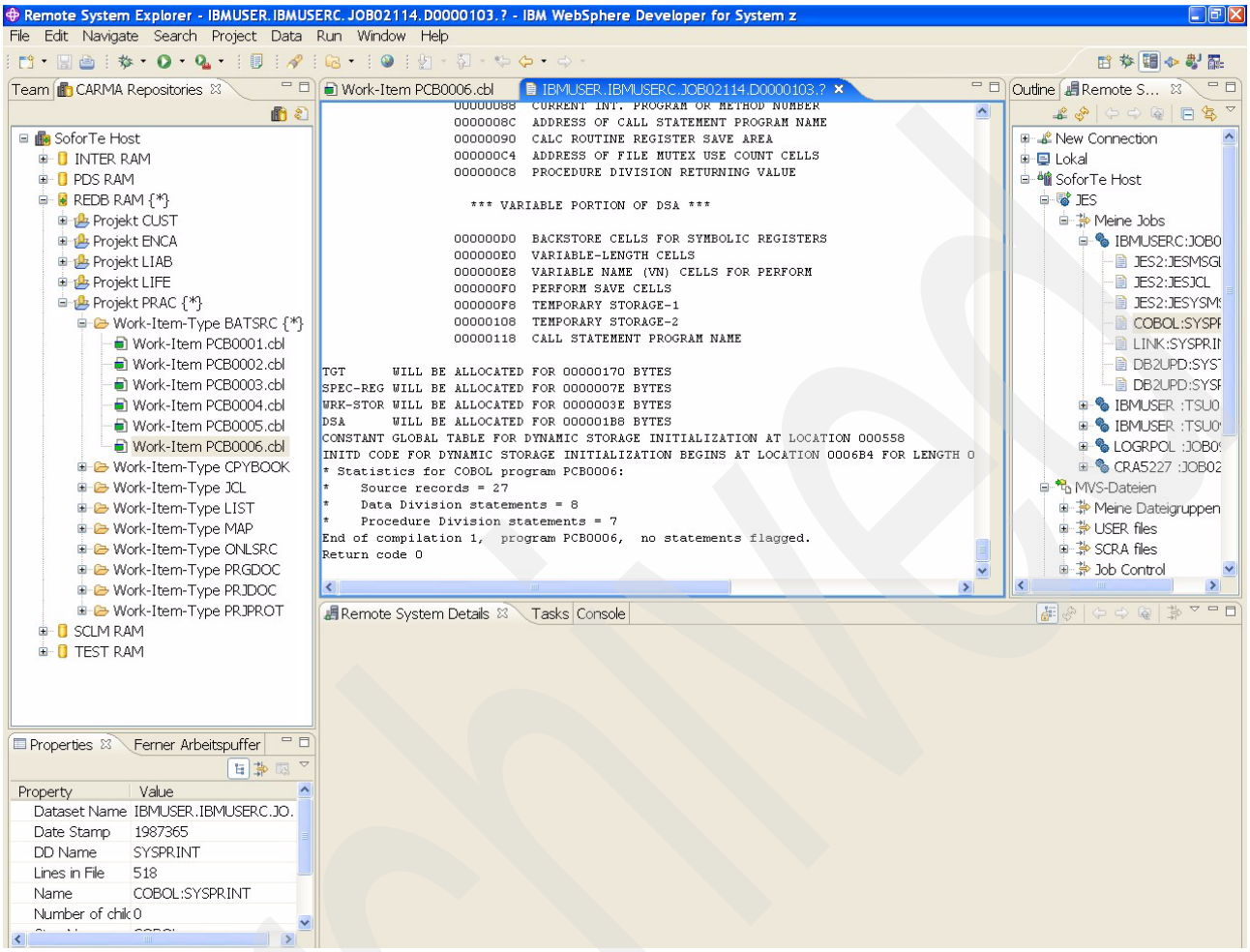


Figure 12-28 Return code 0 of compile step

Archive

Looking at the linkage output shows that the link was also successful, as shown in Figure 12-29.

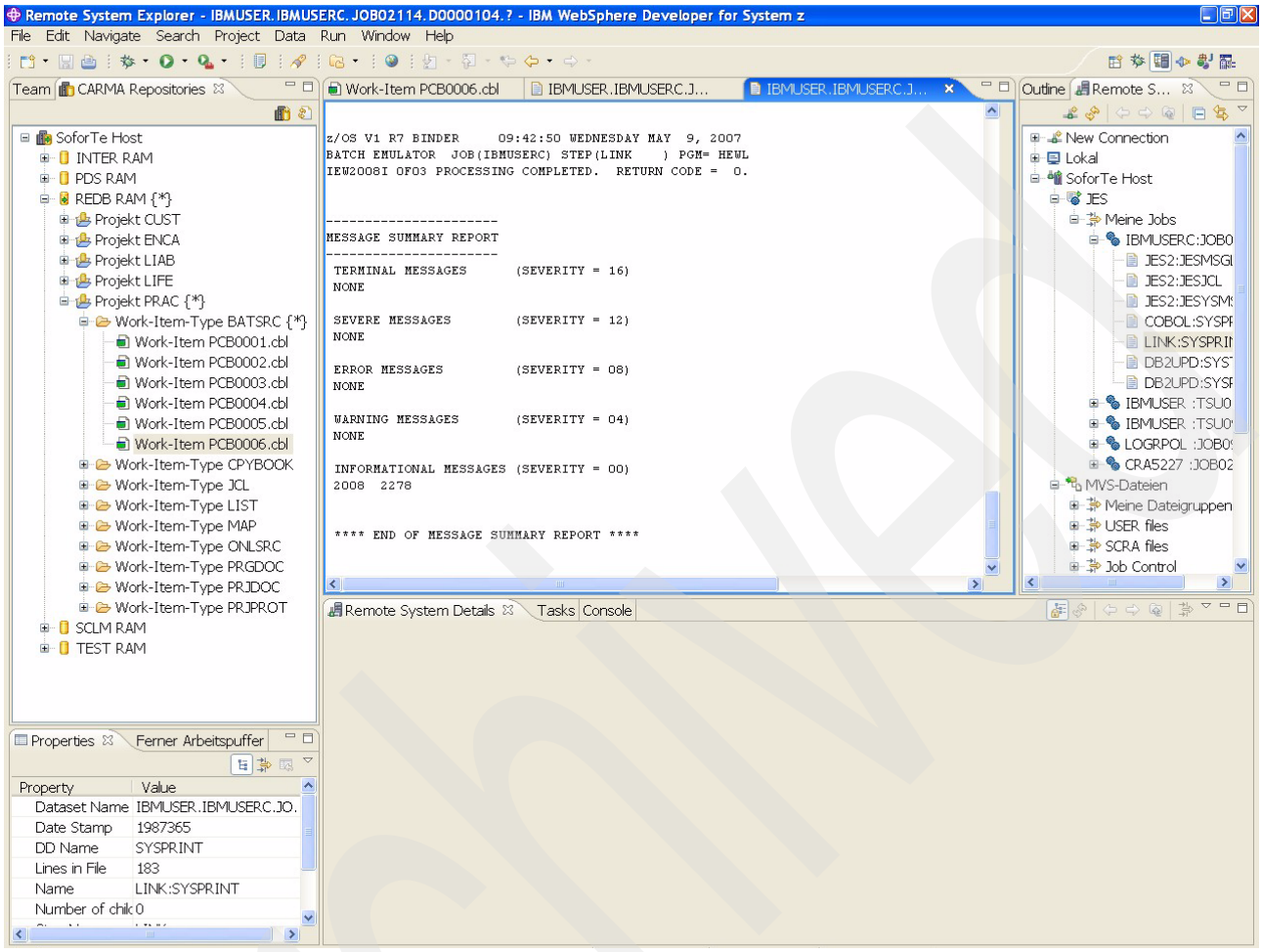


Figure 12-29 Message summary of the link step

Archive

In our JCL data set, we find a job to execute the program. We navigate to the appropriate data set on the MVS data set display of the Remote Systems view. There we find the job PCB0006, which we can execute by choosing **Submit** from the context menu, as shown in Figure 12-30.

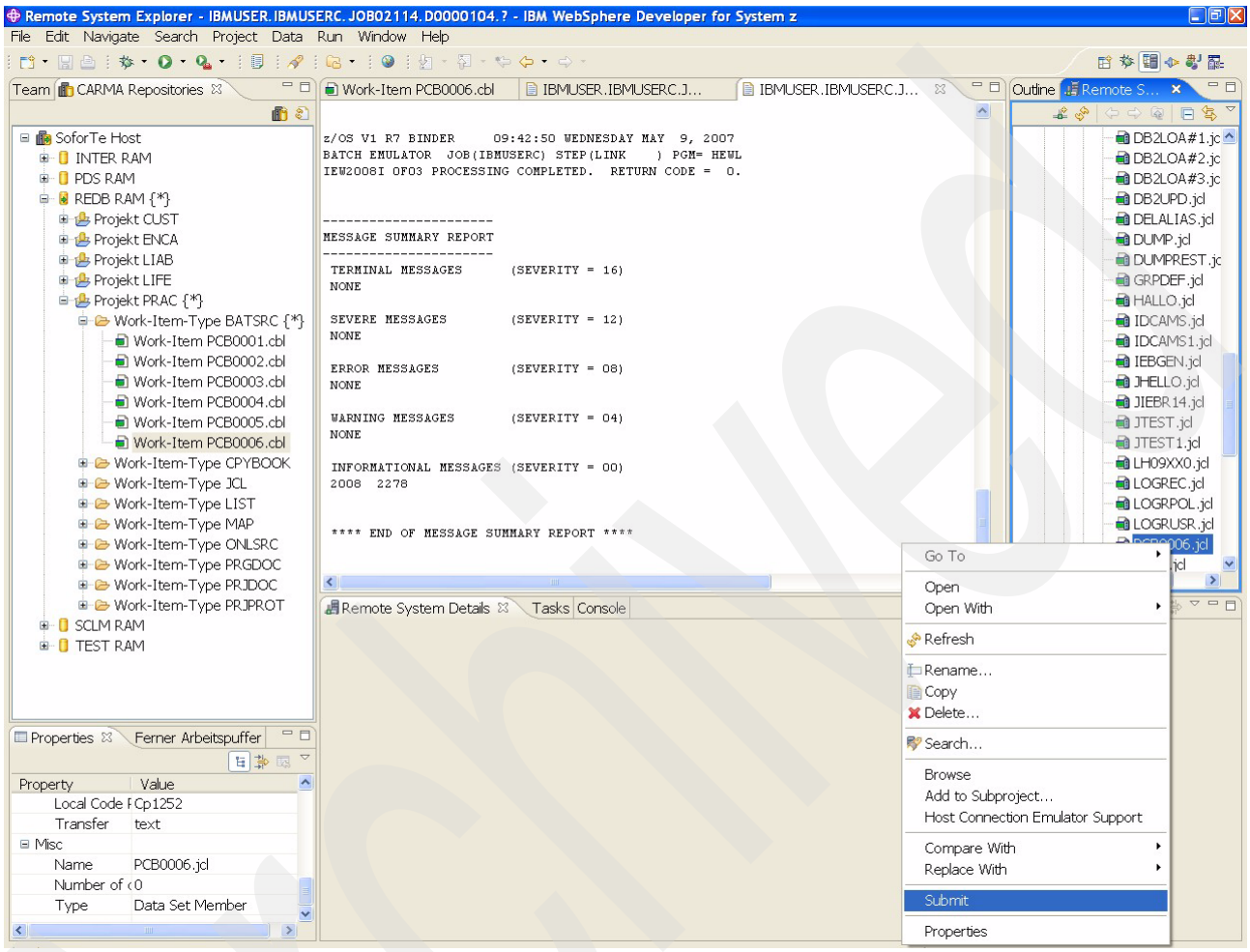


Figure 12-30 Submission of the PCB0006 test batch job

Archive

We get a message telling us that the job output can be found under the JES display of the Remote Systems view, as shown in Figure 12-31.

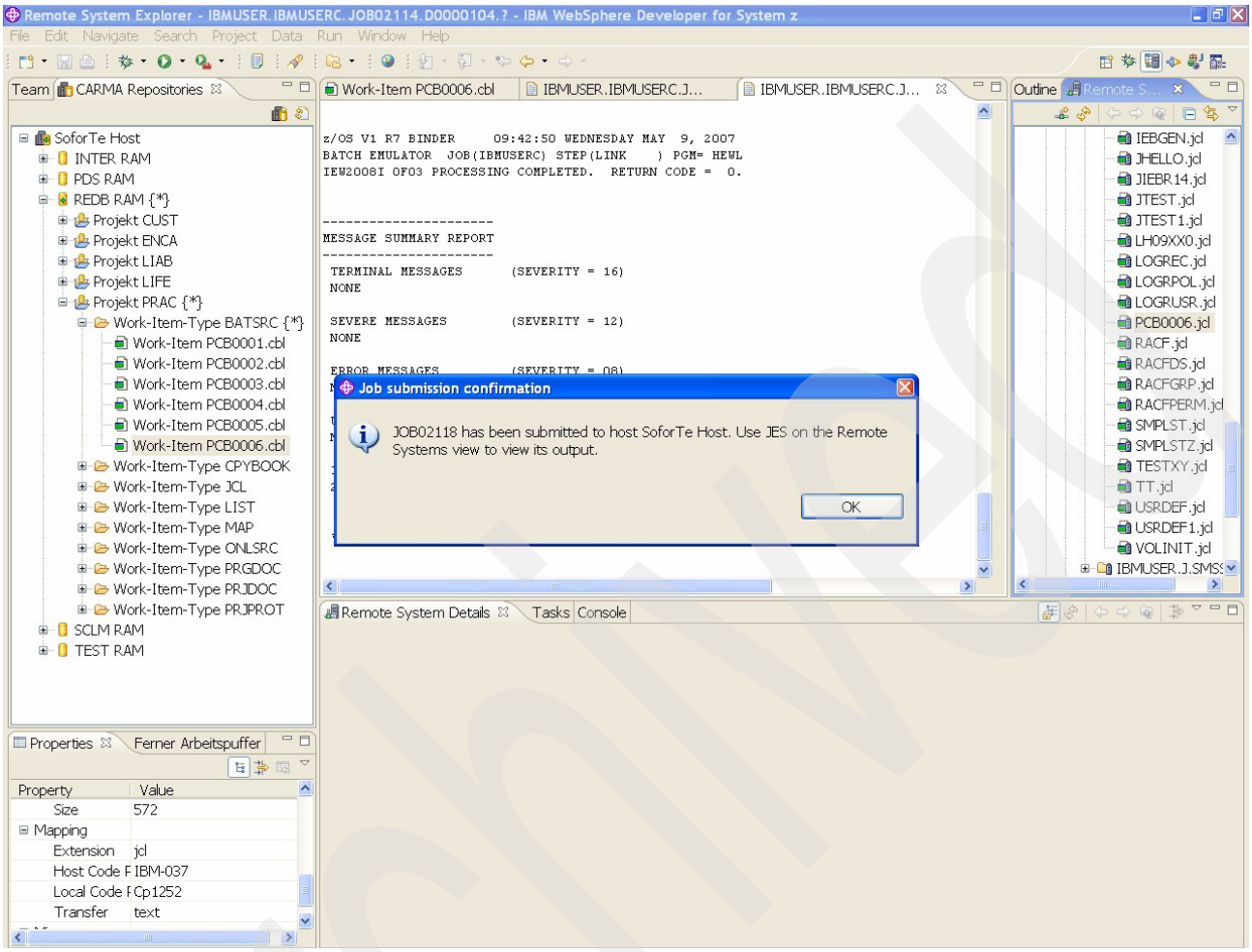


Figure 12-31 Confirmation message of the submission

Archive

We have to refresh the JES display in order to see our current job. When expanding the job output, we see that it has produced a dump, as shown in Figure 12-32.

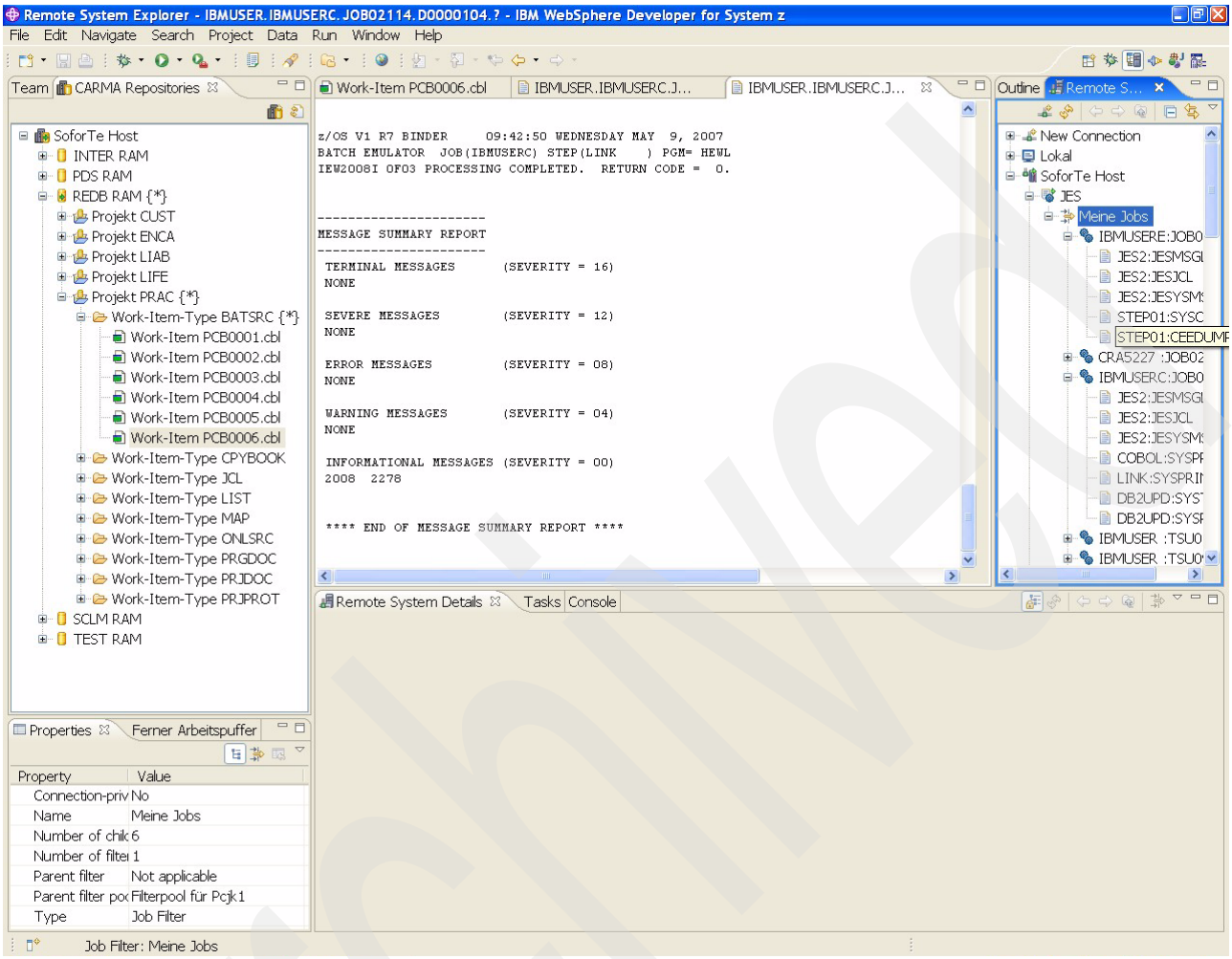


Figure 12-32 Job output with a dump DD

We can take a closer look at the job output to discover what the problem was. However, we prefer to debug the job immediately. Here we want to show the integration of the IBM debugger into the WD/z development environment because we believe it really is a great benefit. Your program, which is running on the host, can be debugged in the WD/z perspective with all the advantages of a GUI display. Moreover, if you have a COBOL program that calls a Java subroutine for some front-end processing, the debugger will smoothly enter the Java subroutine without a visible break. You only have to update your debug parameter file on the host and then re-submit the job.

In the parameter file, you have to enter the name of the program and check the IP address of your workstation, as shown in Figure 12-33.

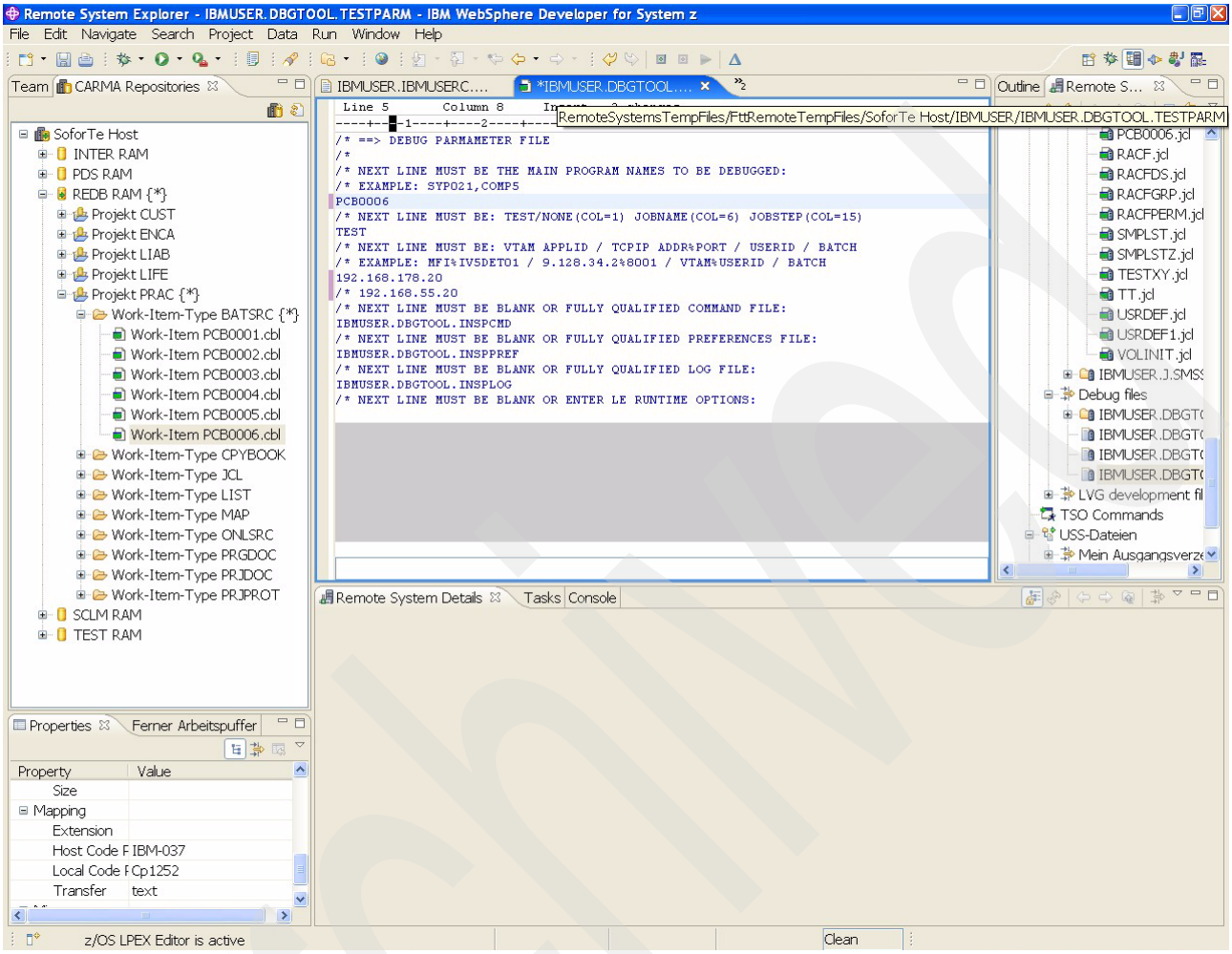


Figure 12-33 Debug parameter file

After these changes to the debug parameter file, we have to save the file by pressing Ctrl+S on the host. Then we can re-submit the job, as shown in Figure 12-34.

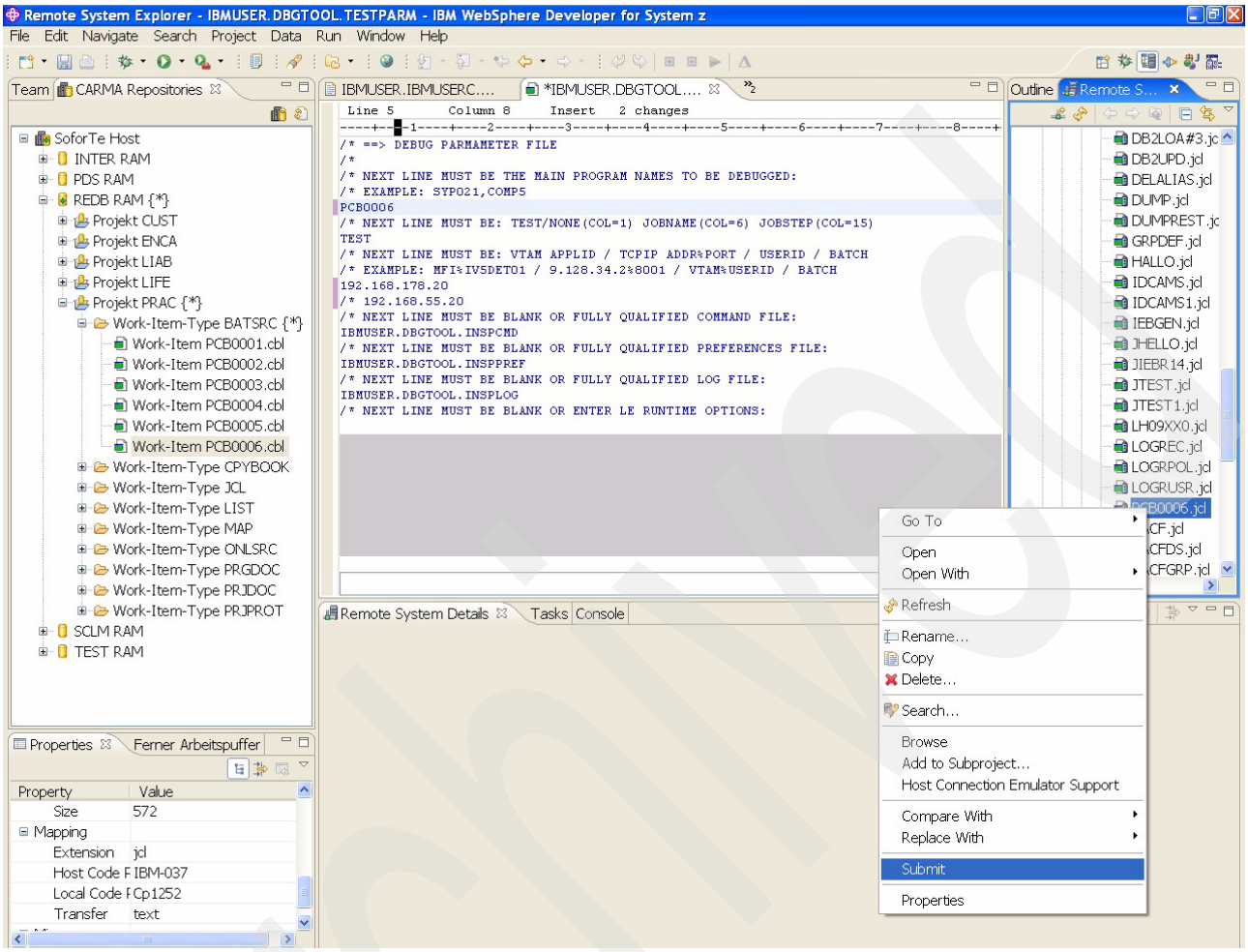


Figure 12-34 Resubmit the PCB0006 test batch job

Archive

After the message appears that tells us that the job output can be found under the JES display of the Remote Systems view, we are asked if we want to switch into the debug perspective, as shown in Figure 12-35.

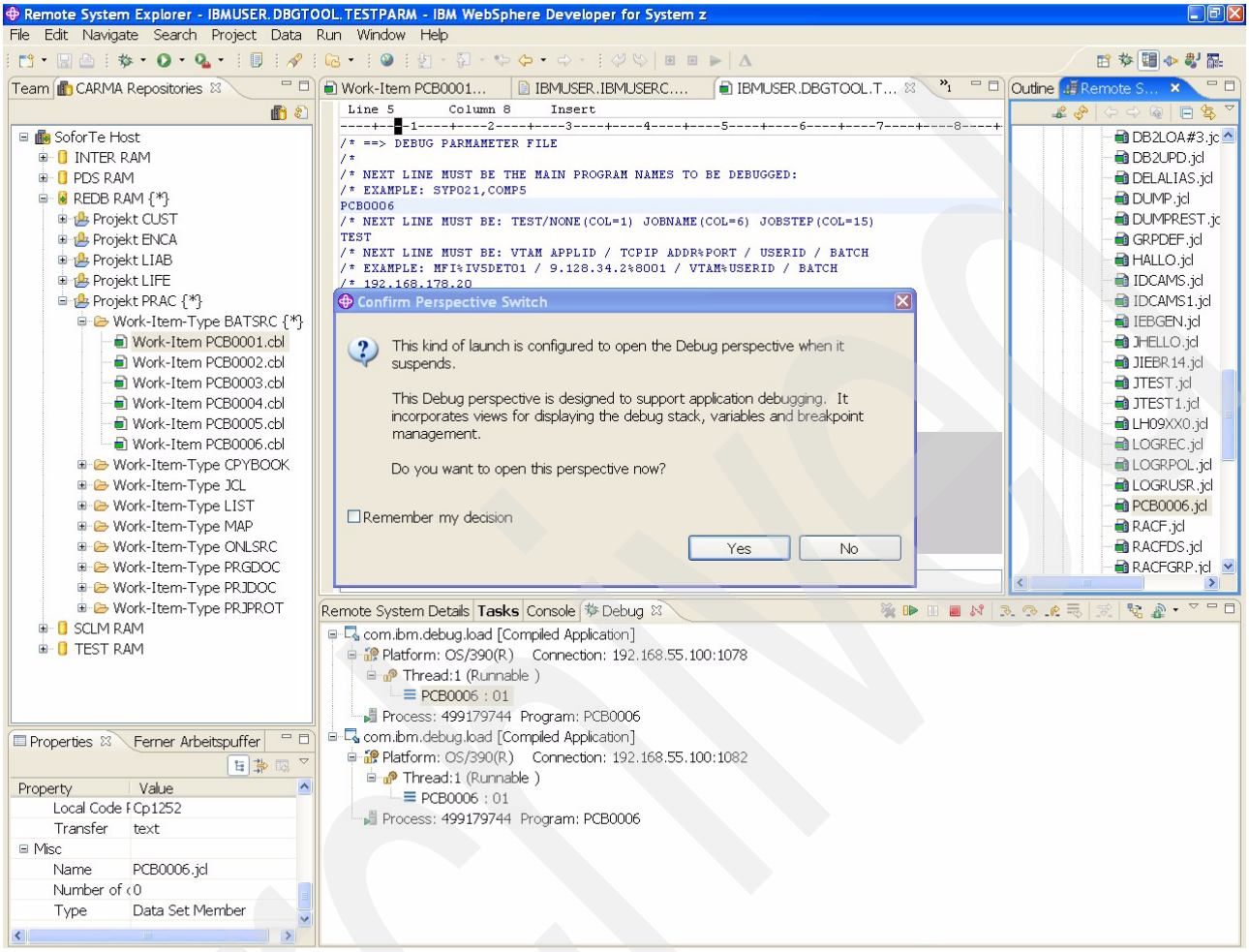


Figure 12-35 Request for perspective switch

Archive

When we click the **Yes** button, we go directly to the debug perspective with our PCB0006 program loaded for debugging. We can now select the **Resume** option on the menu bar of the debug perspective to get more information where the abend of our program has occurred, as shown in Figure 12-36.

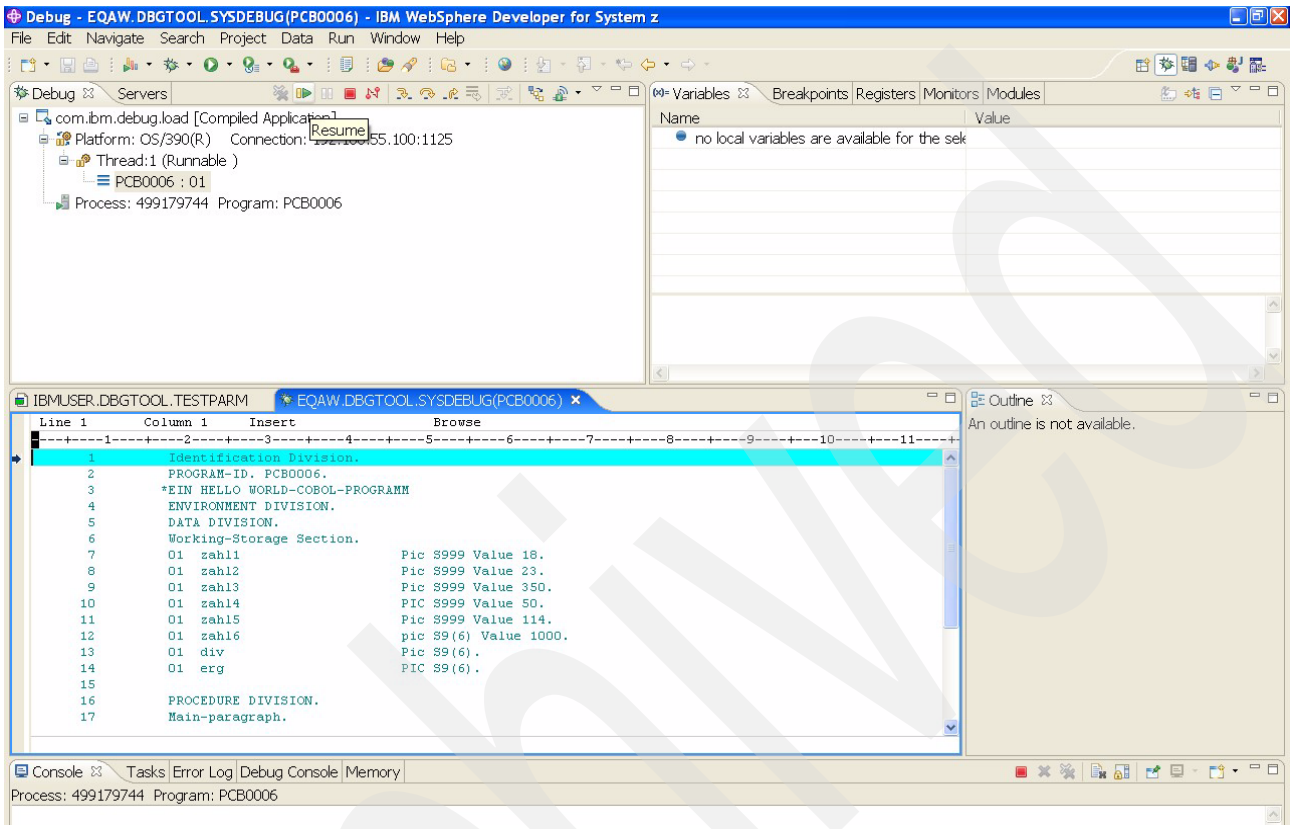


Figure 12-36 Debug perspective with PCB0006 loaded

Archived

As we can see, the program detected a decimal-divide exception. The variable “div” has the value 0, as we can see in the variable display on the right, as shown in Figure 12-37.

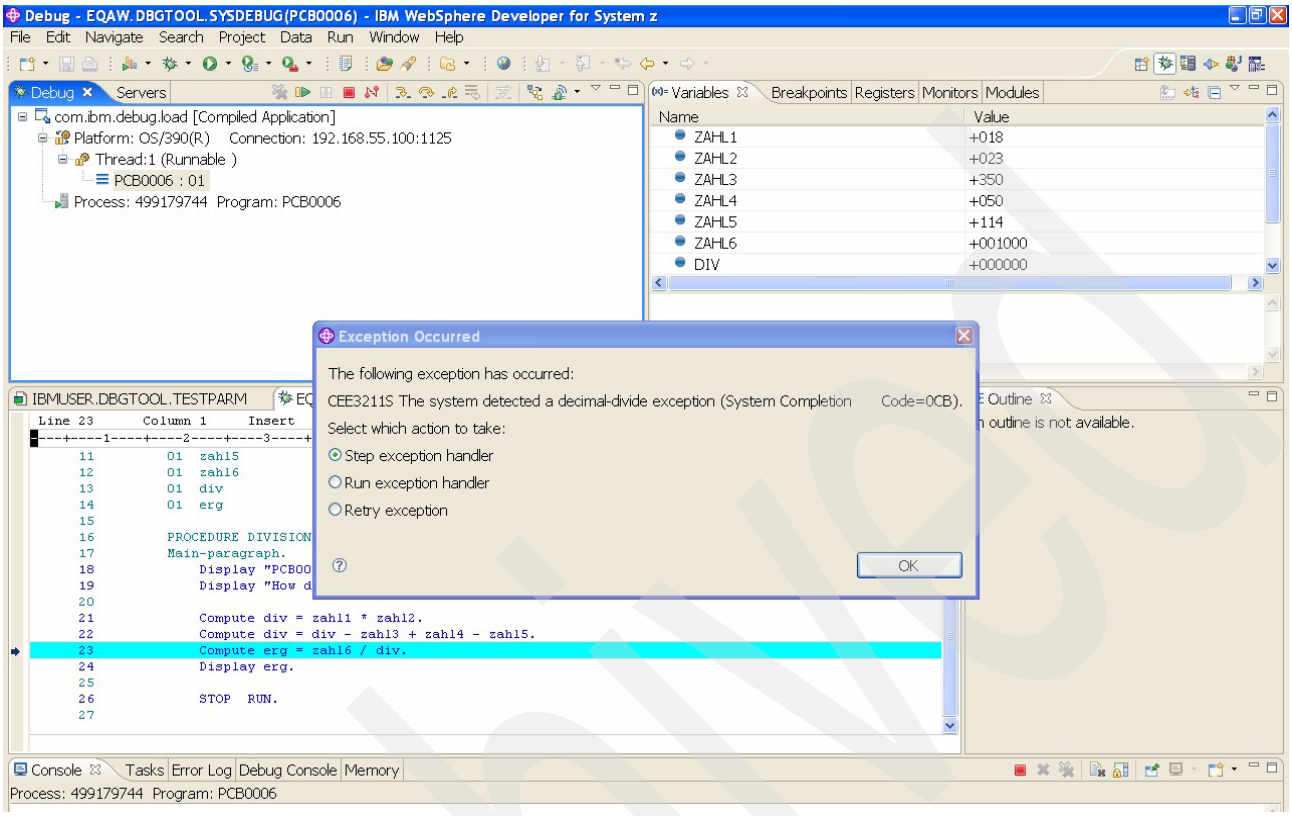


Figure 12-37 Description of the occurring exception

Archive

When we click the **OK** button, the debug thread ends, as shown in Figure 12-38.

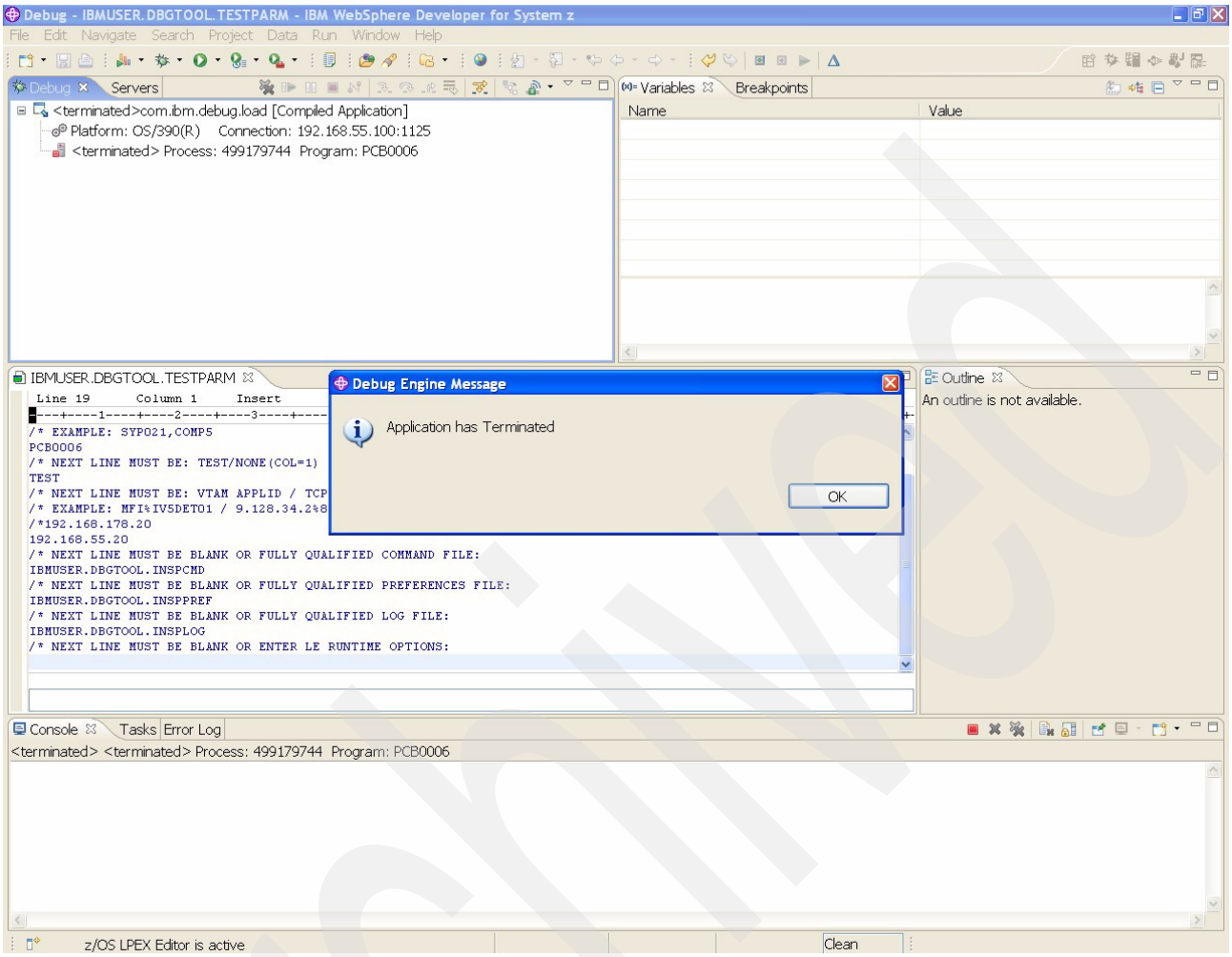


Figure 12-38 Debug thread termination message

Archived

13

IBM System z server components

To remotely develop on an IBM System z server, multiple components, delivered in WD/z, need to be installed and configured on the host. These components, which we describe in this chapter, work together with the WD/z client to enable WD/z for host development in a remote development mode. Architecturally, these host components interface to subsystems on System z and provide the interface for the WD/z front end.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

13.1 Packaging differences of WD/z V6

In WD/z Version 6, there were six Function Modification Identifiers (FMIDs) containing the server components. To reduce the confusion about what needs to be installed, the servers have been consolidated. Three of the server components are found bundled in FMID HHOP700. They are:

- RSE
- Job Monitor
- TSO Command Server

The server component CARMA as well as SCLM Developer Toolkit have their own FMIDs:

- CARMA: HCMA700
- SCLM Developer Toolkit: HSD1210

13.2 Remote System Explorer (RSE)

Initially developed for the AS/400®, Remote System Explorer (RSE) has since been ported to the Intel® platform, IBM eServer™ pSeries®, and finally System z. Since 2Q 2007, there is even an RSE version available on open source systems.

RSE provides in WD/z the transport mechanism for data between the host and the client, except in the case of remote debugging. Remote debugging builds its own socket to socket connection between the Debug Engine client and the debug front-end server. All other data handled by WD/z, whether output queue information (JES), source files views, or CARMA communications. utilize RSE.

Therefore, it is within your RSE connection where page code information is required and Double Byte Character Set (DBCS) can be enabled. Working in tandem with RSE is an additional technology that also provides bi-directional source files. This is necessary for languages such as Hebrew, which read from right-to-left instead of left-to-right. Countries that do not need this support do not need to configure it. The same applies to DBCS support.

Because RSE is the interface to TCP/IP, it is within RSE where a user must define his z/OS TCP/IP address (system name) as well as the port number that the RSE daemon is listening to. Any WD/z TCP/IP requests coming to the host are handed off by the inetd daemon from TCP/IP to RSE, the RSE service having been defined in the inetd.conf configuration file. There is also the ability to use REXEC for the hand off of inbound and outbound messages, but, in practice, almost all installations use the RSE daemon.

13.3 Job Monitor

The Job Monitor is the server component responsible for providing an interface to the System z JES spools. Security in WD/z is the same as in SDSF, meaning JES security manager. What a developer could not do in SDSF he also cannot do in WD/z. An additional function of the Job Monitor is access to the JES internal reader. Through the Job Monitor, completed and executing jobs are available in WD/z as well as the ability to perform job submission. Jobs submitted from WD/z are sent through RSE to the Job Monitor server, which submits them to the internal job reader. The job monitor accepts the job number determined by JES and sends this information back to the WD/z client. The job output can be seen in the Remote

System view. This view is kept current through polling of the job monitor, but it can be, and normally is, refreshed through a direct request from the developer. Through the view, normal actions are available in SDSF, such as:

- Purge
- Cancel
- Hold
- Release

as shown in Figure 13-1.

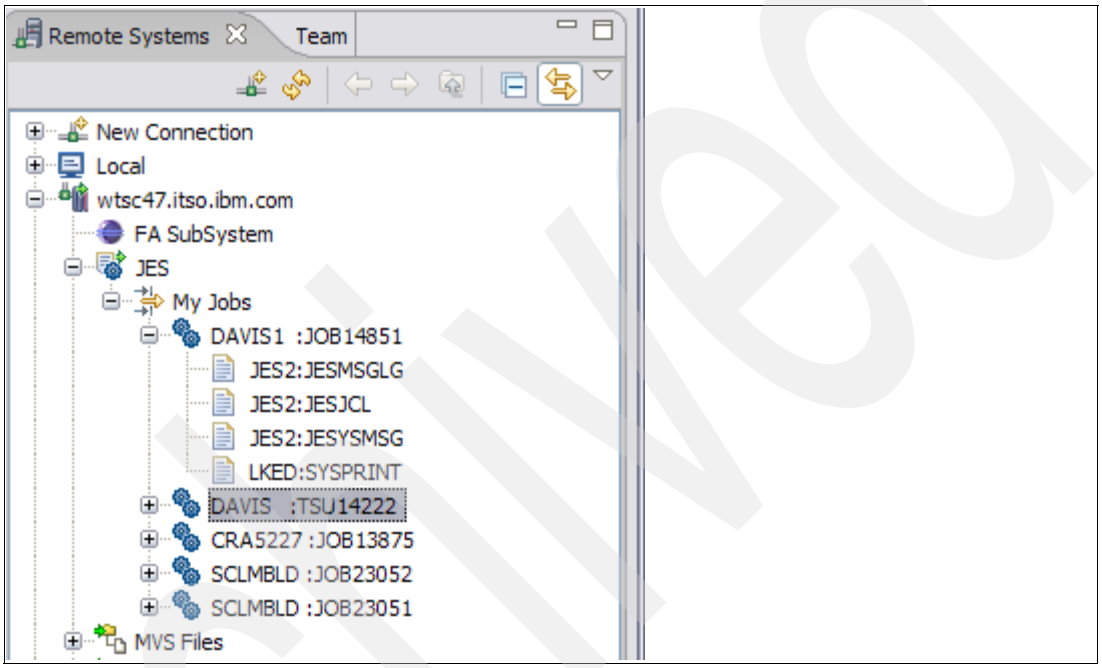


Figure 13-1 Job monitor

Archived

Figure 13-2 shows JES information based on the default filter. An advantage of WD/z is the ability to have multiple filters. The filters limit the jobs shown in Figure 13-2. In SDSF, a developer could do the same thing, but must enter the constraints on the command line. In WD/z, the filters can be defined once and simply opened when needed.

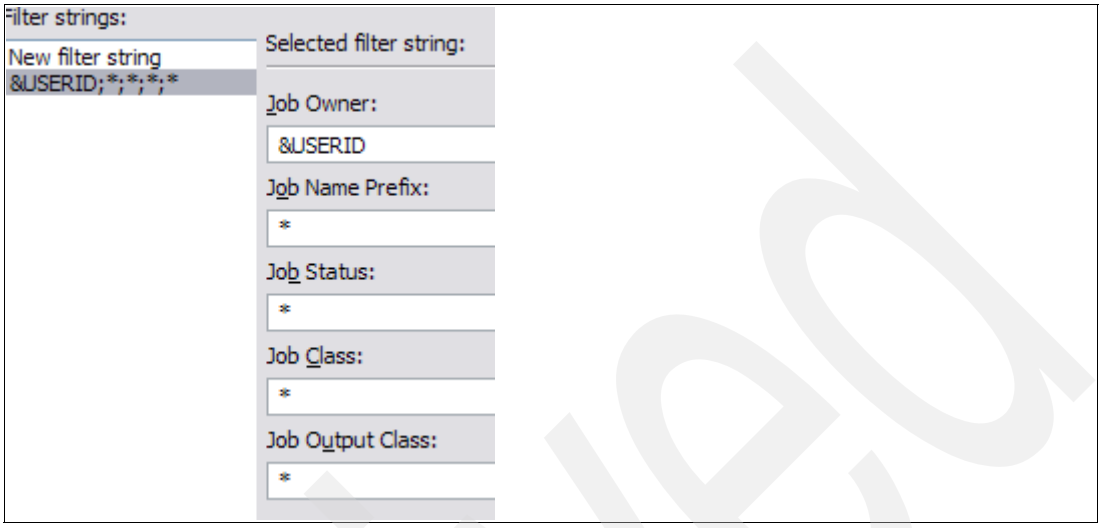


Figure 13-2 Filter

After expanding the filter, a developer can simply double-click the job, which performs the same function as entering S in SDSF, or the developer expands the job and then double-clicks the desired section. In both cases, the requested output comes up in the editor view and browse mode, as shown in Figure 13-3.

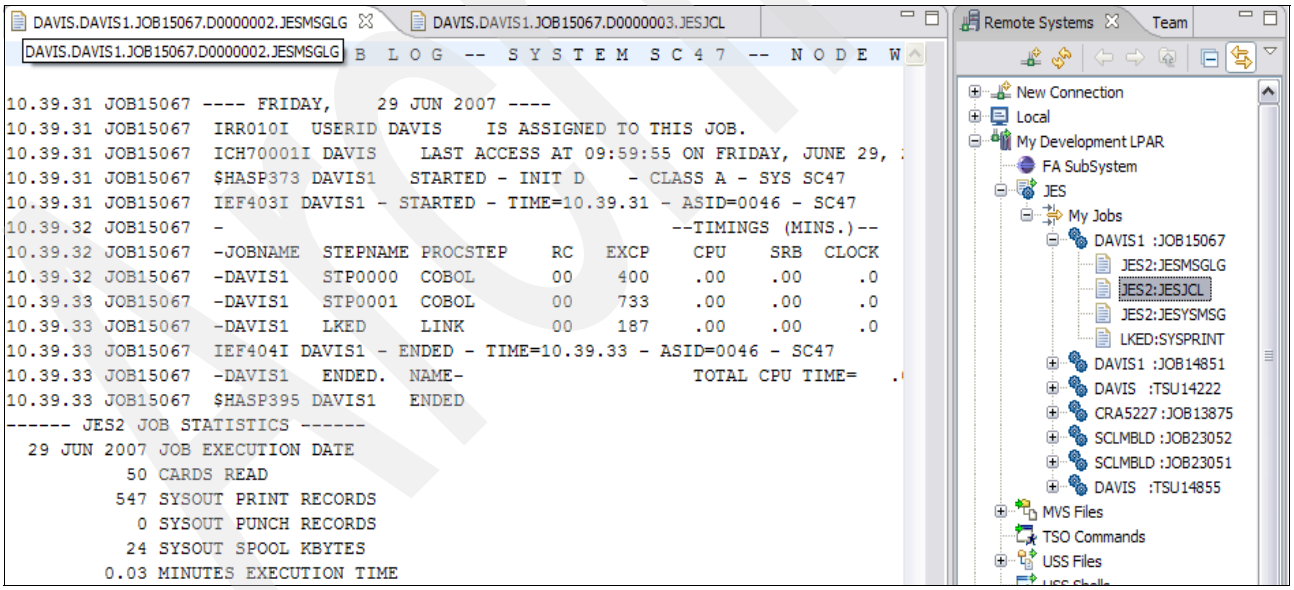


Figure 13-3 Editor view

13.4 TSO Command Server

The TSO Command Server, FEKFRSRV, is the technology that interfaces with TSO. It allows developers to enter TSO commands and run REXX procedures and CLISTs. It is also a scheduler for tasks running under the user ID of the developer. It is a task that is initiated by establishing a WD/z connection to the System z server, so there is one started task per active WD/z user. It will run until the session is disconnected or because of an inactivity timeout, as shown in Figure 13-4.

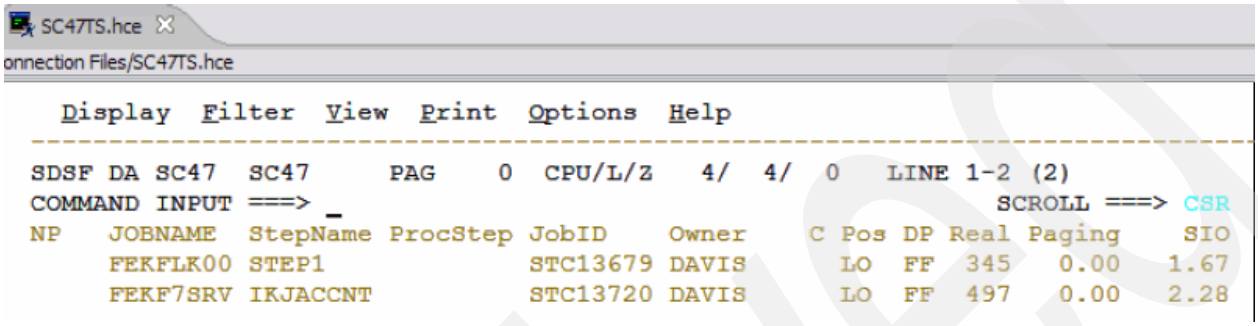


Figure 13-4 TSO Command Server

If a timeout does occur, any activity from the WD/z side to the host will automatically start another Command Server session. From a developer standpoint, the TSO Command server is transparent. A user would only notice it by looking in his JES spool under active sessions. During a session with a scheduler, you will not see data within the jobs. Only after a connection has been closed is the scheduler moved into SYSOUT. This is not the primary location of tracing information, but certain technologies write to it, as shown in Figure 13-5.

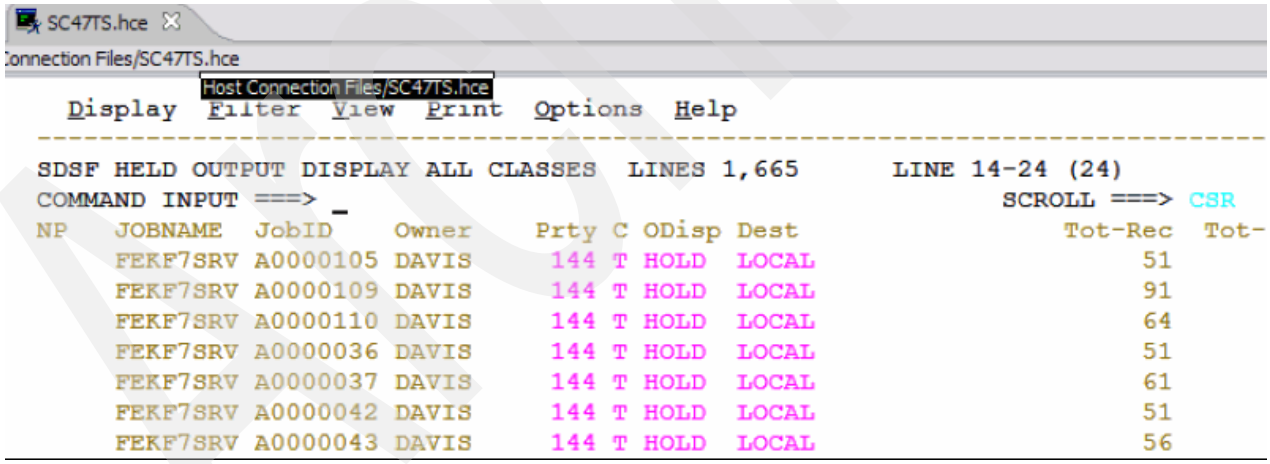


Figure 13-5 TSO Command Server

A good source of troubleshooting information for setting up the command server is found in *APPC and WebSphere Developer for System z*, SC23-5885.

13.5 CARMA and SCLM Developer Toolkit

The last server components delivered in WD/z are Source Control Management (SCM) based. The SCLM Developer's Toolkit is a host server component that is used in conjunction with the SCLM perspective in WD/z. The SCLM perspective is the interface to the contents of a defined SCLM installation. A connection to an SCLM system is shown in Figure 13-6.

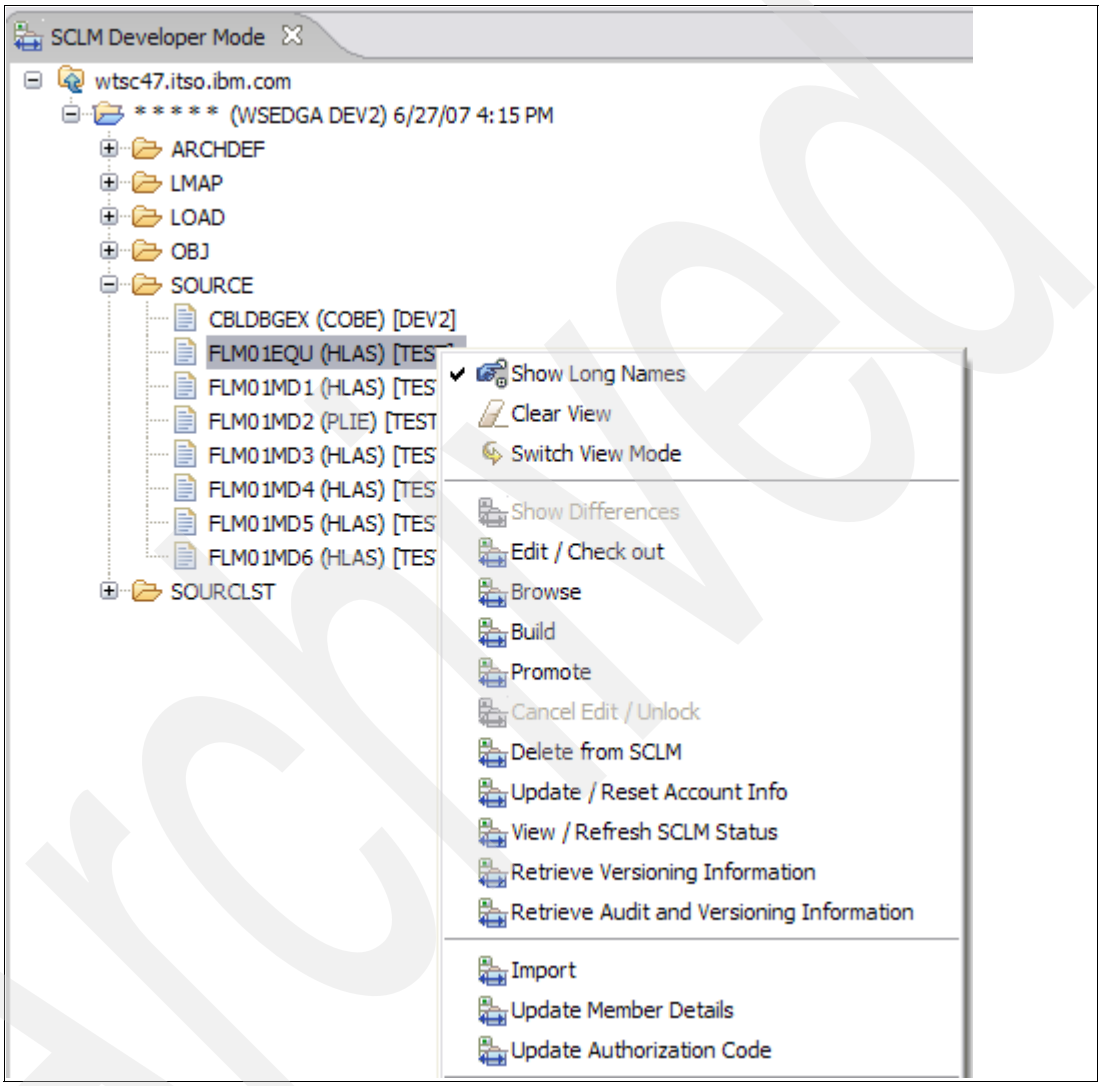


Figure 13-6 Connection to SCLM system

Through Common Access Repository Manager (CARMA), *any* SCM can be accessed. It allows the WD/z interface to interact with the SCM and allows application development to take place within the current controlled environment. Limitations or controls experienced today by developers would still be in effect in WD/z. Figure 13-7 on page 469 shows a connection to an SCM system.

Archived

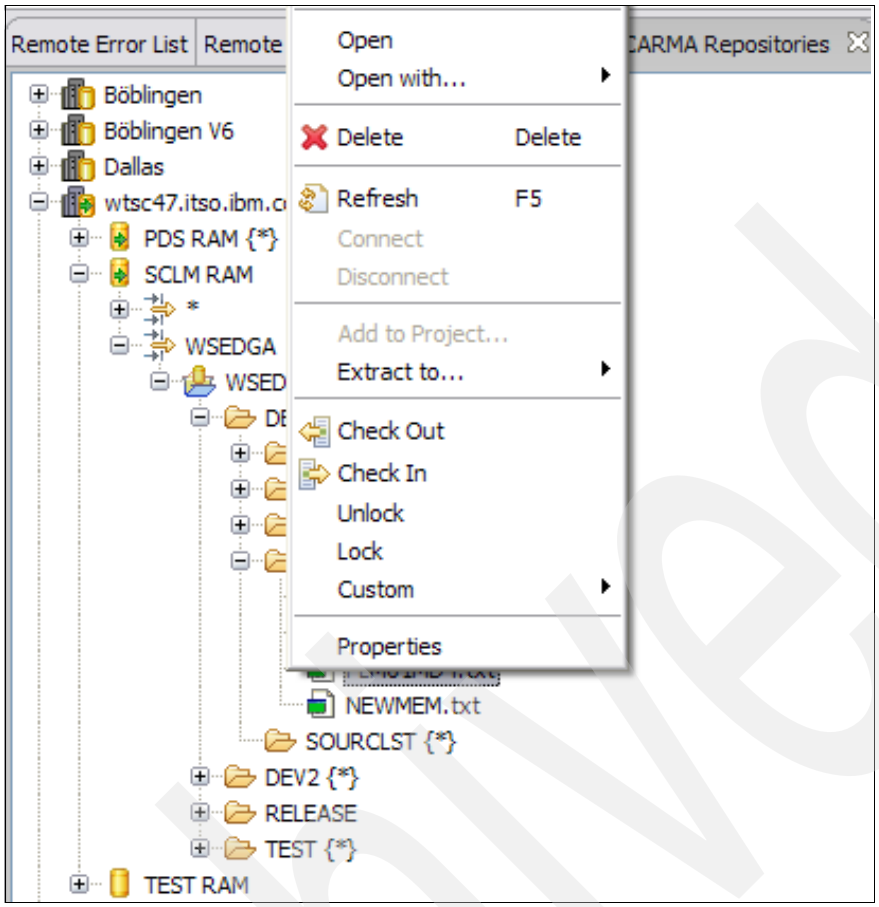


Figure 13-7 Connection to SCM

Archived

13.6 Initial view

Here we discuss the initial view.

13.6.1 Workspace

When WD/z has been started, a developer is first asked to define the workstation directory location where the *workspace* will be located. A workspace is a location where the plug-ins used by WD/z for that instance are stored. It is the location where any local files are stored and where the cache for the file server is stored. Physically, it is a directory structure where all resources for the workspace are stored, managed, and used by that workspace instance. Conceptually, a workspace is a container that allows multiple workspaces to coexist on a workstation, as shown in Figure 13-8.

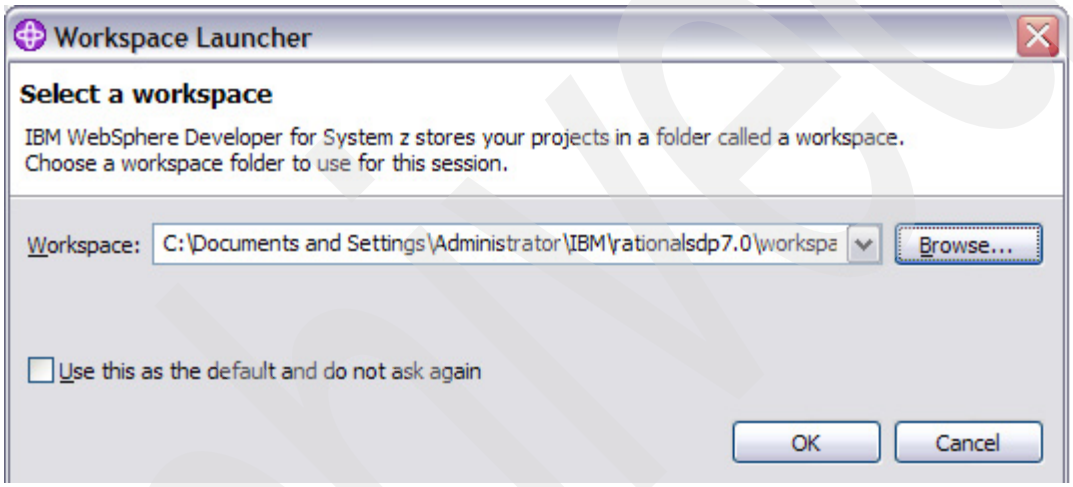


Figure 13-8 Workspace

13.6.2 Version difference coexistence

A major benefit of WD/z is that a developer can have multiple versions of WD/z installed on his workstation that can interact with separate workspaces. There can be workspaces that use Version 6 and workspaces that use Version 7. There can be, for example, WD/z Version 7 installed on a workstation, but a self-written, untested, and newer plug-in version can be used in a separate workspace and the standard plug-in in the others. This capability has even been extended in Version 7 to allow version differences between the host and the client so that a Version 6 host will function with Version 7 clients. Likewise, Version 6 clients will work with a Version 7 host. The only limitation is that functionality offered in a newer version will naturally not be available to the older version.

13.6.3 Perspectives

After the workspace location has been chosen, WD/z's Welcome window appears, as shown in Figure 13-9 on page 471. Here a developer can enable or disable roles.

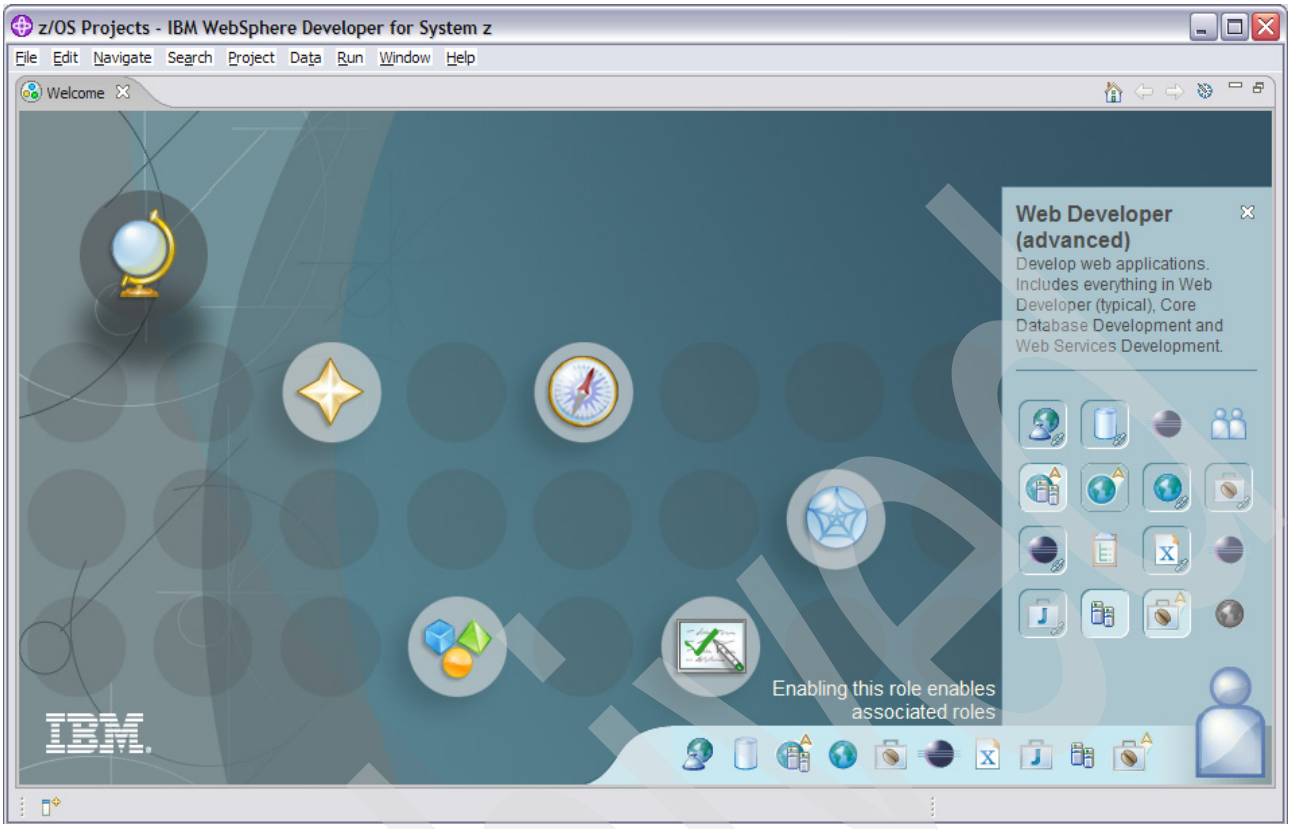


Figure 13-9 Welcome window

A role in Eclipse is a definition of what a person may develop. One role is, for example, an *Advanced Web Developer*. If enabled, Eclipse will activate the tools, views, and perspectives that are needed to perform this role. The opposite holds true for disenabling; if the Web Developer role is disabled, the Web perspective, for example, will not be available.

Archive

The ability to enable or disable perspectives in not limited to roles; individual perspectives as well as views can be enabled or disabled in the workspaces properties. Select **Window** → **Properties**, expand the **General** grouping, and select **Capabilities**, as shown in Figure 13-10.

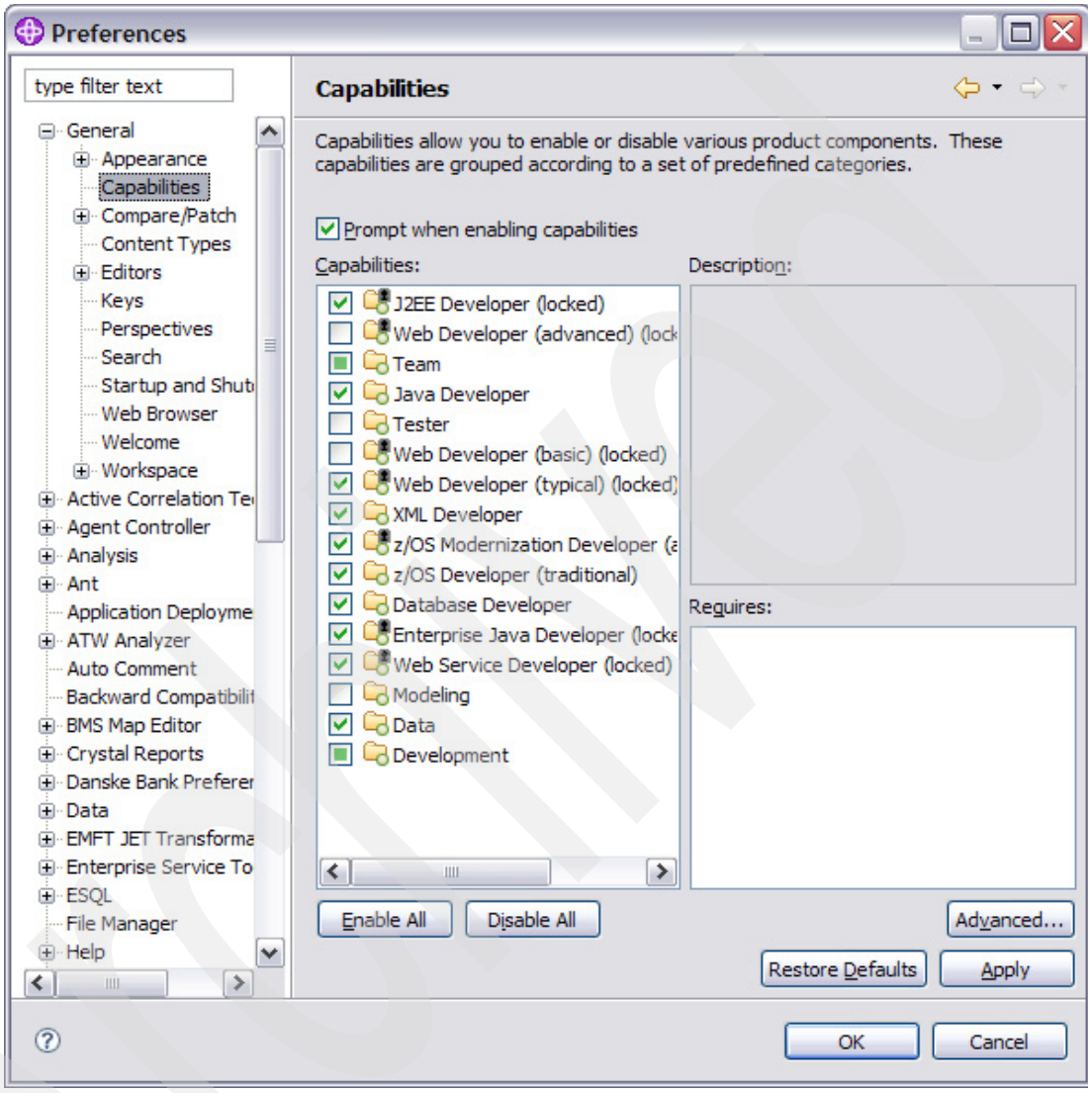


Figure 13-10 Preferences

If you close the Welcome window, the default configuration of WD/z appears. It is made up of views from two perspectives: z/OS Projects and Remote Systems, as shown in Figure 13-11 on page 473. The views Properties and Tasks, in the lower left corner, are basic Eclipse views.

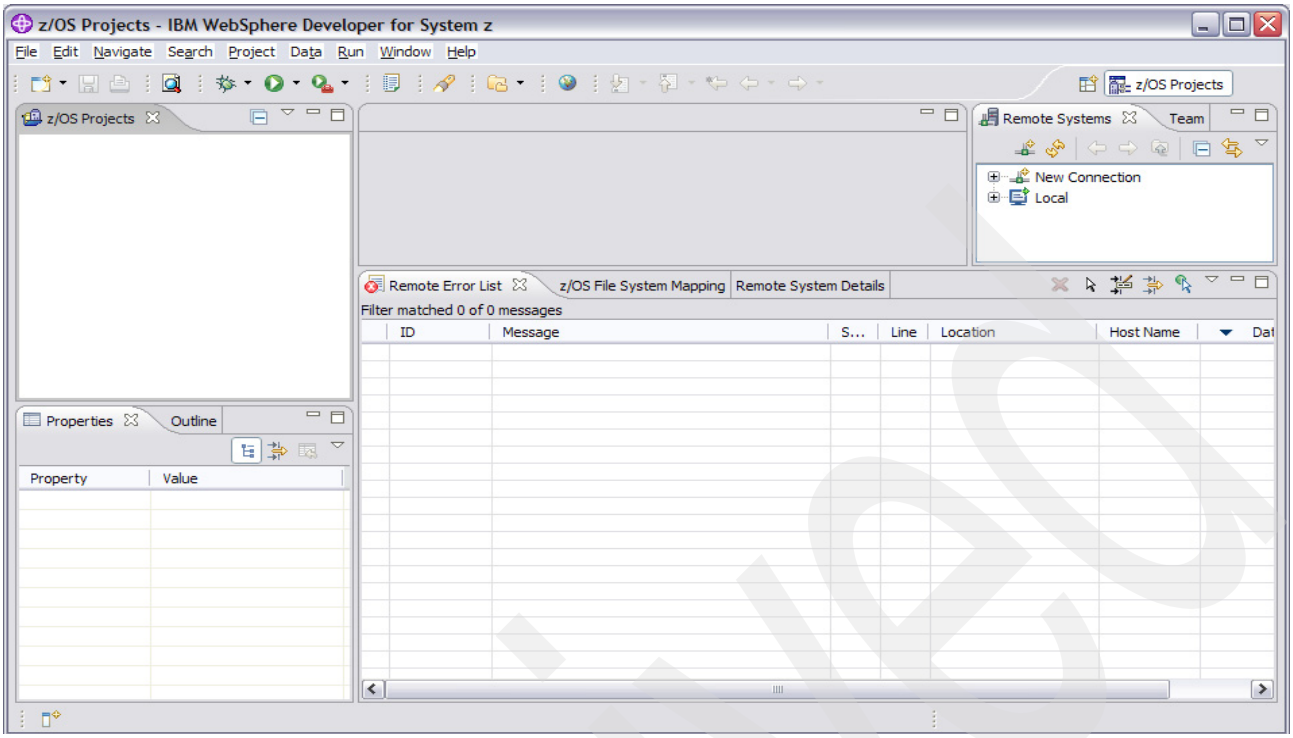


Figure 13-11 Default perspectives

You may choose the views that appear by selecting **Window** → **Show View**. Expand the perspective and choose a view. Its location within the workspace can also easily be changed by dragging and dropping it to a different location, as shown in Figure 13-12.

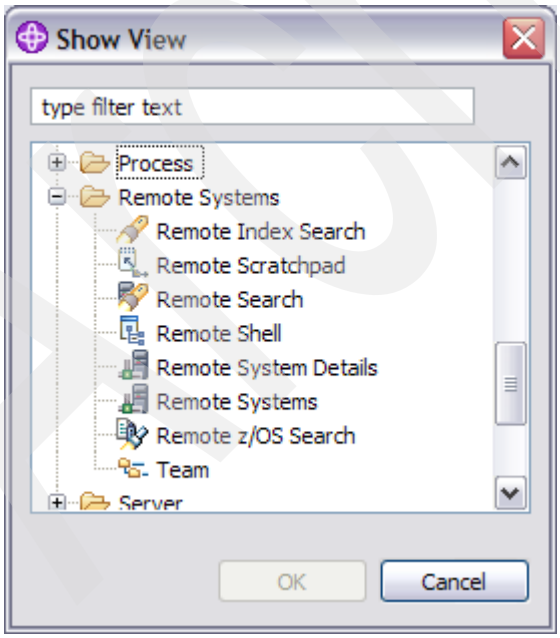


Figure 13-12 Show View

13.7 Connecting to a System z

It is a fairly straightforward process to connect to a host. First, a system must be defined within WD/z and then a connection can be established.

13.7.1 Differences of Version 6

In earlier releases up to Version 6, the properties of a defined system were exportable and importable. This means a connection could have been defined and then exported as an XML file. This file could then be imported by others to define their system. In Version 7, this capability was withdrawn for connection definition. It is still available at other levels, like MVS Files or Project, but not at the system level.

13.7.2 Defining a system connection

In the Remote Systems Explorer View, select **z/OS** → **New Connection**, as shown in Figure 13-13.

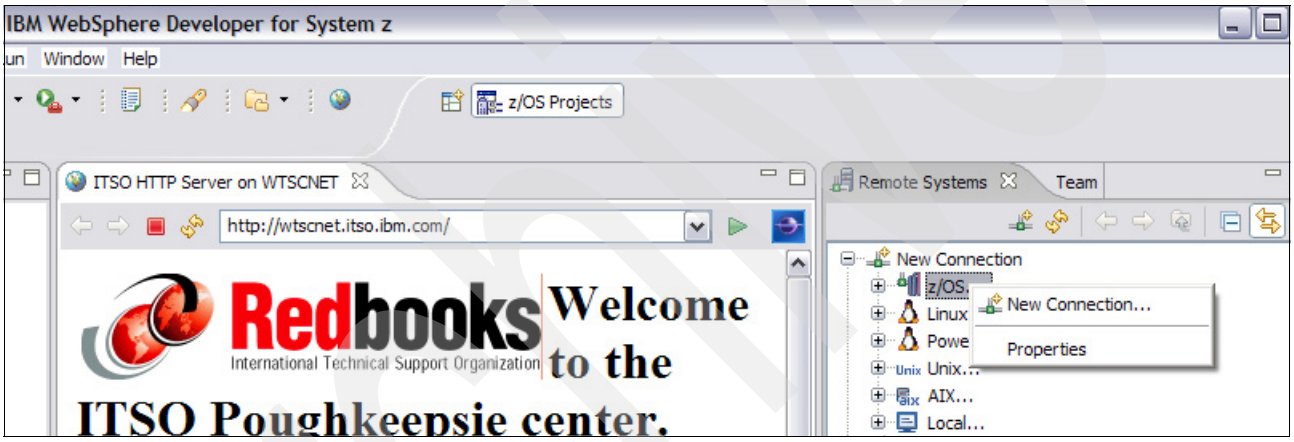


Figure 13-13 New Connection

Archived

This brings up the connection wizard. A developer would receive most of the necessary information for this wizard from his system programmer or WD/z coordinator. The profile name is the computer name registered to the Windows operating system. Although connections can be shared, in practice each developer normally defines the connection only for themselves. Click **Next**, as shown in Figure 13-14.

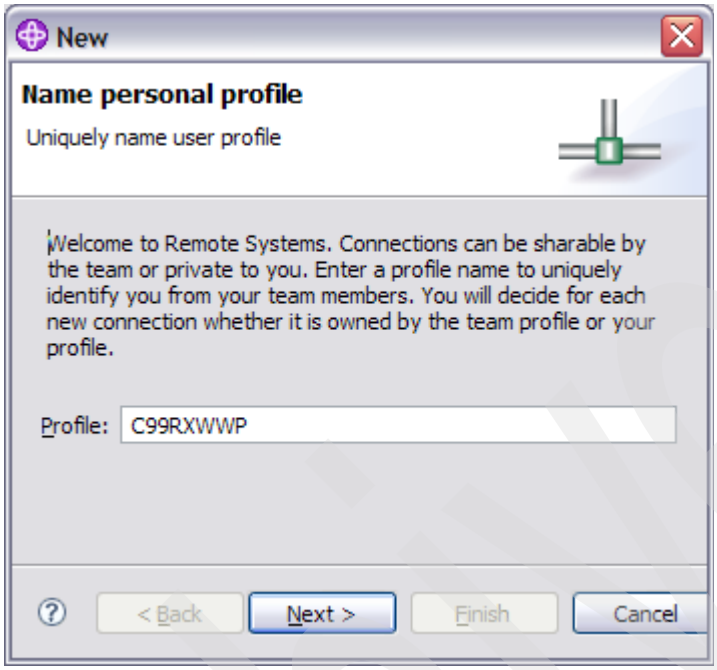


Figure 13-14 Profile

Archived

This is where the TCP/IP information mentioned earlier needs to be defined. Host name is the TCP/IP DNS name or address of the system. Connection name is what will appear in the Remote System Explorer view and represents this connection, as shown in Figure 13-15.

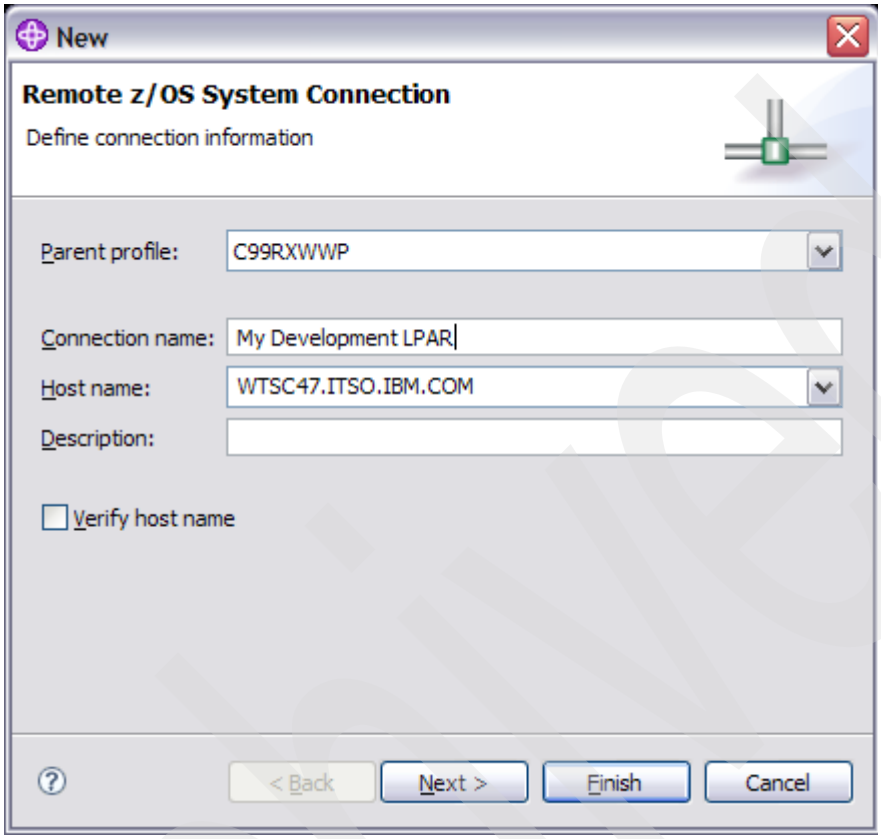


Figure 13-15 Remote connection

Archive

JES Job Monitor Port is the port defined for the job monitor in the job monitor's configuration file. Because a job's output can sometimes be very large, a limit can be defined. The limit is used when a developer wishes to browse the job output. If the number of records in the job output is larger than that defined in Figure 13-16, WD/z forces a developer to confirm that he wants to open the file.

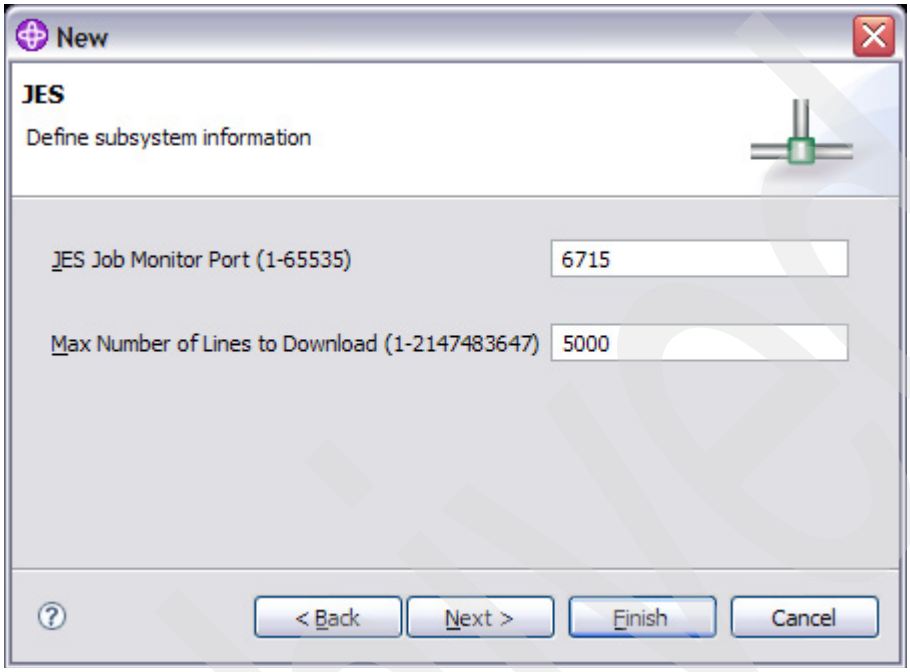


Figure 13-16 JES

Archive

Here the RSE information mentioned earlier needs to be defined. The RSE port number that the RSE daemon is managing needs to be entered. If the data transport needs to be secured, WD/z provides support with SSH and SSL in with Version 7. This should be defined. This security is applicable for all information except that which concerns the Host Connection Emulator Support, which we discuss later in this chapter. Certificate and alias information is provided by system programmers, as shown in Figure 13-17.

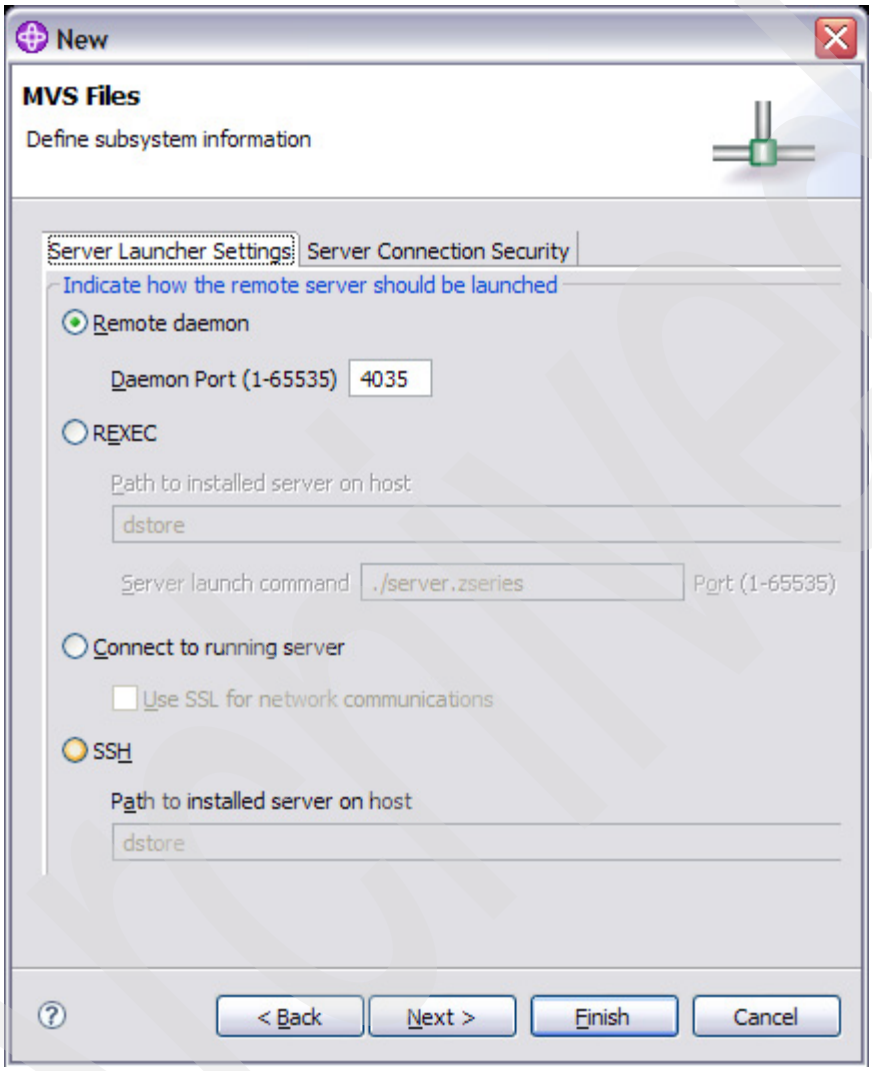


Figure 13-17 MVS Files

This information, however, must not immediately be defined. SSL as well as almost all property information can be found and managed in the workspace properties. Click **Finish** to conclude the connection definition phase, as shown in Figure 13-18.

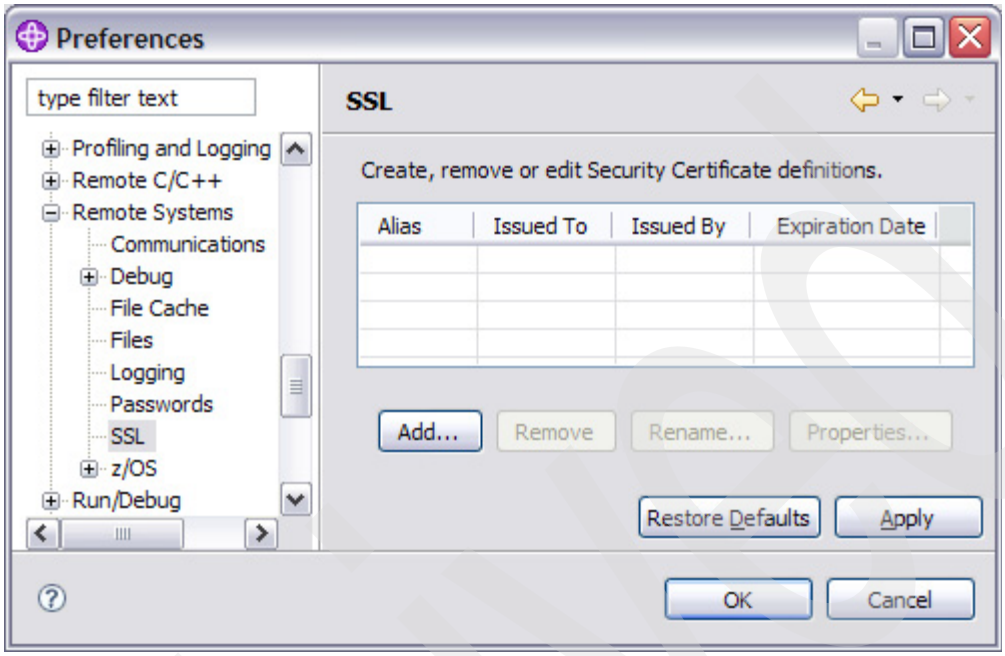


Figure 13-18 Preferences

Archive

13.8 Connecting to the defined system

Now that a system has been defined, a connection can be established. WD/z supports the ability to connect to JES and the MVS file system separately or both together. A developer can expand **My Development LPAR** and select **JES** or **MVS Files** and **Connect** or select **My Development LPAR** and **Connect**, as shown in Figure 13-19.

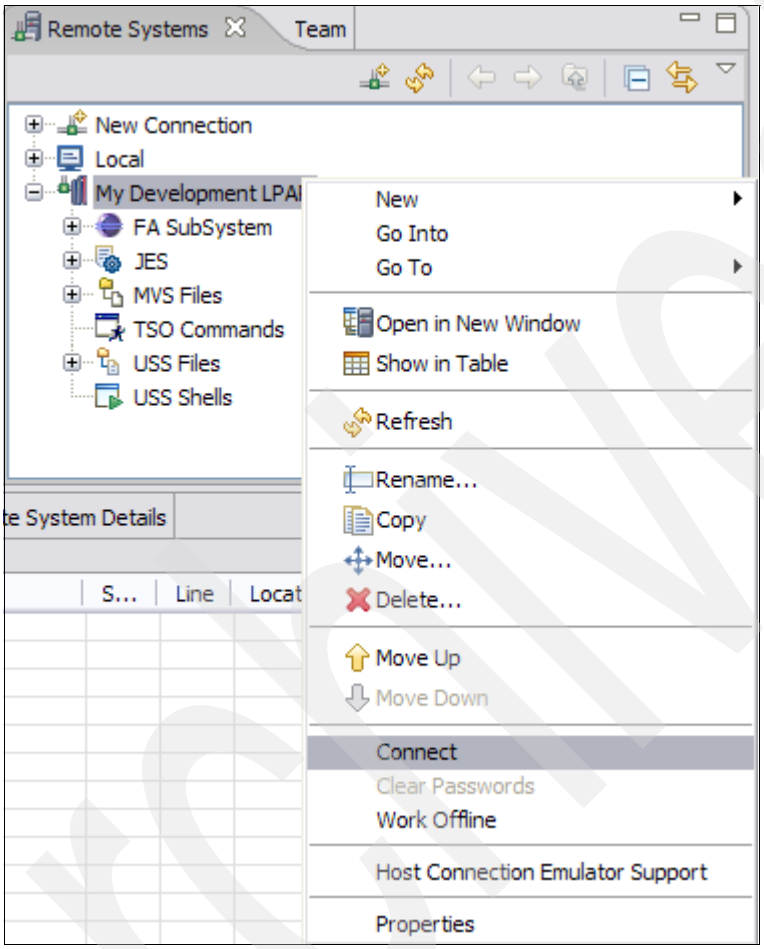


Figure 13-19 Connect

Archived

RSE security consists of RACF using the TSO user ID and password. A connection to the host from WD/z does not, however, block the user ID from using a 3270 emulation session logon. It does, however, produce a TSU job while connected. This can be seen in the bitmap earlier in the JES explanation, as shown in Figure 13-20.



Figure 13-20 ID and password

Archived

After the sign-on process has run to a successful conclusion, the resources available in the TSO systems are displayed in the Remote System Explorer view. MVS Files and UNIX System Services Files are handled the same way JES resources are. A filter is used to quantify and qualify what should be shown. A default filter is provided in WD/z. In MVS, it is the user ID. An * is used as a high level qualifier and is called My Data Sets. UNIX System Services files are initially, by default, UNIX based. My Home are those directories and files in the standard u/userid directory. Root is the complete UNIX System Services mounted in TSO, as shown in Figure 13-21.

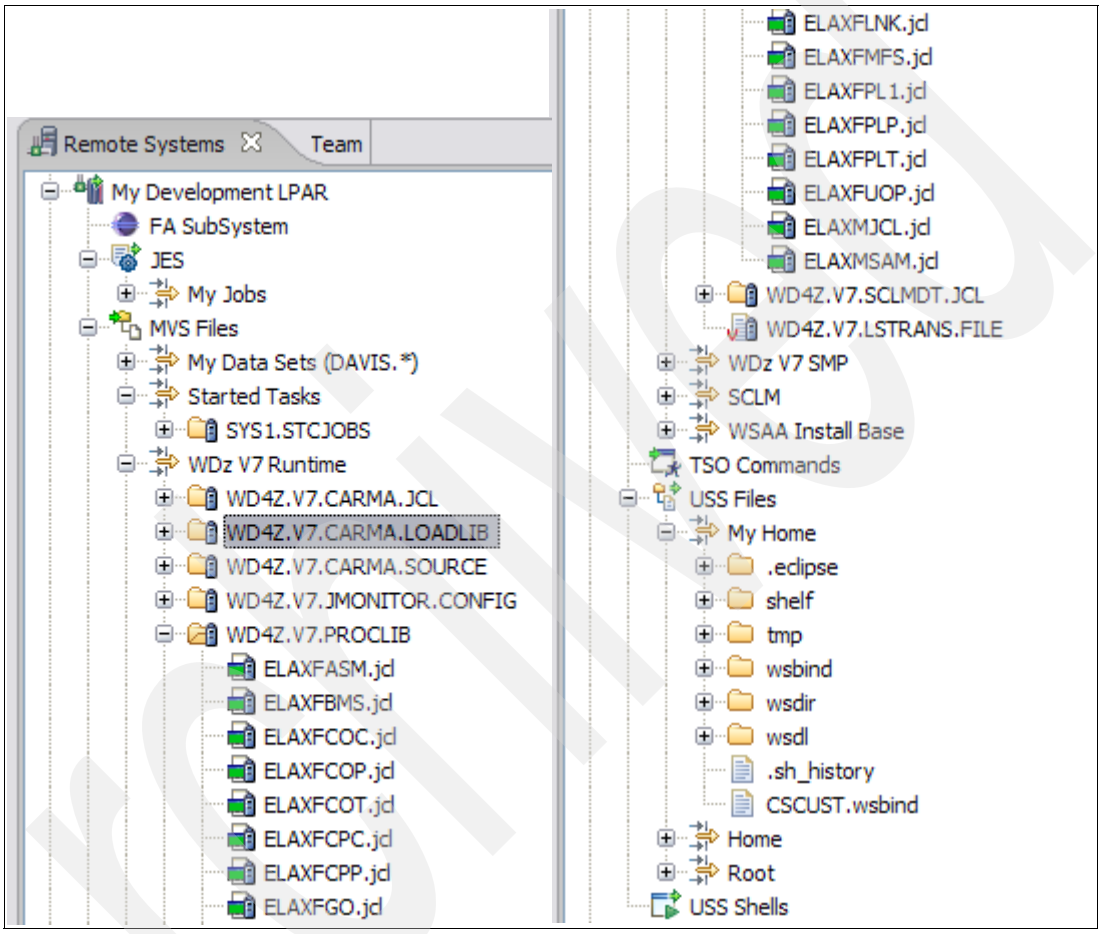


Figure 13-21 TSO resources

A filter is defined in both MVS and UNIX System Services by selecting the file tag and then selecting **New** → **Filter**.

13.8.1 Mapping

WD/z, like most workstation tools, uses a file mapping mechanism to determine in advance what tooling or capability is applicable to a specific file. In ISPF, a developer could be editing a COBOL file, jump to the command line, and enter the **sub** command. The file would be sent to the JES reader, which would immediately notice that there is no job card information and abend the job. WD/z eliminates these problems by allowing only files with a file extension that is identified by the JCL as having submit capability. The mapping in TSO is based on naming conventions, PDS names as well as file names, as shown in Figure 13-22 on page 483.

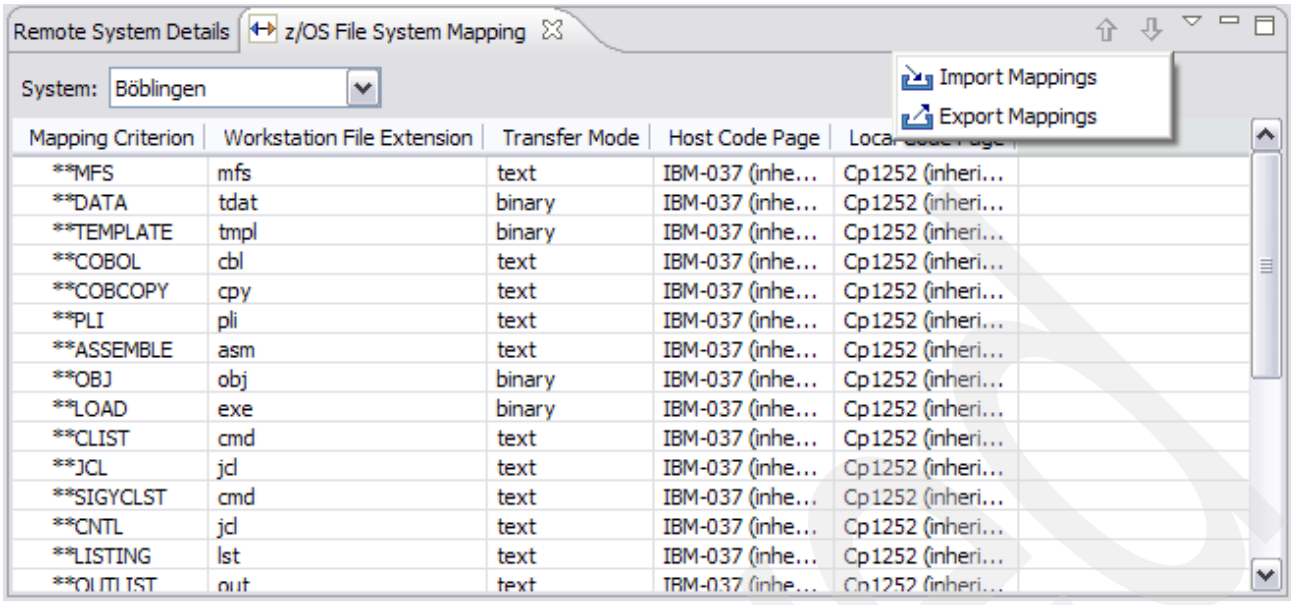


Figure 13-22 File System Mapping

The default mappings provided define any PDS name where the last HLQ ends in JCL to have the properties of a JCL. As seen in Figure 13-23, the PDS DAVIS.BATCH.JCL fulfills the mapping definition, so the Submit menu is available.

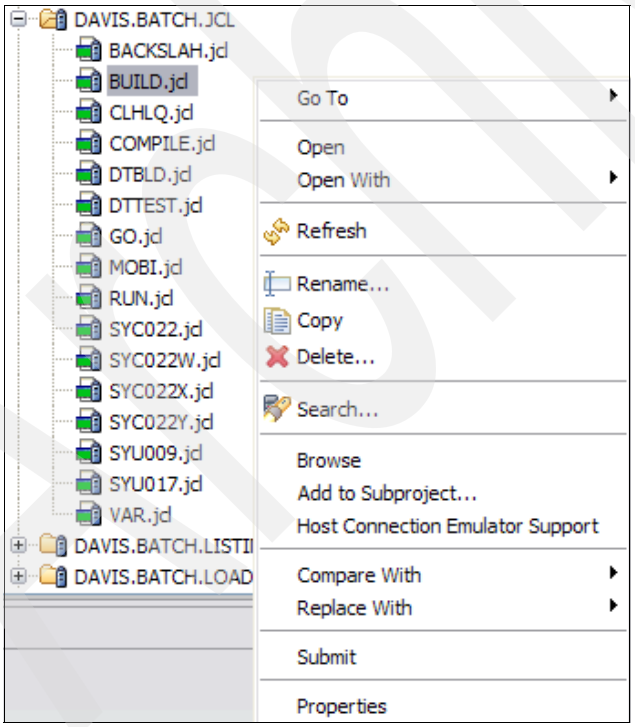


Figure 13-23 Default mappings

Archive

Mapping can be done not only at a PDS name level, but also at a member name level. That is why there is the ability to prioritize the mapping between PDS and member name mapping. This allows support in WD/z when one PDS has mixed languages or source files and JCL files together in one PDS, as shown in Figure 13-24.

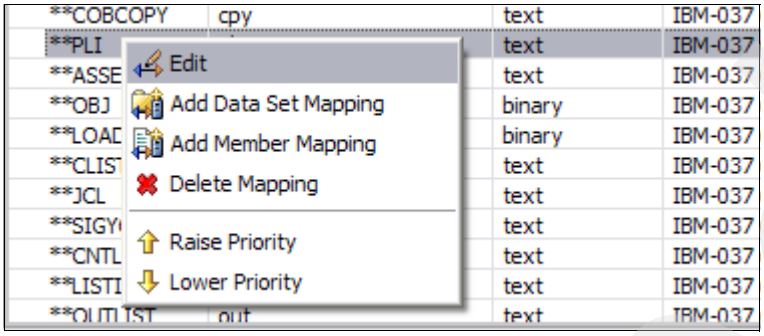


Figure 13-24 Mapping PDS

13.9 Alternative views

All resources, files, and JESs can be seen in additional separate views. This is accomplished by selecting **Open in New Window**, which brings up a duplicate / visual copy of the workspace, or **Show in Table**, which brings up a separate view within the workspace. Show in Table changes the representation of the information from a tree structure to a table. The table format is also more like SDSF in that the table columns shown are prioritized and determined by the developer, as shown in Figure 13-25.

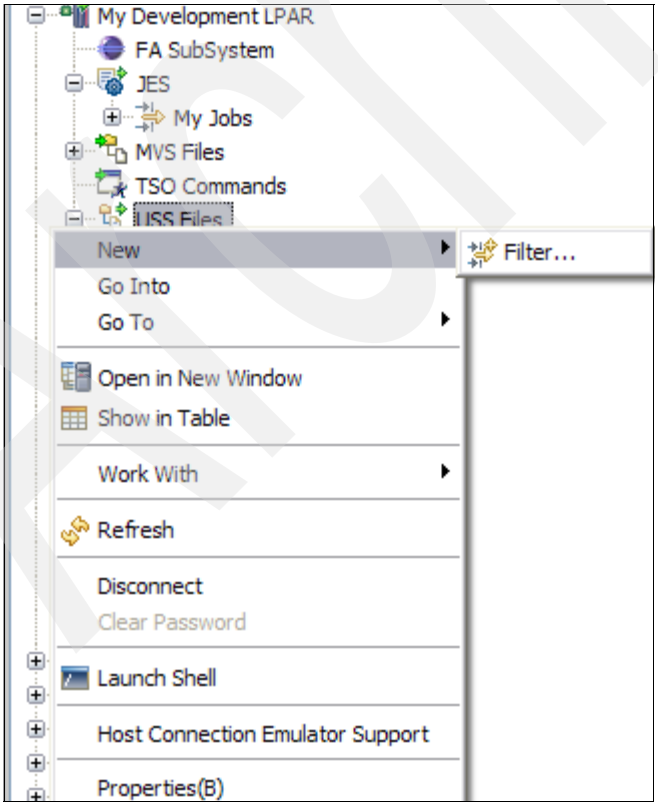


Figure 13-25 Alternative view

Because functionality is built into the view, developers often miss the place where the table's contents, for example, are determined. Each of the icons in the upper right of the table represents functions / tools available within the view, as shown in Figure 13-26.

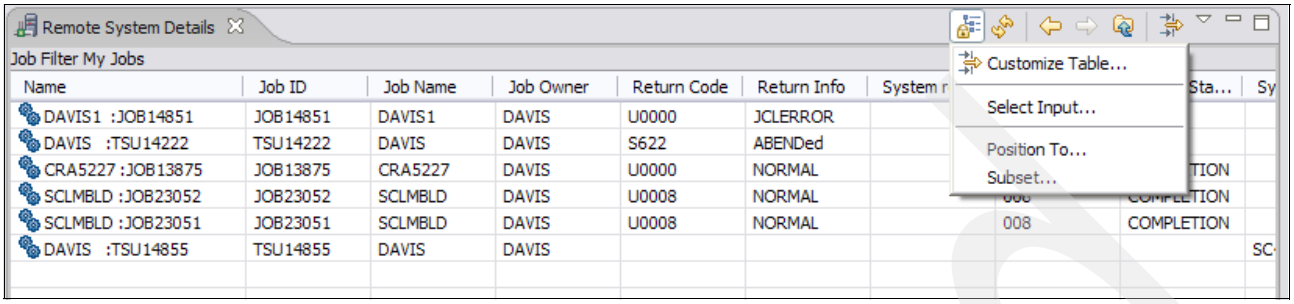


Figure 13-26 View

Each RSE connection has access to a command shell within each of the file systems. In the Remote System Explorer view, select **Launch Shell**. A view will open that shows a command-line interface to the UNIX System Services system. In TSO (select **Launch TSO**), commands such as **`netstat`**, REXX procedures, or UNIX System Services commands such as **`ls`**, or Java scripts, are executed. The shell is a command-line interface, not a conversational tool. The results from the commands or input are fed back to the shell, but it is not interactive.

Archive

Not all tools or functions in ISPF are available or appropriate in WD/z. To keep from having to change interfaces (tools), a telnet session is available as a view. It is a 3270 emulator session and appears in the Editor view. The IP information is taken from the My Development LPAR definition, so it can be consistently called from Host Connection Emulator Support. The keyboard and key combinations can be customized either in the view or in the workspace properties, as shown in Figure 13-27.

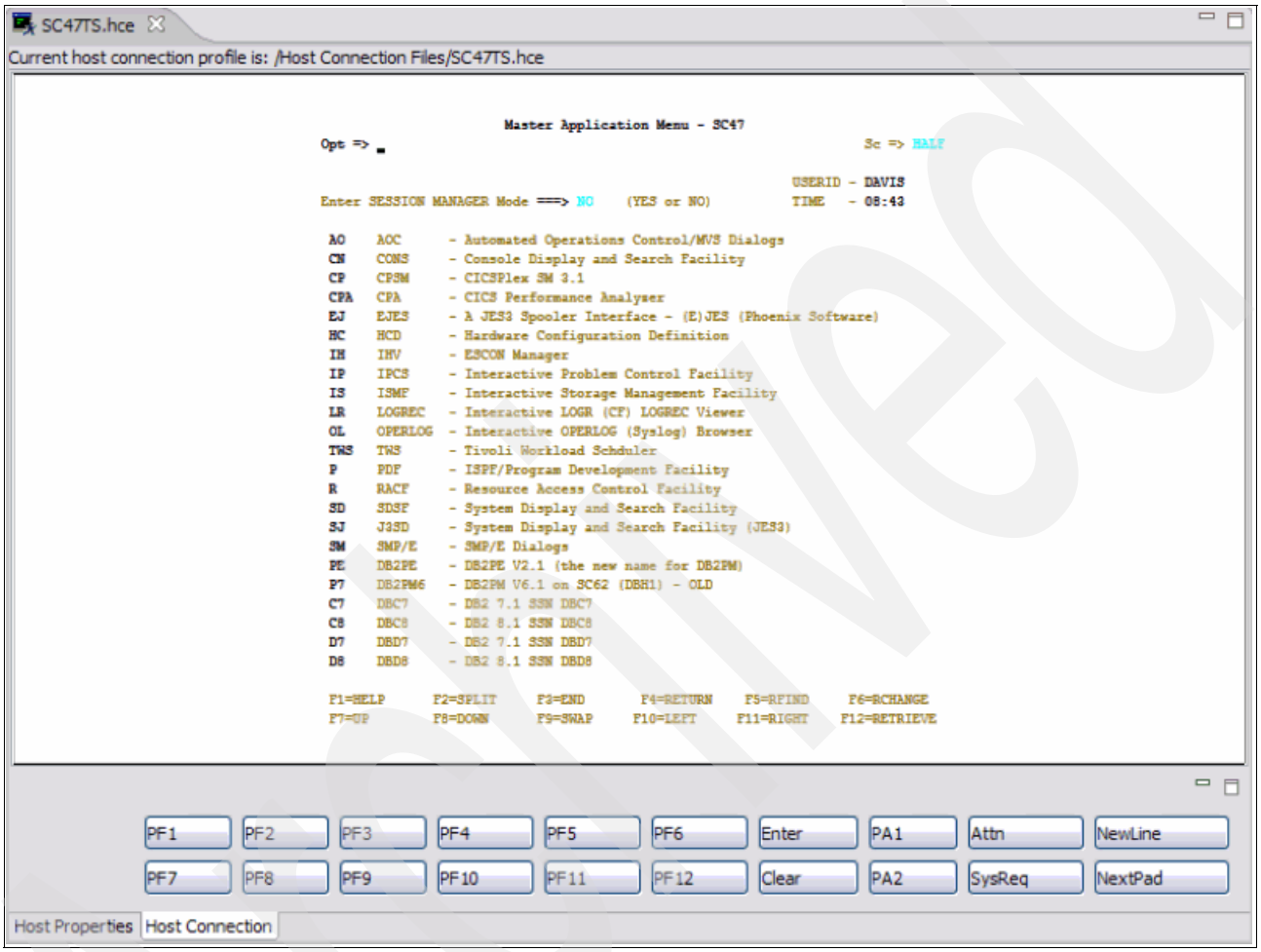


Figure 13-27 Application menu

The connection sessions are scriptable, so developers can define multiple sessions, script them to reach the system they want, and store the file identified for the session in a Simple Project created for that purpose, as shown in Figure 13-28 on page 487. This allows, for example, a session for CICS or IMS, one for production, and one for development.

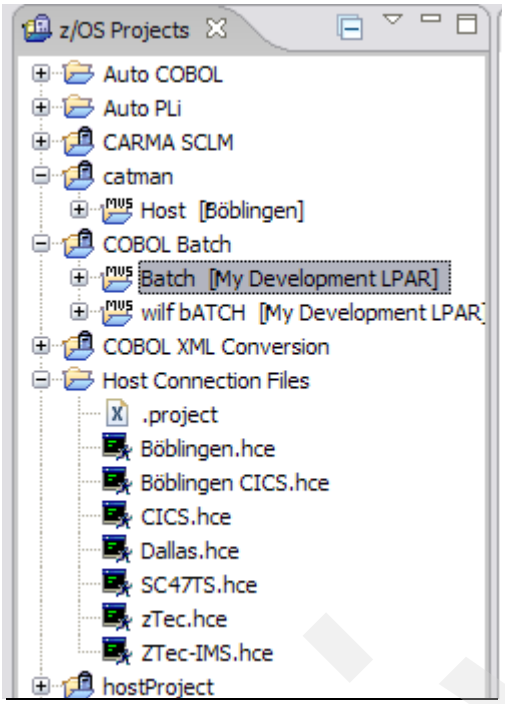


Figure 13-28 z/OS Projects

13.10 Developing single resources or projects

Depending upon the developer's goal, development can take place either in the Remote System Explorer view or within the z/OS Projects view. This is due to certain capabilities only being offered in the z/OS Projects view. The major difference is the RSE view's inability to process load modules.

Within the RSE view, a developer can:

- Edit
- Submit jobs
- Perform remote syntax check (added in WD/z V 7)
- Generate a JCL to:
  - Compile / assemble
  - Link
  - Go

There is, however, no Build capability in the RSE view, since it a compile and link function. The debugger also cannot be started based on a load module, although it is displayed and mapped within a PDS. Why is this?

The concept from Eclipse is that development takes place in a *project,* a project being a container that, at the end of the development cycle, results in one binary. In the case of Eclipse, this would be a JAR file.

Archived

WD/z, built on top of Eclipse, has extended this model in Version 7. WD/z V7 offers the capability of multiple load modules being produced in one project. This is attained through the implementation of a super-project.

Archived

14

Connecting to CICS

Integrating earlier systems, made up of COBOL/CICS-based applications, with systems running under Web application servers using Java, is an extremely common requirement. We discuss this topic in this chapter.

Note: For more information, refer to the SG247482-Chapter 13.zip file. Instructions for downloading this file can be found in Appendix B, “Additional material” on page 645.

Archived

© Copyright IBM Corp. 2009. All rights reserved.

14.1 Introduction

This chapter will illustrate an example of how to code Java classes that connect to an existing COBOL/CICS program using the J2EE Connectors (J2C) tools available in Rational Application Developer for WebSphere Software (RAD) Version 7.0.0.1 and test it using IBM CICS Transaction Gateway for Multiplatforms (CTG) and IBM TXSeries for Multiplatforms that is delivered with WebSphere Developer for System z (WD/z) Version 7.0.0.1.

The objective is to create and test Java code accessing the CICS (TXSeries) that runs in the workstation, without having to deploy to the mainframe, since WD/z delivers the necessary infrastructure for that. You could create those connectors with RAD only, but you would not be able to test it. The advantage of using WD/z is the ability to test the code in the workstation. Since WD/z is installed on top of RAD, all capabilities of RAD are available in WD/z.

Also in this chapter, we show you how to create a simple JSP test client that invokes the Java bean created and also how to create a Web Service that will invoke this Java bean.

14.1.1 Connecting to existing COBOL/CICS programs

It is possible to connect to COBOL/CICS using many different technologies. In this example, we use the J2EE Connector Architecture (JCA), which is an open architecture and fully compliant with J2EE specifications.

The J2EE Connectors (J2C) tools, resource adapters, and file importers allow you to create J2EE Connector artifacts, which you can use to create enterprise applications. Also, those tools enable you to create J2EE applications running on WebSphere Application Server to access operations and data on Enterprise Information Systems (EIS), such as CICS ECI or IMS. It offers a number of advantages that can be provided by a J2EE application server. These advantages include:

- Security credential management
- Connection pooling
- Transaction management

These advantages are provided by means of system-level contracts between a resource adapter provided by the connector (CICS Transaction Gateway or IMS Connect, for example) and the application server. There is no need for any extra program code to be provided by the user. Thus, the programmer is free to concentrate on writing the business code and need not be concerned with providing quality of service. The J2EE Connector Architecture (JCA) defines a programming interface called the Common Client Interface (CCI). This interface can be used, with minor changes, to communicate with any EIS.

Figure 14-1 on page 491 illustrates the architecture of the J2EE Connector (J2C) tools within the development environment.

Archived

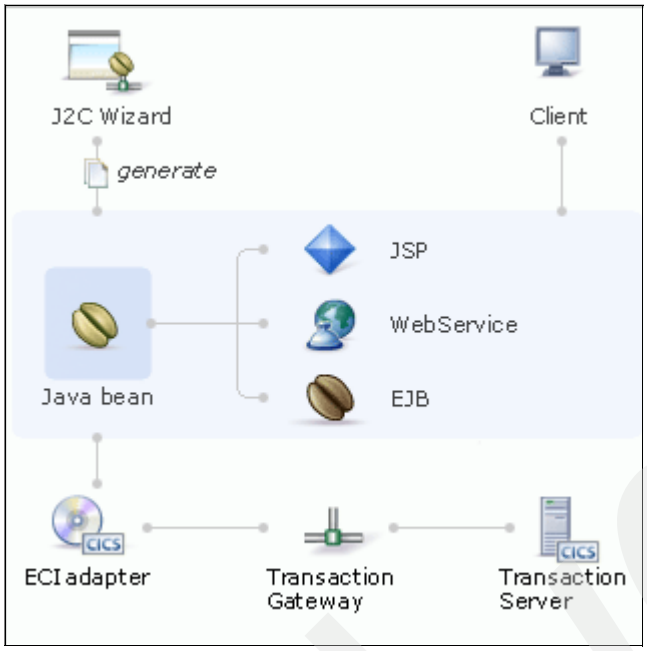


Figure 14-1 J2C wizards and J2EE Connector architecture

14.1.2 Resource adapters

A resource adapter is a system-level software driver that is used by a Java application to connect to an enterprise information system (EIS). The resource adapters reside on the application server and provide connectivity between the EIS, the application server, and the enterprise application. Applications deployed on the application server communicate with the resource adapter using the CCI. The resource adapters (RAR files) contains all the information necessary for installing, configuring, and running a JCA Resource Adapter; they can be provided by and used by any vendor, provided they comply with the J2EE Java Connector Architecture specification.

In order that our application can communicate with an Enterprise Information System, such as CICS or IMS, a resource adapter is required to create a communication link. J2C tools include a number of resource adapters that allow you to create and test J2C enterprise applications in its unit test environment. These RAR files can be imported into the workbench and used to create enterprise applications.

Two CICS resource adapters are shipped with RAD V7.0.0.1 (also WD/z):

- CICS ECI Adapter V5.1: This version of the CICS ECI resource adapter is based on Version 1.0 of the J2EE Connector Architecture (JCA V1.0). Because CICS ECI resource adapter for Java Version V5.1 is a JCA V1.0 resource adapter, it will only run in a JCA V1.0 application server or WebSphere Application Server Version 5.0.2 (or later).
- CICS ECI Adapter V6.0.2: This version of the CICS ECI resource adapter is based on Version 1.5 of the J2EE Connector Architecture (JCA V1.5). This version of CICS ECI resource adapter for Java runs with WebSphere Application Server Version 6.0 and later.

Archived

14.1.3 Importers

In order that our application can process source files from the CICS or IMS information systems, the data needs to be imported and mapped to Java data structures. Three importers are available in RAD Version 7: the C Importer, the COBOL Importer, and the PL/I importer.

These tools allow you to import COBOL, C, or PL/I programs into your application through a process of data type transformation. The importers map the data types contained within the source file so that your application can access the source material. In other words, if you are coding Java applications to access transaction programs written in COBOL, C, or PL/I in CICS or IMS through J2C resource adapters, the Java applications will need to:

- Serialize values from Java to the COBOL or C byte buffer that the IMS or CICS program expects.
- Deserialize the returned value from the COBOL or C buffer for processing in the Java application.

14.1.4 J2C wizard

The J2C wizard enables you to create J2C applications, either as a stand-alone program, or as added functionality to existing applications. The wizard dynamically imports your selected resource adapter, allows you to set the connection properties to connect to the EIS servers, guides you through the file importing and data mapping steps, and facilitates the creation of Java classes and methods to access the transformed source data.

Figure 14-2 illustrates the flow of the J2C Java bean wizard through the creation of a J2C Java bean, a data bean, and optional deployment artifact.

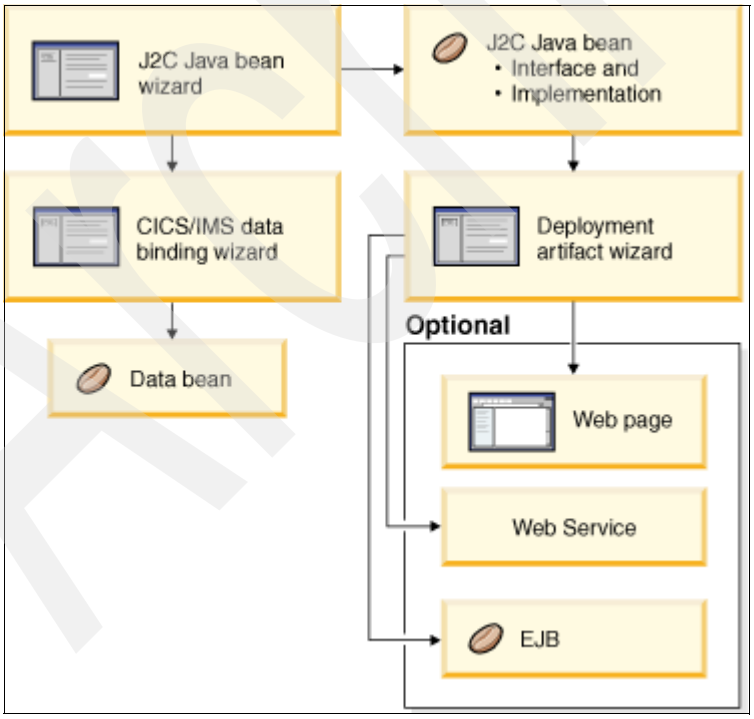


Figure 14-2 Creating J2C artifacts with RAD

The process of using the J2C wizard to build a Java application that runs an EIS transaction is summarized by the following steps:

1. The J2C wizard imports C, COBOL, or PL/I definitions of the EIS transaction input and output messages into the Java Data Binding wizard to map to Java data structures. This wizard creates Java data bindings for the input and output messages.
2. The J2C wizard provides Java data bindings to the J2C Java bean wizard. This wizard creates a J2C Java bean with methods that can be used to run EIS transactions on the host.
3. The J2C wizard creates a J2EE resource that you can associate with the J2C Java bean. This J2EE resource can be deployed to WebSphere Application Server and used to run your EIS transactions. The types of J2EE resources that can be created from a J2C Java bean are
   - JSP
   - Faces JSP
   - Web Service
   - EJB™
4. The wizard exports the J2EE resource, packaged as an EAR file, so that it can be deployed to and run on a stand-alone WebSphere Application Server.

14.1.5 Example application

For this example, you would only need a COBOL/CICS server program that would be invoked through EXEC CICS LINK, but in order to be sure that this program is working fine, you also deploy a COBOL/CICS client program that will invoke this server. Also, a TXSeries transaction named JCA1 will need to be created to start this COBOL client program.

The client program sends a map, asks for a data, and invokes the server program that accesses DB2 and returns the data to the client program, which displays the data back.

Figure 14-3 shows the programs that we use in this example. This code is located in the file Local_Development_1f.zip, which can be downloaded by following the instructions found in Appendix B, “Additional material” on page 645.

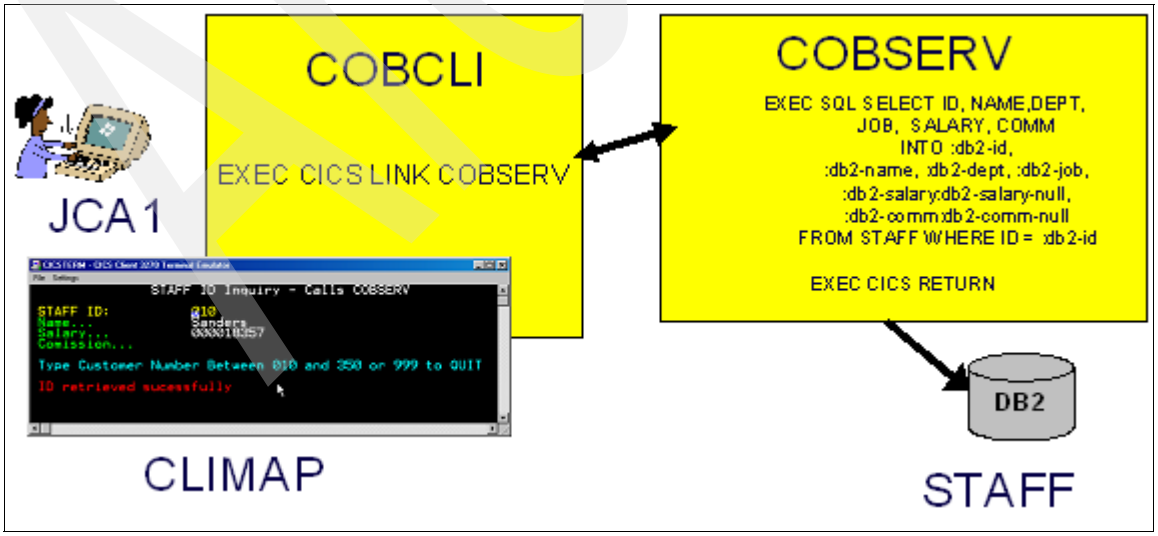


Figure 14-3 Map and COBOL/CICS programs provided

Before we start using the wizards to create the adapters, we show here how to prepare our workstation to be able to test the created connectors without having to go to the CICS on the mainframe.

14.2 Preparing the workstation for this example

Before we start this exercise, we need to prepare the workstation so that we will be able to test the code that we produce later on. The following steps are necessary for the preparation:

- Verify that DB2 has the required sample table.
- Install WebSphere Developer for System z Version 7.
- Install CICS Transaction Gateway (CTG).
- Install TXSeries Version 6.1 and create a Region.
- Configure TXSeries to support CTG.
- Configure CTG to communicate with TXSeries.
- Test if CTG is communicating with TXSeries.
- Load the sample COBOL into WD/z, compile/link, and create the executables.
- Install the compiled programs and required resources into TXSeries.
- Test the provided Client/Server COBOL programs.

14.2.1 Verify that DB2 has the required sample table

We need DB2 for this example, since the COBOL server programs uses it to retrieve data. IBM DB2Universal Database Enterprise Server Edition is delivered with WD/z and must be installed. Refer to the product documentation for the DB2 installation instructions and sample database creation. In this example, we have used DB2 Enterprise Server Edition Version 8.2; other versions could be used, and at least Version 7 is necessary. We will use the sample table named STAFF located in the SAMPLE database.

Archived

To verify that the STAFF table is created, we can use the DB2 Control Center and list the table contents, as shown in Figure 14-4.

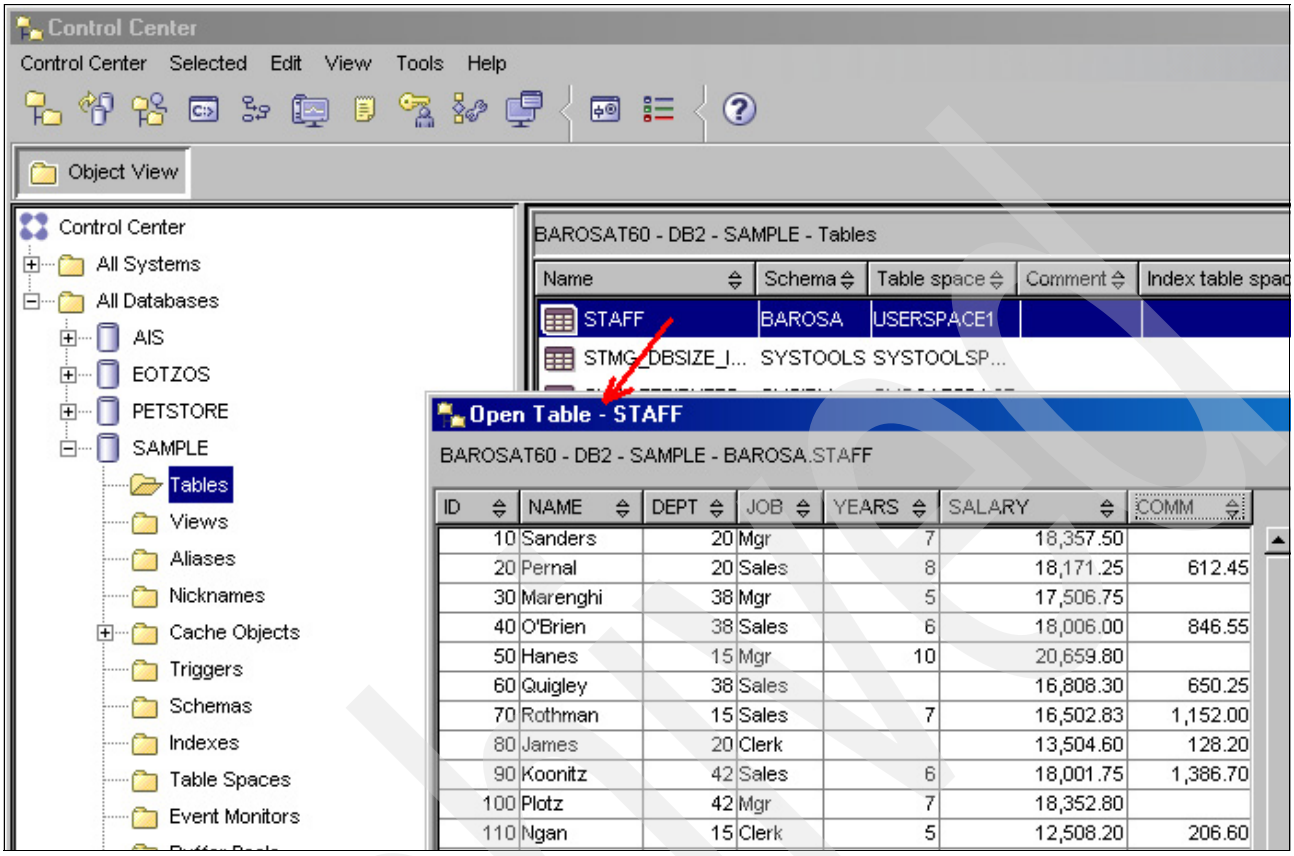


Figure 14-4 Simple query against table STAFF

14.2.2 Install WebSphere Developer for System z Version 7

Refer to the product documentation for the installation instructions. Remember that this will require RAD V7 to be installed first.

Archive

Be sure that you have applied the latest available services on both products, RAD and WD/z. You must use the Installation Manager and Update Packages dialogs and the services can be automatically download from the Web, as shown in Figure 14-5, which also shows the RAD and WD/z services level that we have used in this example.

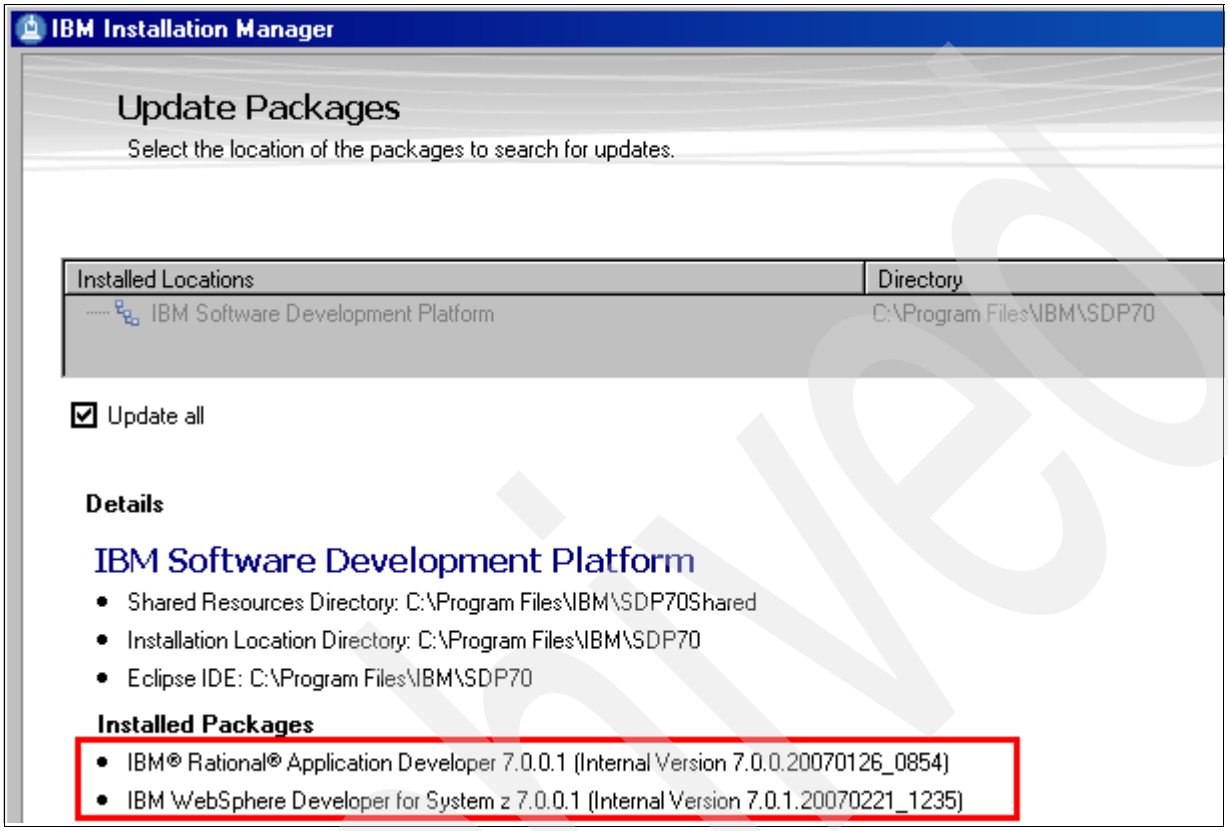


Figure 14-5 Update Packages window

14.2.3 Install CICS Transaction Gateway (CTG)

IBM CICS Transaction Gateway for Multiplatforms (CTG) provides secure, easy access from Web browsers and network computers to CICS applications, using standard Internet protocols in a range of configurations.

CTG is located outside the applications and it is used during application development and, after deployment, at run time. The gateway supports many programming paradigms and includes a J2EE Connector Architecture interface that allows you to develop applications to take advantage of the many facilities of compliant J2EE Servers. The CTG provides both a local (direct to CICS) protocol as well as networked protocols, such as TCP and SSL for communication to a CICS Transaction Gateway Server, which directs calls to CICS Servers (TXSeries). For more information about the various aspects of the CTG, such as setup, administration, and program development, refer to the documentation supplied with the product. In this example, we shall use the TCP protocol.

The CTG is delivered with WD/z Version 7 for testing purposes and can be found with the CDs package shipped with WD/z.

Archived

14.2.4 Install TXSeries Version 6.1 and create a region

IBM TXSeries for Multiplatforms is an advanced transaction processing solution that coordinates and integrates servers, managing high-performance applications and data sources across a network. It incorporates Customer Information Control System (CICS) and enables you to create a distributed client/server environment with all the reliability, scalability, and data integrity that are required for online transaction processing. It is your option to install it or not when installing WD/z. We will need it for this example.

The default directory where CICS will be installed is C:\opt\cics. We use this default in this example. We assume that TXSeries has already been configured and a TXSeries Region has been created. Note that the TXSeries region in this example will be called JCACICS.

14.2.5 Configure TXSeries to support CTG

After you have installed CTG and TXSeries and created a Region, your next step is to set up the communication links between those two products.

Using the TXSeries administrator console, you need to create the listener named DEMOTCP, as shown in Figure 14-6. Note that this name will match the server name defined in the CTG configuration.

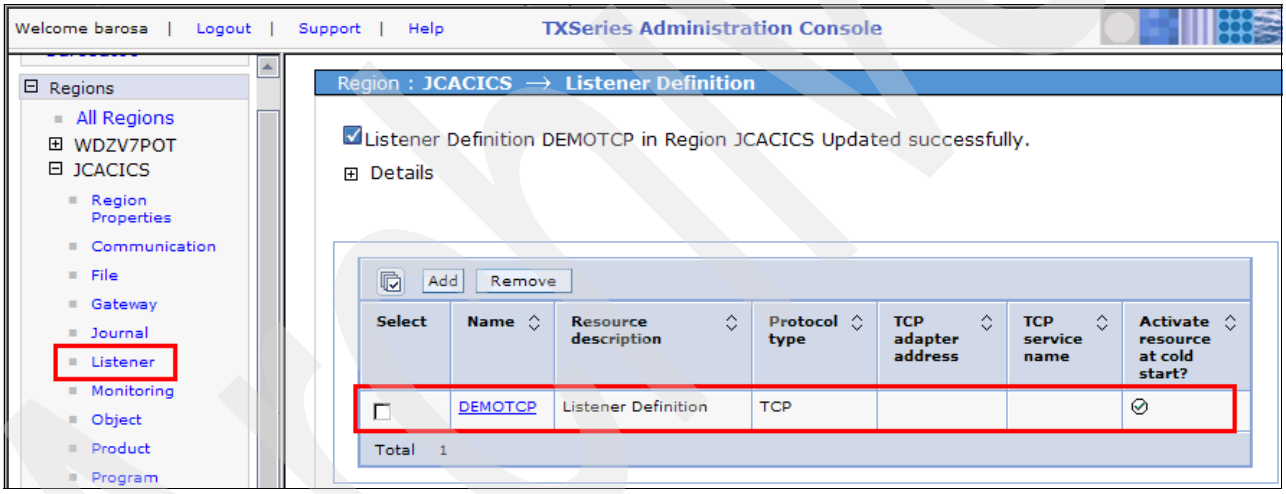


Figure 14-6 TCP/IP listener named DEMOTCP defined for TXSeries

A *cold* start is required for the CICS server (JCACICS) to activate this listener.

14.2.6 Configure CTG to communicate with TXSeries

In our example, we use TCP/IP and assume that TXSeries and the CTG are on the same Windows machine. The steps to configure CTG are listed here:

1. Start CTG Configuration Tool by selecting **Programs** → **IBM CICS Transaction Gateway** → **Configuration Tool**. If the CTG does not start and you get the message "CCL8807E Unable to find a JVM™", consider installing a Java Runtime Environment (JRE™) that is shipped with CTG.
2. Create a CICS Server definition. This task must be done for each CICS SERVER that you want to connect to. From the IBM CICS Configuration Tool window, select **Tools** → **New Server**.

3. When the dialog opens, type in the server name, such as DEMOTCP, TCP/IP as the protocol, and localhost as the host name, as shown in Figure 14-7.

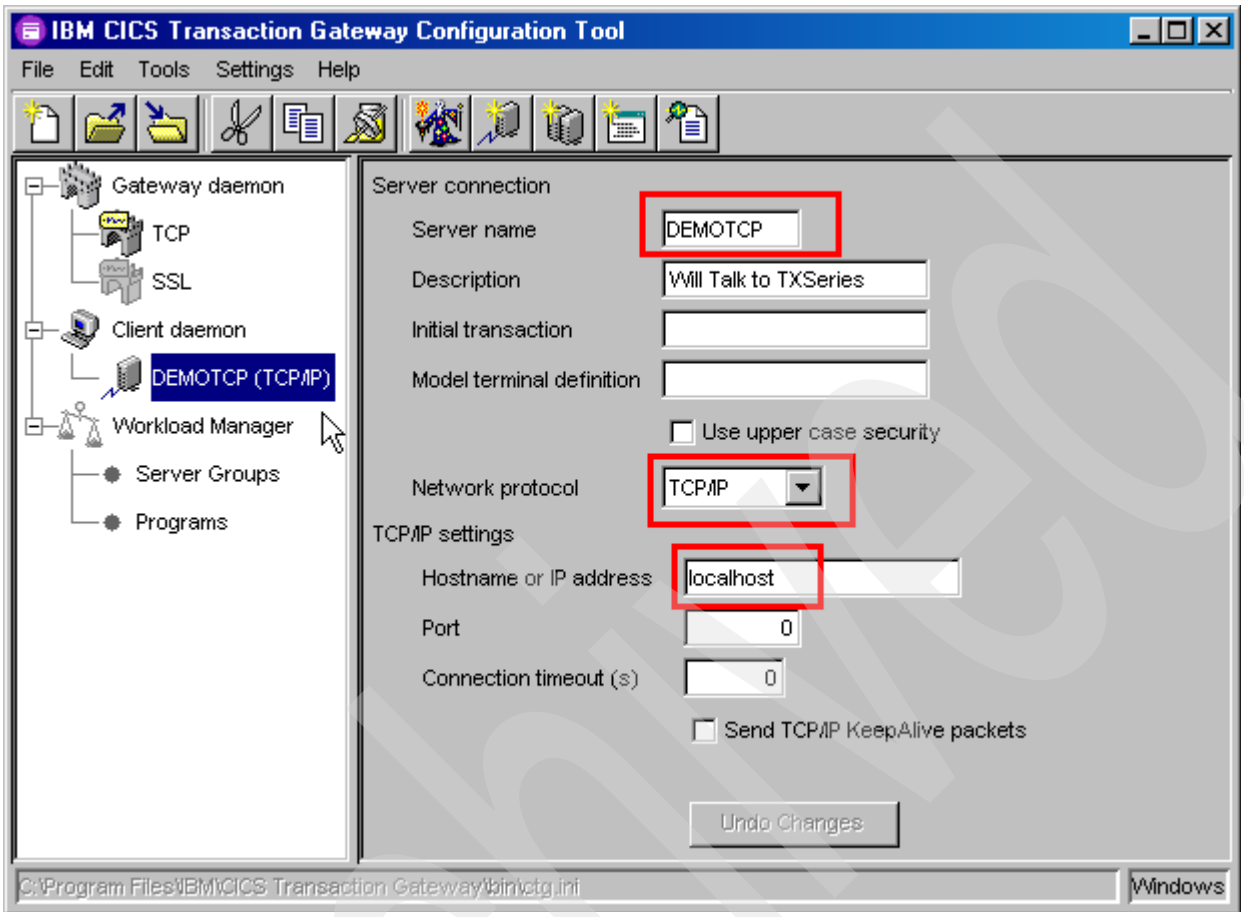


Figure 14-7 Configuring CTG client

4. Select the first left icon (**TCP**) and click **Enable protocol handler**. You can use the default port number (2006**)**. This is the port number that must be used in the programs that will communicate with CTG that will communicate with CICS. See our example in Figure 14-8 on page 499.

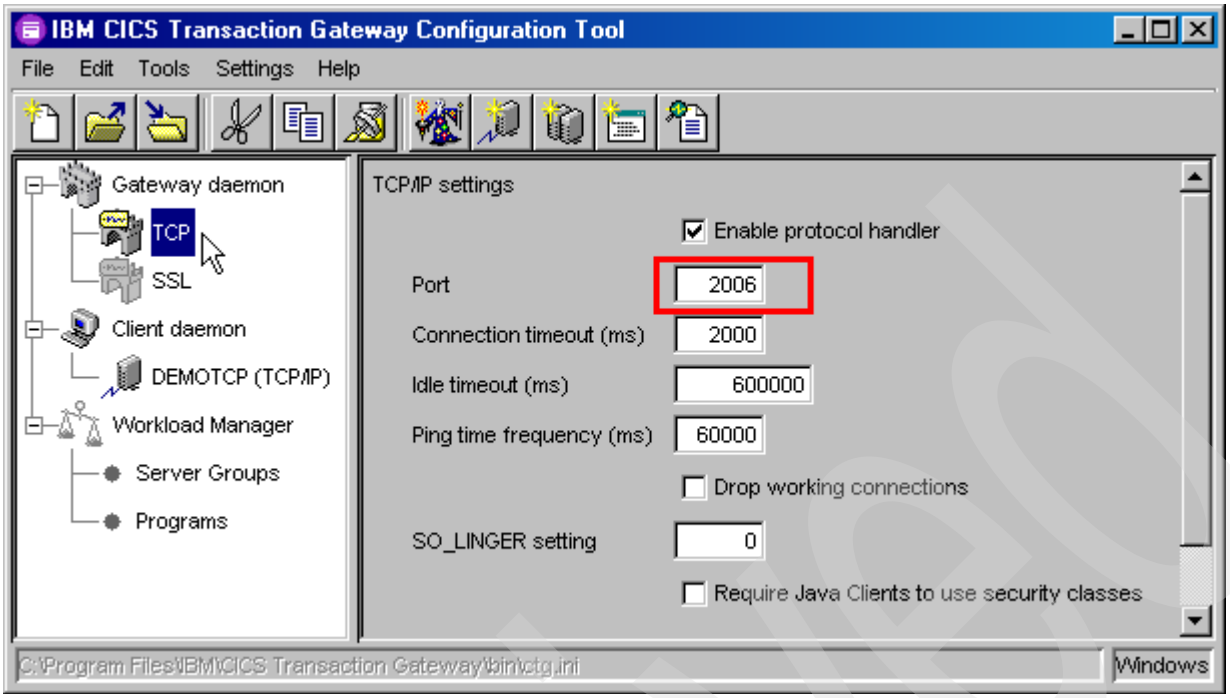


Figure 14-8 Configuring TCP Java gateway

5. Select **File** → **Save**. This will save the values into CTG.ini, which is located in the bin directory of your installation root. At this point, your CTG is ready to "talk" to the TXSeries.

14.2.7 Testing if CTG is communicating with TXSeries

To be sure that CTG is correctly configured and communicating with TXSeries, do the following steps:

1. Start the gateway at the operating system command prompt of the computer on which you have installed it, for example, **Programs** → **IBM CICS Transaction Gateway** → **IBM CICS Transaction Gateway**. The program will start and the window shown in Figure 14-9 must be active; note that CTG will listen through TCP and port 2006. Do not close this window; just minimize it.

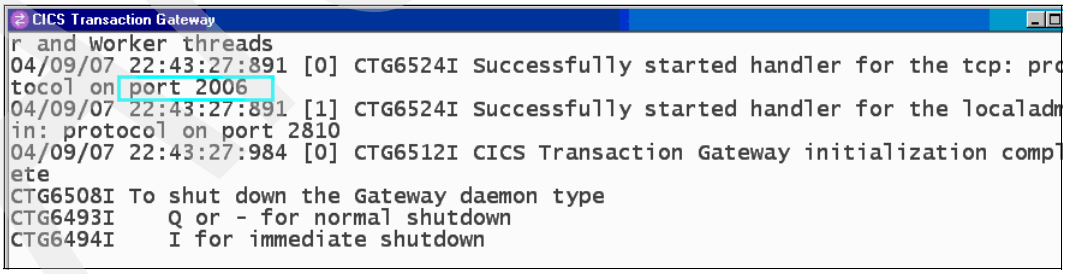


Figure 14-9 CTG window after start

2. Be sure that TXSeries and CTG are started and start the CICS Terminal by selecting **Programs** → **IBM CICS Transaction Gateway** → **CICS Terminal**. This command will start the 3270 terminal emulation. If your configuration is correct, you will see a screen that is going to CICS Server through CTG.

If you have problems with the DEMOTCP listening, you may need to stop the Windows service named IBM CICS Universal Client, and be sure that no software firewalls are blocking this communication because we are using TCP protocol.

3. Type any transaction, such as xxx, just to be sure that you are communicating to CICS. Note that the Enter key is the right Ctrl key. If you typed xxx, a window like the one shown in Figure 14-10 should appear, because this transaction is not defined. If CTG is not correctly configured, you will have an error message and you need to fix this before continuing with the next step.

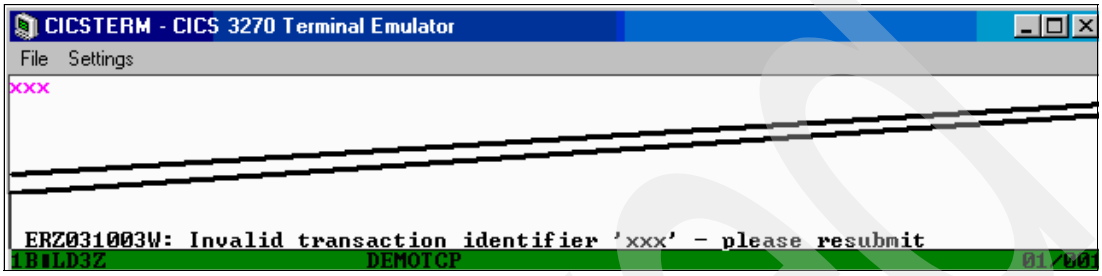


Figure 14-10 Emulating 3270 terminal using CTG

14.2.8 Load the sample COBOL into WD/z, compile/link, and create the executables

We provided two COBOL programs: one client program that shows a BMS map that will ask for an employee number, and a server program that is invoked by the client through EXEC CICS LINK, which reads the employee data from the STAFF DB2 table.

You can now start WD/z. It is a good idea to use a clean workspace, so it will be easy for you to find the generated code. In this example, we use a workspace located in C:\WorkspacesV7\JCADEMO.

At this point, you must load the sample COBOL program that is provided with this example. Unzip the COBOL_TXSERIES.zip file (instructions for downloading this file can be found in Appendix B, “Additional material” on page 645) to the C:\temp directory and follow the instructions below. Note that the executable is also delivered with the sample code and, optionally, you can bypass this section and install the *.ibmcob in the TXSeries.

1. Import the client and server programs into two separate projects using Project Interchange. Using the z/OS Projects Perspective from the Workbench, select **File** → **Import** → **Other** → **Project Interchange**, select **c:\temp\LOCAL_COBOL_CICS.zip**, and select the **LOCAL_COBOL_CLIENT** and **LOCAL_COBOL_SERVER** projects.

   In Figure 14-11 on page 501, you can see those assets after they are loaded with the two COBOL programs.

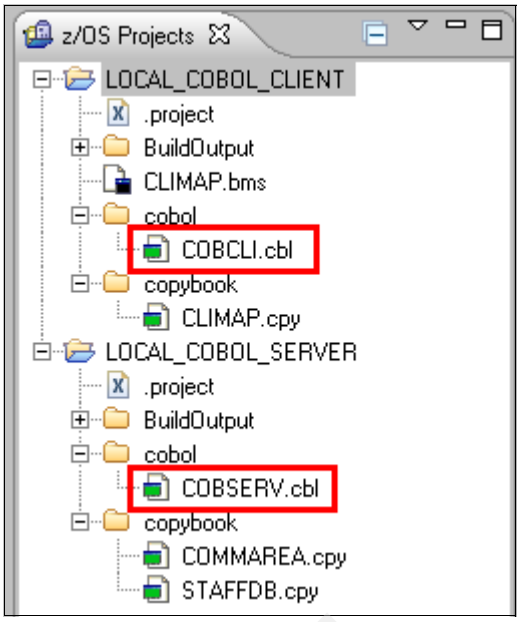


Figure 14-11 Provided COBOL client/server programs

2. Import the properties from COBOL_CLIENT_PROPERTIES.xml and COBOL_SERVER_PROPERTIES.xml. This can be done by right-clicking each project and selecting **Import Properties...**. Verify the COBOL compile properties match your installation. One example for the server program can be seen in Figure 14-12. Note that the SYSLIB is pointing to the copybook, assuming that you are using the workspace on C:\WorkspacesV7\JCADEMO. If this is not the case, you must correct it to avoid compile errors due to missing copybooks. The same would apply to the SQL required copybooks that usually are located at c:\sqllib\include\cobol_a;.

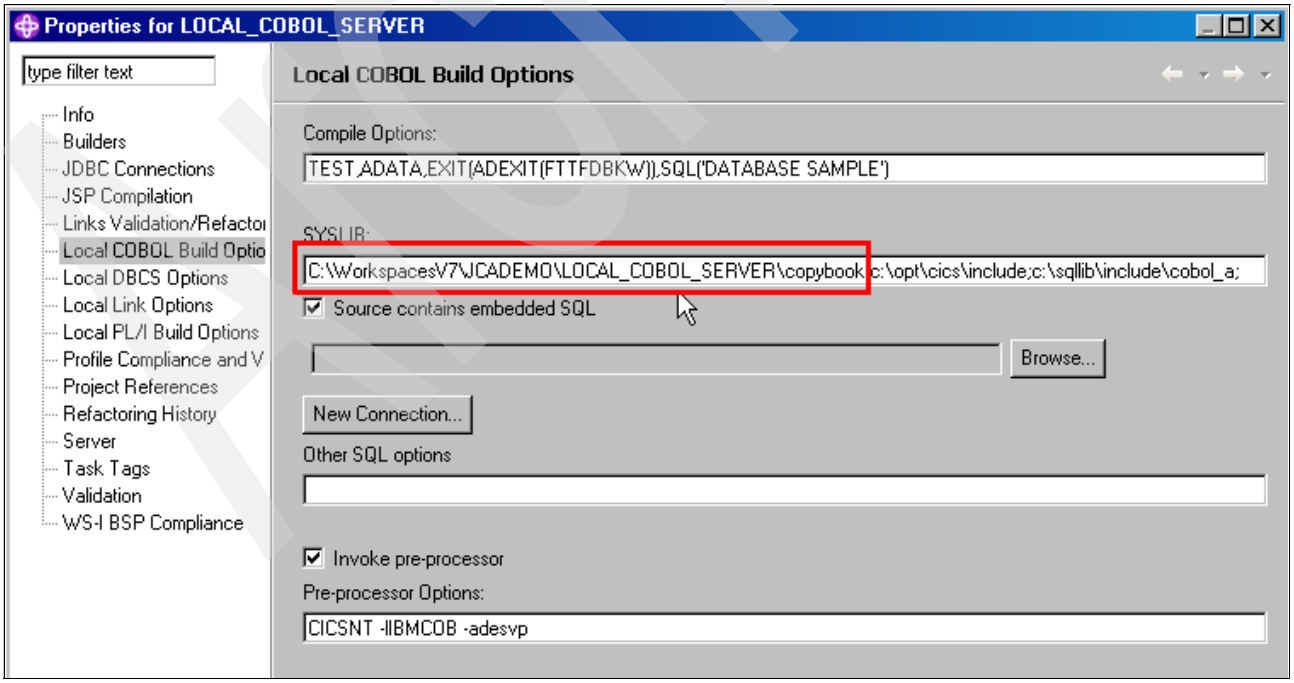


Figure 14-12 Local COBOL Build Options

3. Build the program to generate the executables and deploy to TXSeries.
   - Using the z/OS Projects view, expand **LOCAL_COBOL_CLIENT**, right-click **COBCLI.CBL**, and select **Nominate as Entry Point**. If you do not do this task, one DLL will also be generated with the project name. Repeat this operation for COBSERV.CBL.
   - To generate the executables, right-click the projects and select **Rebuild Project**. Figure 14-13 shows the executables created for the Client project under the BuildOutput folder. Similar executables will be created for the Server project. Later you will need to export the *.ibmcob to TXSeries.

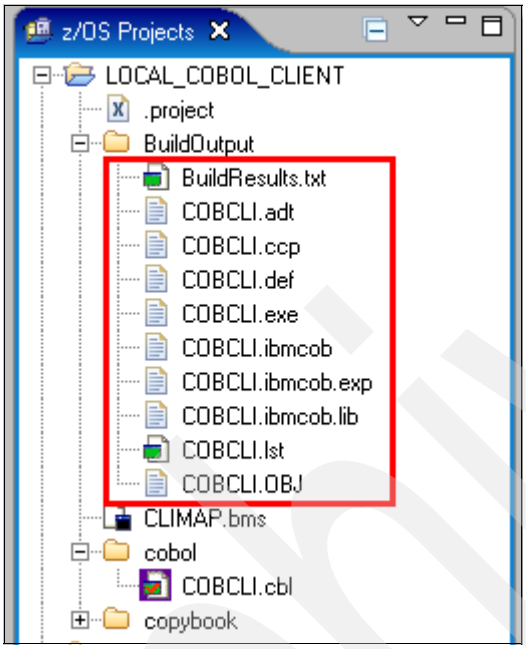


Figure 14-13 COBOL BuildOutput

Note that for the Server program a DB2 package is also created and can be seen using the DB2 Control Center. The package COBSERV can be seen under the database SAMPLE, Application Objects, and Packages.

   - To deploy the executables to TXSeries, select the **COBCLI.ibmcob** and **COBSERV.ibmcob** files under the BuildOutput folders, go the Menu actions, select **File** → **Export ...**, and choose **General** → **File system** from the Export dialog. Be sure that you are exporting to the directory C:\opt\cics\bin. This directory is the default location to deploy TXSeries applications to. There are other ways to have this *.ibmcob visible to TXSeries, such as using classpath variables or specifying the path in the TXSeries.

4. Deploy the BMS Map to TXSeries.
   - Export CLIMAP.bms to a work directory and execute the following command:

     ```
     cicsmap CLIMAP.bms
     ```

     Depending on the version you are using, you might see a warning returned from the map translation. This is a known bug and can be ignored. It will be fixed soon. The END statement within the source is not being accepted by the translator. This will not impact the use of this map in CICS.
   - Move the created map to the TXSeries location using the following command:

     ```
     xcopy CLIMAPm.map C:\opt\cics\maps\En_US   /y
     ```

Archived

14.2.9 Install the generated programs and map into TXSeries

After having created the TXSeries components, you need to define the TXSeries resources using the TXSeries Administration Console.

You need to add the two COBOL programs COBCLI and COBSERV and the Map CLIMAP as TXSeries resources, as shown in Figure 14-14.

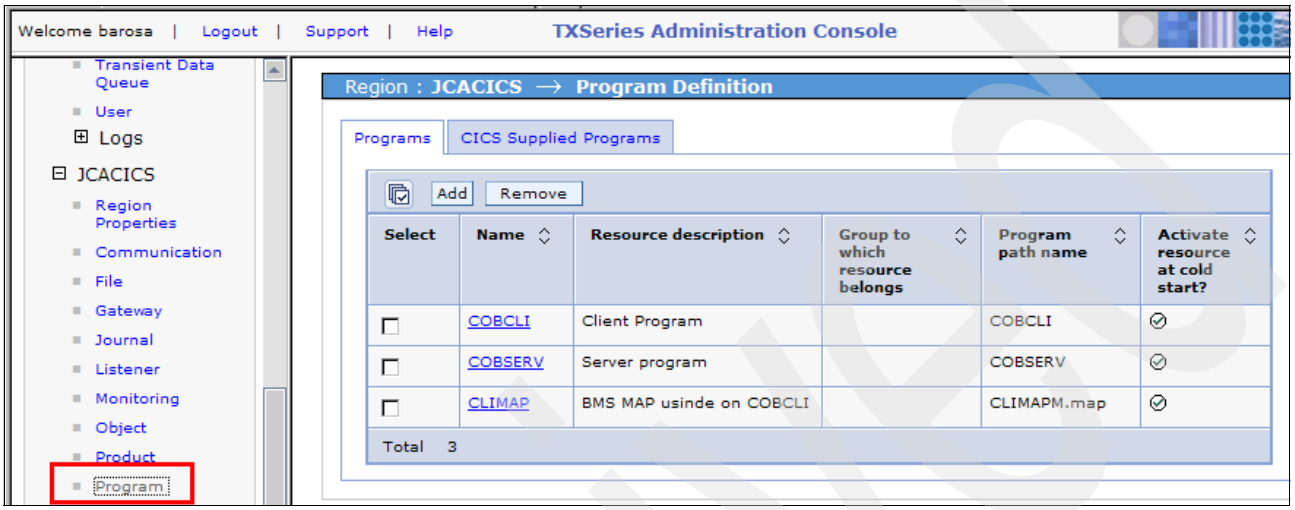


Figure 14-14 Program definitions for TXSeries

Archive

Be sure that you have specified Program type MAP for the map and that the program path name is CLIMAPM.map, because this is the object that you have created and copied into C:\opt\cics\maps\En_US. Also, be sure that the COBSERV program has no security issues and is defined, as shown in Figure 14-15.

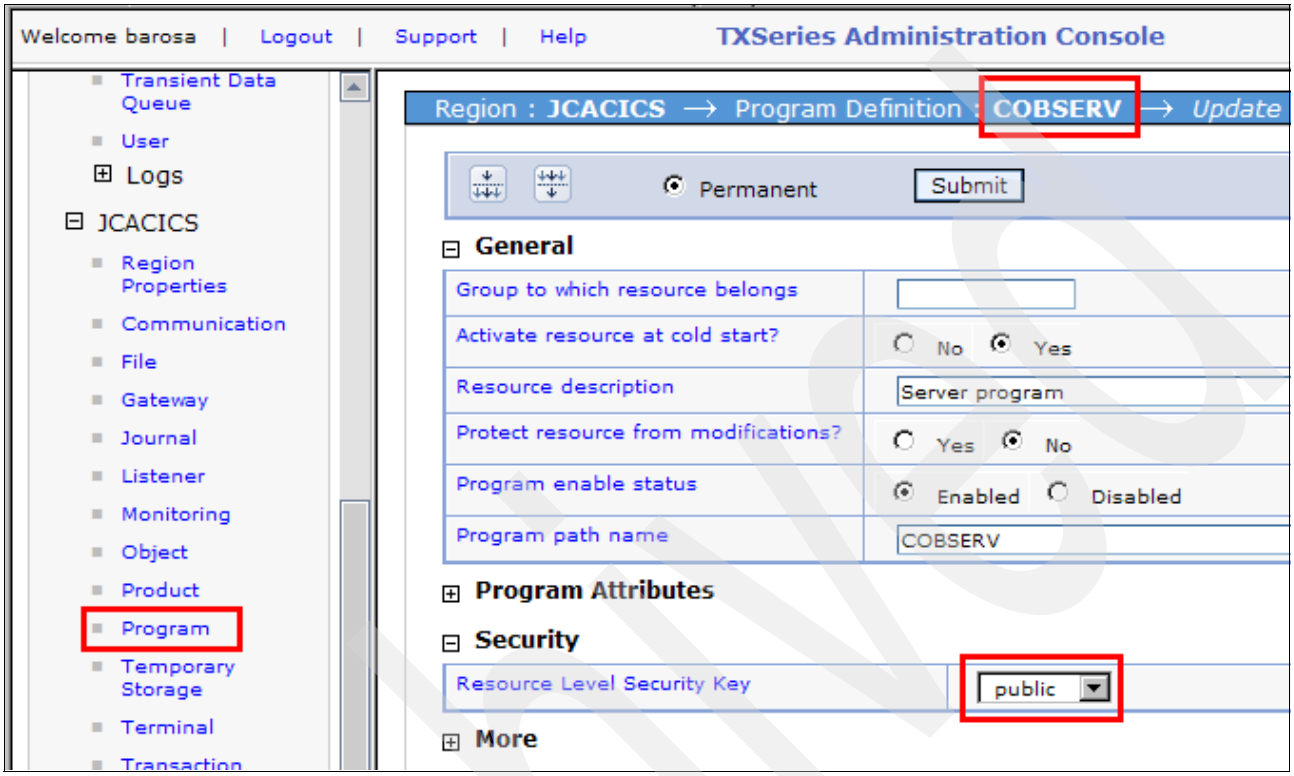


Figure 14-15 COBSERV program must have Resource Level Security Key = public

You also need to define the transaction that invokes the COBOL client program. In our example, we named this transaction JCA1, as shown in Figure 14-16 on page 505.

Archive

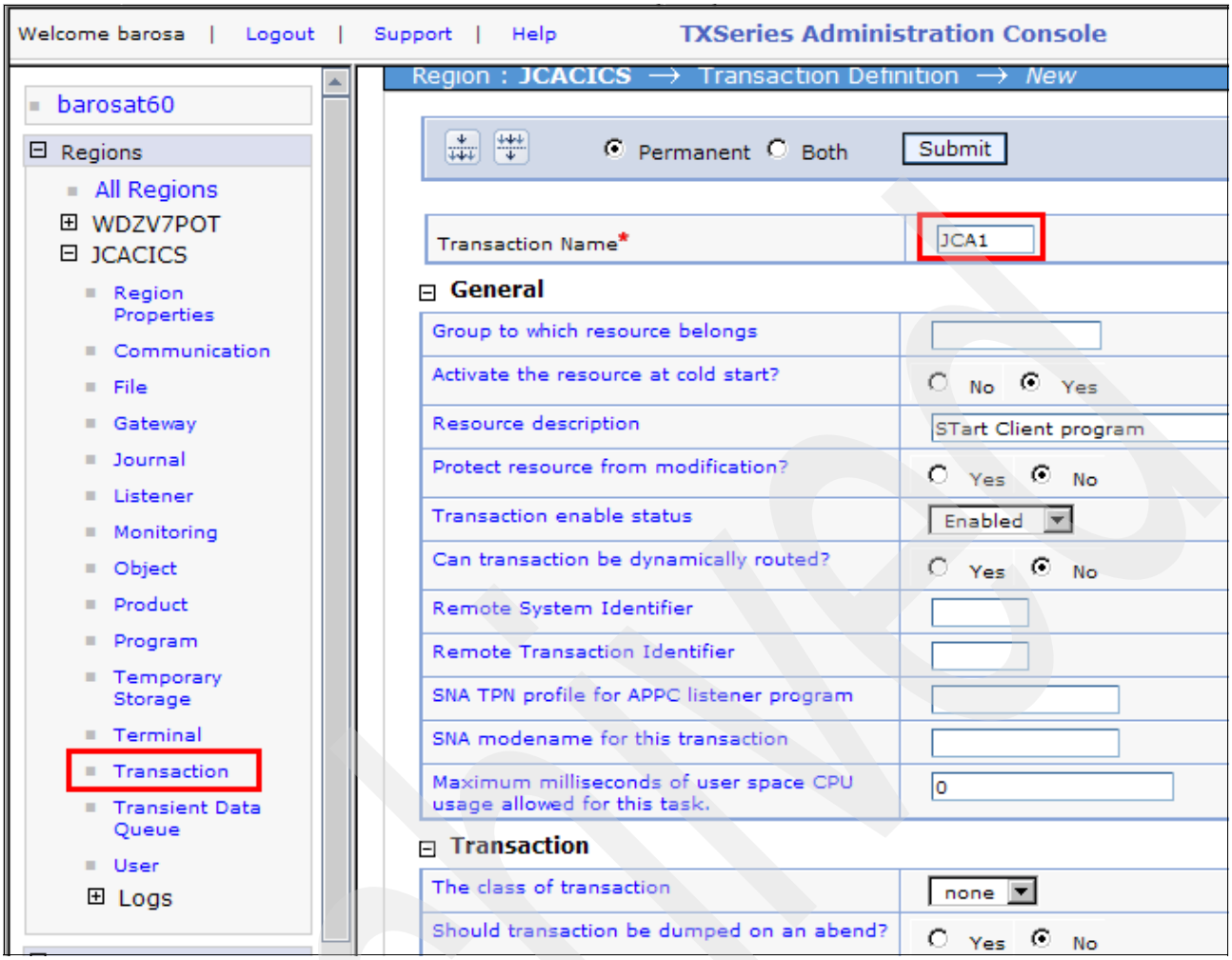


Figure 14-16 JCA1 Transaction definition

14.2.10 Test the provided Client/Server COBOL programs

Now you can test the deployed resources and its definitions. You may need to stop and start the region JCACICS again. If necessary, use a *cold* start.

Archive

Using a CTG CICS Terminal, type in the transaction JCA1 that you have defined and, if all is done correctly, you will see the map. Use some known data like ID = 020. Figure 14-17 shows this execution if everything is working fine.

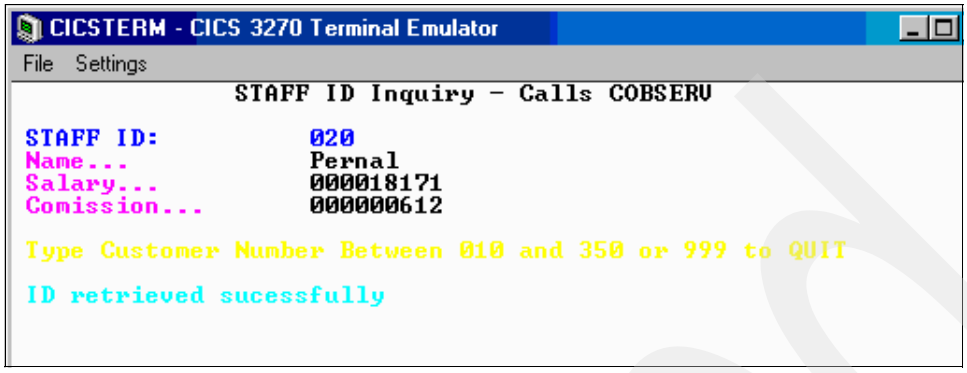


Figure 14-17 JCA1 transaction being executed

At this point, you have successfully deployed a COBOL Client Server program that accesses DB2 and retrieves some data. Type 999 to end this application.

14.3 Build a J2C application to invoke the COBOL/CICS Server

At this point, you must have the necessary code deployed to TXSeries and you are ready to use the WD/z wizards to create the J2C connectors that will invoke the COBOL server program.

We show you how to use the J2C Java Bean wizard to connect to a CICS ECI server. While completing the exercises, you will:

- Use the J2C Java Bean wizard to create a J2C application that interfaces with a CICS transaction using an External Call Interface (ECI).
- Create a JSP test client to test the Java bean created that will invoke the COBOL/CICS Server and return the employee details.

The main steps to be executed are:

1. Create a Java bean that invokes the COBOL/CICS Server.
2. Test the CICS ECI Connectors using JSP Test Client.

14.3.1 Create a Java bean that invokes the COBOL/CICS Server

1. Using the J2EE perspective, start the JC2 Java Bean Wizard by selecting **File** → **New** → **Other**. The Select a Wizard dialog box appears. Type in J2C and select **J2C Java Bean**, as shown in Figure 14-18 on page 507, and click **Next**.

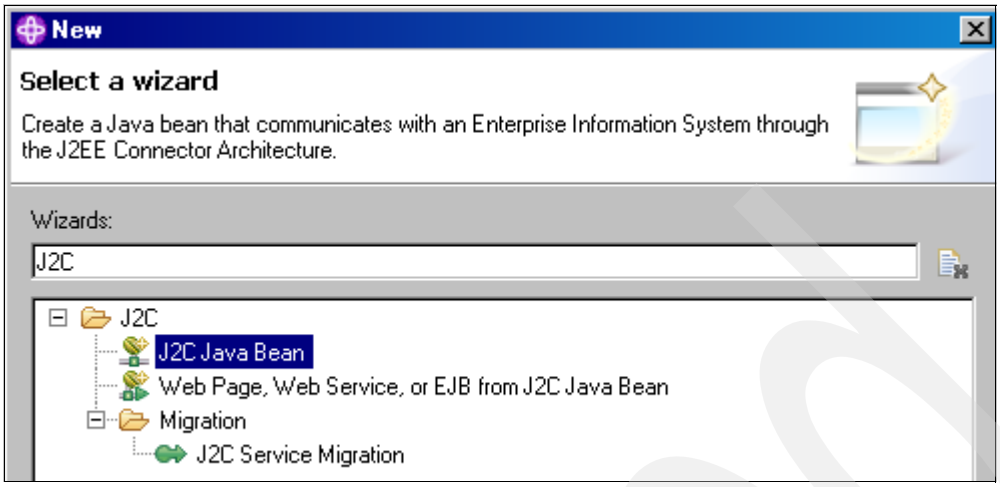


Figure 14-18 Selecting the J2C Java Bean wizard

2. You must select the appropriate resource adapter for the J2C Java Bean. Click **ECIResourceAdapter (IBM 6.02)**. This is the resource adapter that will be used with TXSeries. Click **Next** to continue.

Once you have selected the resource adapter, you must provide the connection properties information before this selection is made. Let us see what the possible options are.

Managed and non-managed connection properties

Connections from the J2C Java bean to the CICS (TXSeries) can be Managed or Non-managed.

A managed connection runs inside an application server, such as WebSphere Application Server. With a managed connection, the application server provides connection management with pooled connections, and the application server can send a user ID, a user ID and password, or no security information, depending on how it is set up.

When using a managed connection, you must define the application server with the appropriate connection characteristics, and usually this connection information is maintained by the system administrator. This is a better option and provides more flexibility; as connection information changes (the type of communications, the CICS port, and so on), the system administrator can adjust the connection characteristics and no Java objects need to be regenerated. Managed connections can be available to servlets, JSPs, and EJBs, because they run inside an application server.

A non-managed connection is designed to run where connections management supplied by an application server is not available. In the non-managed connection, the characteristics of the connection are hardcoded into the generated object. You can change the connection characteristics from your program using this generated code, but you will need to generate you J2C code appropriately. Because non-managed connections are not always convenient to change, you do not have the performance gains of a connection pool managed by an application server, and cannot rely on an application server to provide security information; it is easy to see why managed connections are recommended

For this example, we use a non-managed connection to CICS (TXSeries), and this is the way we recommend for simple tests, because it does not required configurations on the application server.

3. From the Connection Properties window of the New J2C Java Bean Wizard, de-select **Managed Connection** and select **Non-Managed Connection**. Accept the default connection class name of com.ibm.connector2.cics.ECIManagedConnectionFactory.

   Be sure that **TCP/IP** is checked and enter the required connection information:

   – Host name: localhost (because this setup is all on the same workstation)

   – Server name DEMOTCP and Port number 2006 (these are the definitions created on the CTG Configuration tool)

   – User name *your-windows-logon-id* and password *your-password*, as shown in Figure 14-19.

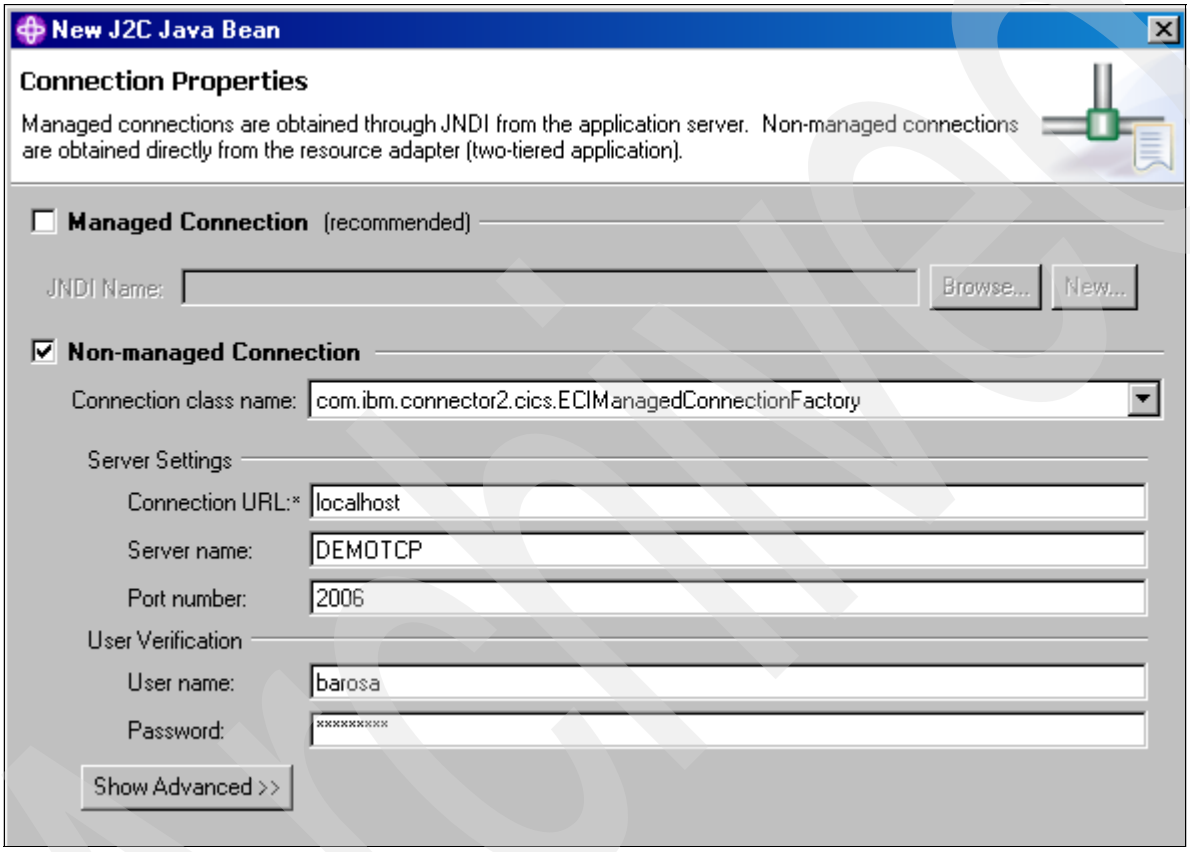


Figure 14-19 Defining the non-managed Connection Properties

   Click **Next** to continue.

4. In the J2C Java Bean Output Properties window, type JCA_TO_TXSeries as the project name and click **New** to define the project properties.

5. In the New Source Project Creation window, select **Web project** and click **Next** to continue.

   We want to create the JCA Java Bean using a Web Project so we can store the JSPs, WSDL, and so on.

6. From the New Dynamic Web Project window, select **WebSphere Application Server v6.1**, leave the Configurations setting as <custom> and check the **Add Project to an EAR** check box to allow the wizard to add “EAR” to your project name. Type JCA_TO_TXSeriesEAR to create the project and click **Finish** to create the Dynamic Web Project, as shown in Figure 14-20 on page 509.

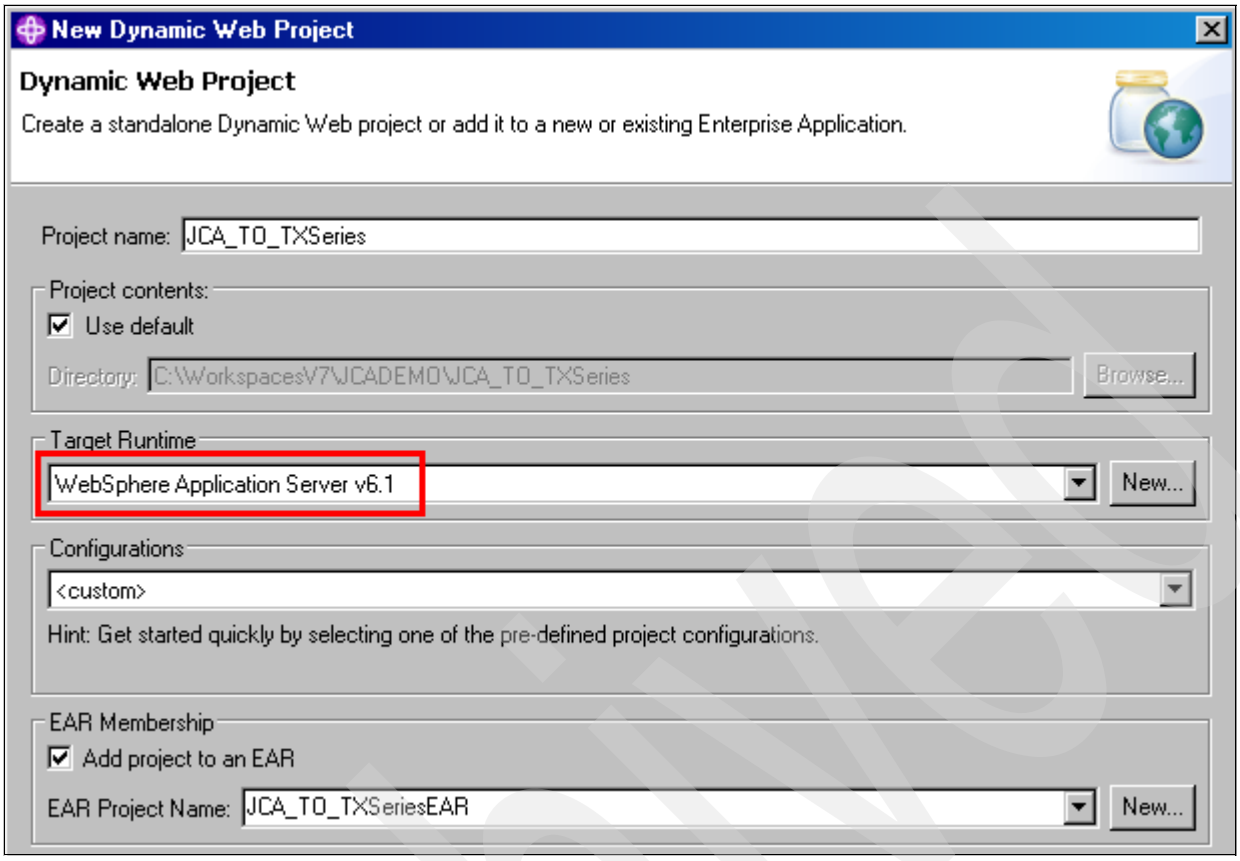


Figure 14-20 Creating a Dynamic Web Project

WD/z will now complete the creation of the J2EE components that support the J2C Java Bean. Notice that both a Dynamic Web Project has been added to your work space.

Archive

7. From the J2C Java Bean Output Properties window, use the project name JCA_TO_TXSeries and type he names for the Java components that will be created. Fill in the following required fields: demo.cobol.cics in the Package Name field, CallCICSCOBOL in the Interface Name field and CallCICSCOBOLImpl in the Implementation Name field, as shown in Figure 14-21, and click **Next** to continue.

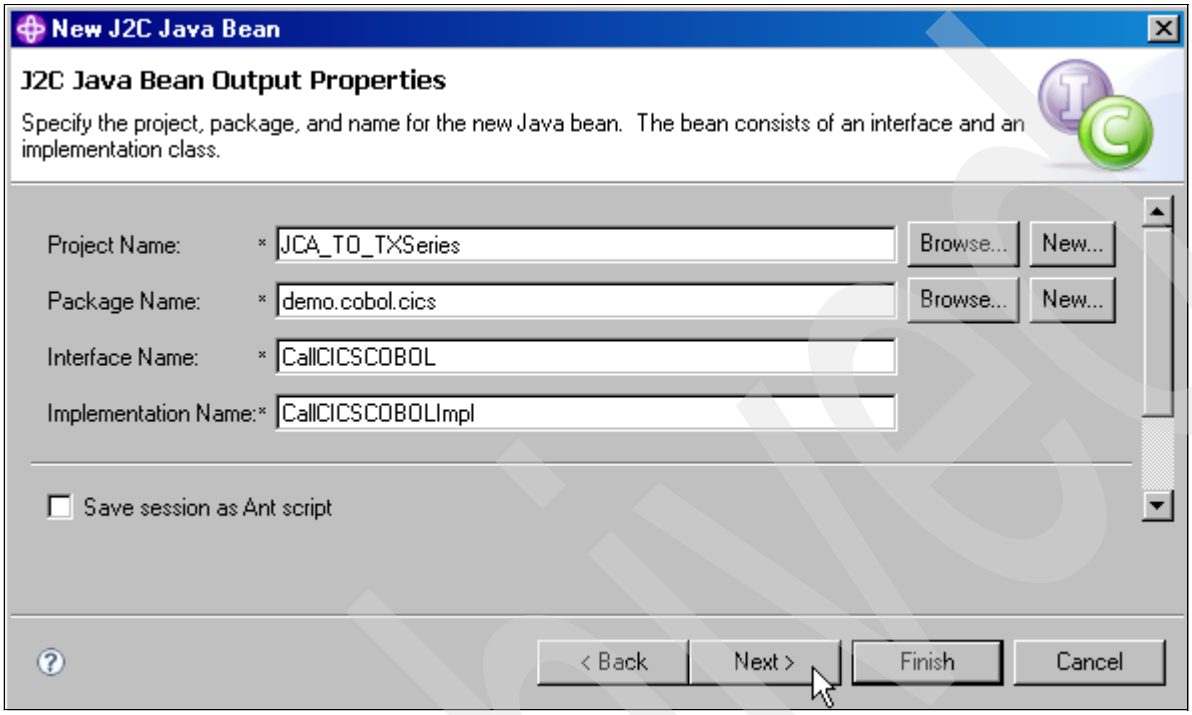


Figure 14-21 J2C Java Bean properties

8. In the Java Methods window, click the **Add** button to add a Java method to the demo.cobol.cics package defined in the previous window and, from the Add Java Methods window, enter staff_Detail as the Java Method Name and click **Next** to continue.

   Now you will create the input and output data mappings between COBOL and Java. In this step, you import the data definitions from the COMMAREA.cpy used in the COBCLI COBOL program.

9. Click **New** (next to Input type), create the import mappings for the staff_Detail Java Method, and, in the Data Import window, accept the default **COBOL to Java mapping**. Click the **Browse…** button, find the provided COMMAREA.cpy file, and click **Open**. This is the DFHCOMMAREA used in our Server program. Click **Next** to continue.

10. The Importer window will open, where you define the appropriate COBOL Import Parameter settings. The default settings are for the Win32® platform. Be sure that Win32 is selected. This is the code page used for Windows.

11. Click **Query** to select the appropriate input data from the COMMAREA.cpy copybook file. Note that the COBOL copybook COMMAREA has only one level and only one data structure is shown, as shown in Figure 14-22 on page 511.

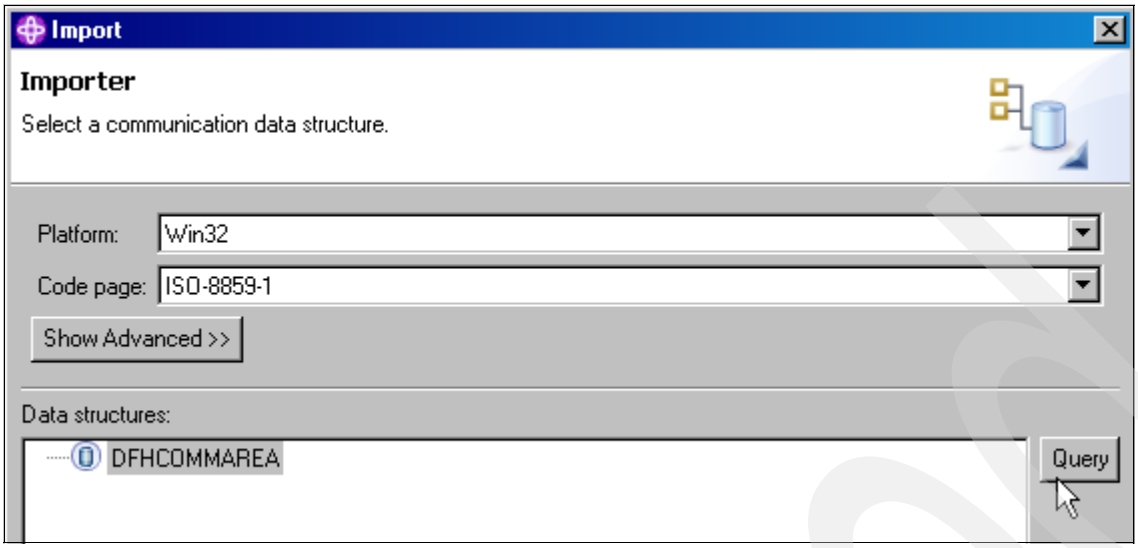


Figure 14-22 COBOL Importer dialog

12. Click the **DFHCOMMAREA** data structure to map this data area to the input record and click **Next** to continue. The Import window appears. Notice that you can change the generation style and data bindings. Accept the default generation style and Data Bindings and click **Finish** to generate the DFHCOMMAREA Java Class for the input mapping.
13. The Add Java Method window reappears. Notice that the Input Type has been defined with the DFHCOMMAREA Java Class. Select **Use the input type for output** and click **Next**. Accept the defaults and click **Finish**.
14. You need to specify the TXSeries Function name that will be called. In the Function Name field, type COBSERV. This was the name that you added to the TXSeries resource (such as CICS PPT) in the first section of this example.

Archived

Refer to the window shown in Figure 14-23. Note that if you click **Show Advanced**, you would be able to specify other CICS parameters, such as Transaction name and so on, but this optional. Click **Next** to complete the Java Method creation and create a Test page using JSP.

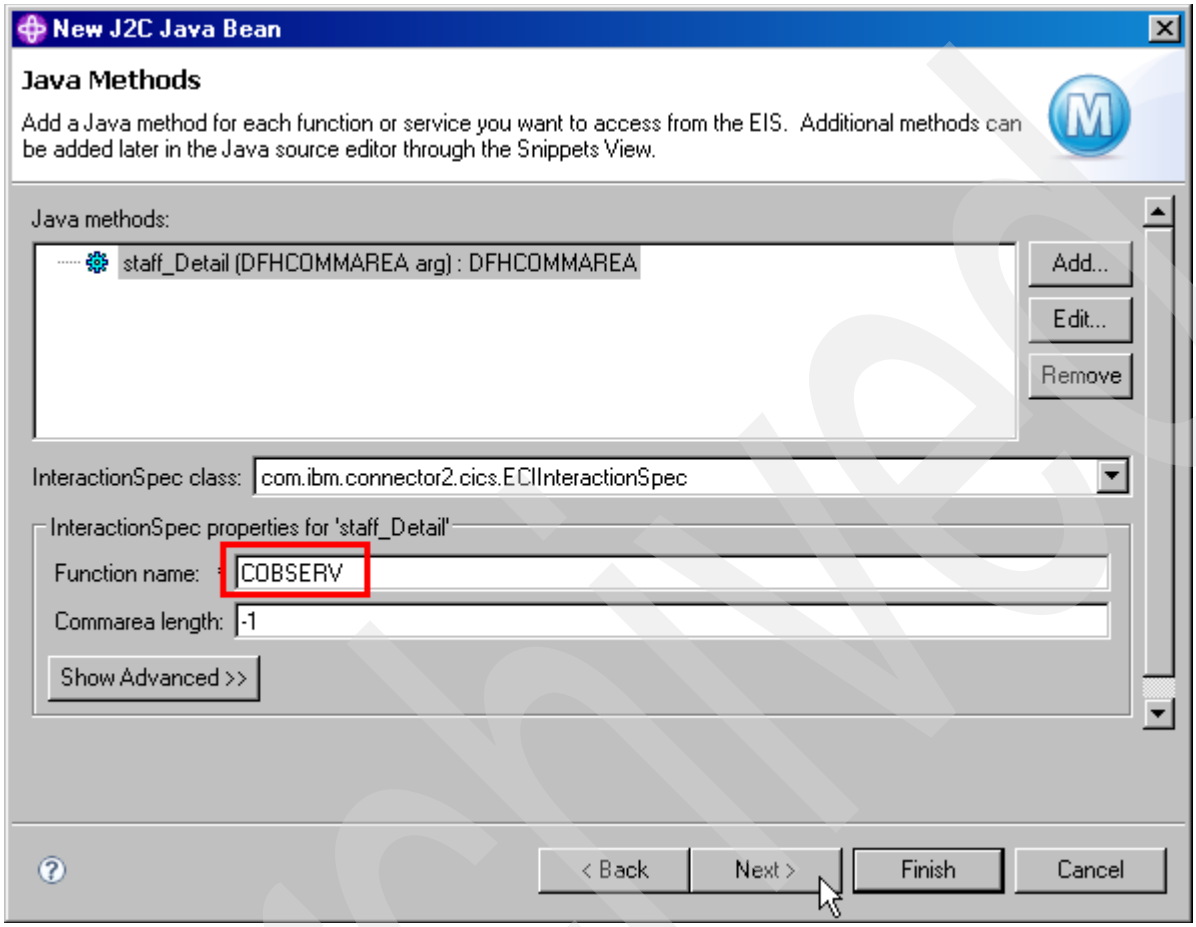


Figure 14-23 Completing the J2C Java Method

The Deployment Information page appears. From here we can complete the J2C Java Bean generation. WD/z provides additional generation capabilities (Web page, Web service, or EJB) that use this J2C Java Bean.

15. Select **Create a Web Page, Web Service or EJB from the J2C Bean**, select **Simple JSP**, and click **Next** to continue.
16. Type TXSeries for the JSP Folder name. Click Finish to complete the simple JSP.

After a while, WD/z opens the CallCICSCOBOLImpl.java file. This is the Java bean that will invoke the TXSeries server program. Look at the generated components in the project explorer under the LAB5B_JCA_TO_TXSeries project. Figure 14-24 on page 513 shows this this Java code and the TestClient.jsp generated.

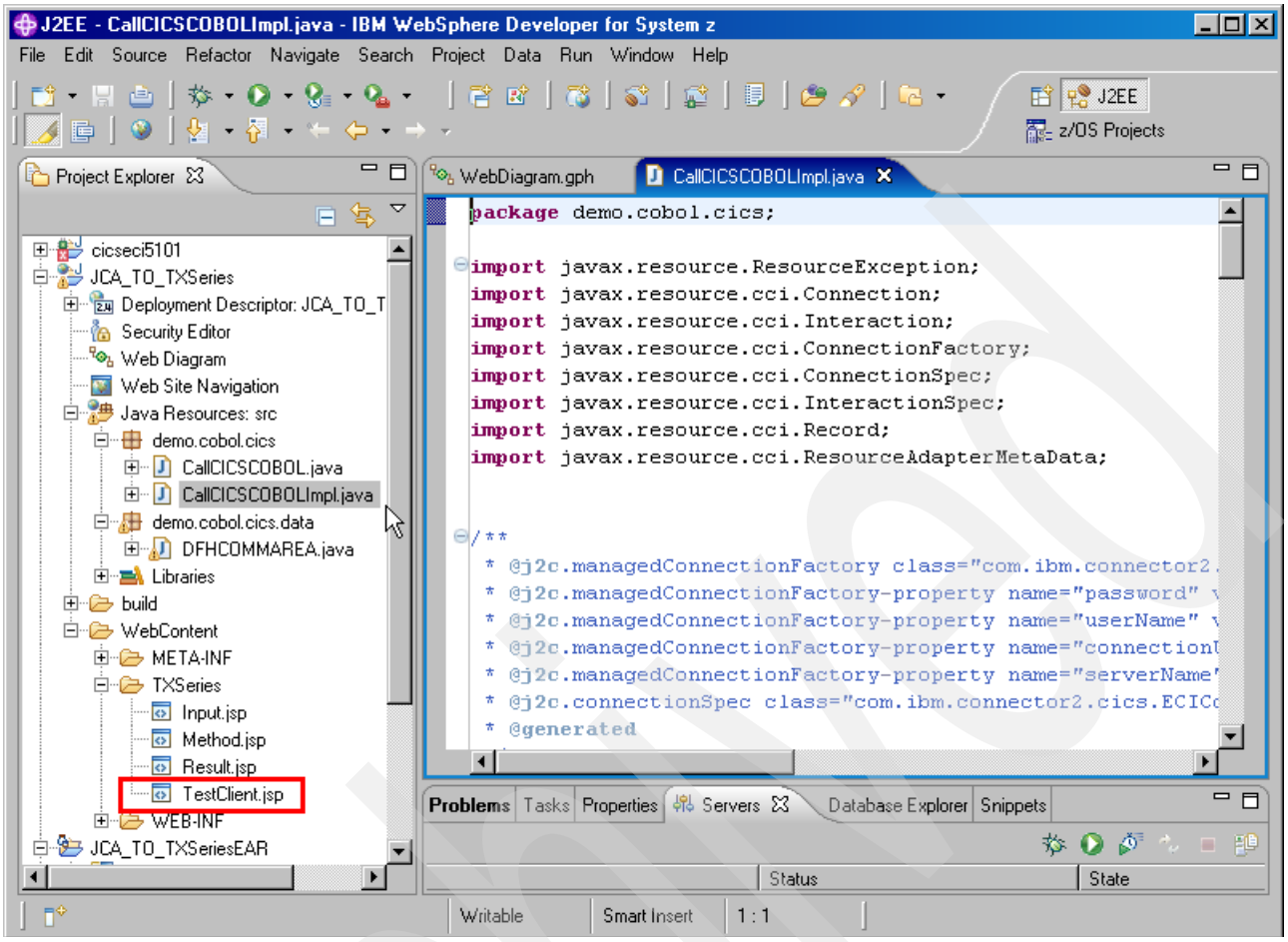


Figure 14-24 Java code and JSP generated

Note that the Java code that invokes the COBOL/TXSeries is created under packages demo.cobol.cics and demo.cobol.cics.data.

At this point the Java bean is generated, as well a JSP test client that will use this bean to invoked the COBOL/CICS Server program. In 14.3.2, “Test the CICS ECI connector using the JSP test client” on page 514, we deploy this JSP and run it.

Also, WD/z provides cheat sheets to guide you through some of its application development processes. Each cheat sheet is designed to help you complete some task, and it lists the sequence of steps required to achieve that goal. A cheat sheet is provided for this sequence that we executed above. If you want to launch this cheat sheet, select **Help → Cheat Sheets… → J2C Java Bean → Creating a Java bean using J2C Java Bean wizard for a CICS or IMS COBOL program**.

14.3.2 Test the CICS ECI connector using the JSP test client

To execute this test, you need to start the WebSphere Application Server, add your EAR project to the application server runtime environment, and test your application using the TestClient.jsp generated by the J2C Java Bean wizard. Do these steps:

1. Using the J2EE perspective, select the **Servers** view within the J2EE perspective and verify that you have WebSphere Application Server Version 6.1 defined. If you do not, you need to define it. If you need to create it, right-click in the Servers view (blank area) and select **New** → **Server** → **IBM** → **WebSphere v6.1 Server** → **Finish**.
2. Start the WebSphere Application Server, select **WebSphere Application Server V6.1**, right-click, and select **Start**. It might take a few minutes for the application server to start.

   Wait until the Status indicator says that the WebSphere Application Server is Started and the State is Synchronized. The Console window will show the Application Server messages.
3. Now you must add your project to the started application server. Right-click **WebSphere Application Server** and select **Add and Remove Projects**.
4. In the Add and Remove Projects window, select **JCA_TO_TXSeriesEAR** and click **Add** to add your EAR to the Configured Projects. Click **Finish** to add your project to the application server.

   The WebSphere Application Server will publish the EAR application, and then return to a Synchronized state.
5. Switch to the Console view and verify that the application has started successfully, as shown in Figure 14-25.

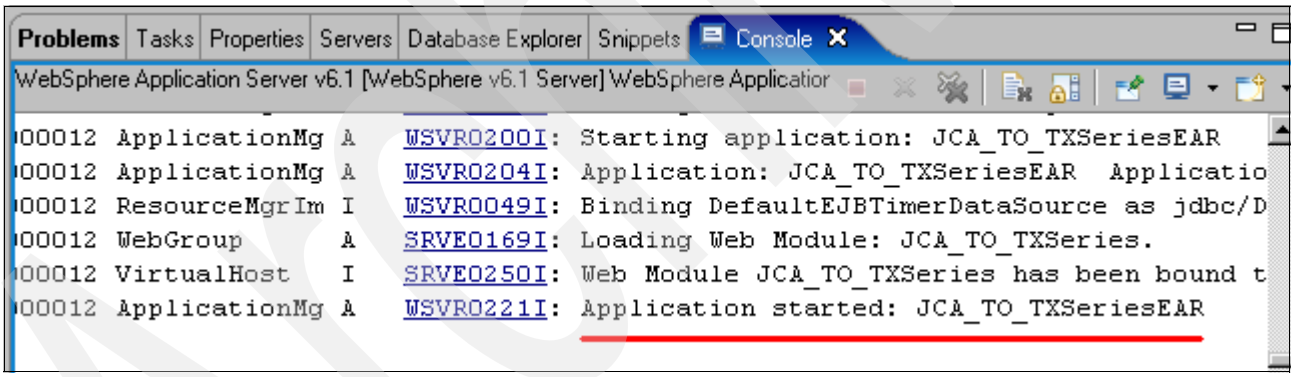


Figure 14-25 EAR project deployed to WebSphere Application Server Version 6.1

Before running this JSP, be sure that CTG and TXSeries and the required region (JCACICS) is started and the transaction JCA1 is working.

6. To run the test client, select **TestClient.JSP**, which is located in the TXSeries folder, right-click it, and select **Run As** → **Run On Server**. Choose the started WebSphere Application Server V6.1 and click **Finish**.

   The JSP will launch a Web browser where you will enter the test input data. Notice that there are three separate panes (Methods, Inputs, and Results) within the TestClient.jsp.
7. Click the **staff_Detail** method and a valid staff ID, such as 020, and type 0 in all numeric fields, as shown in Figure 14-26 on page 515, and click the **Invoke** button.

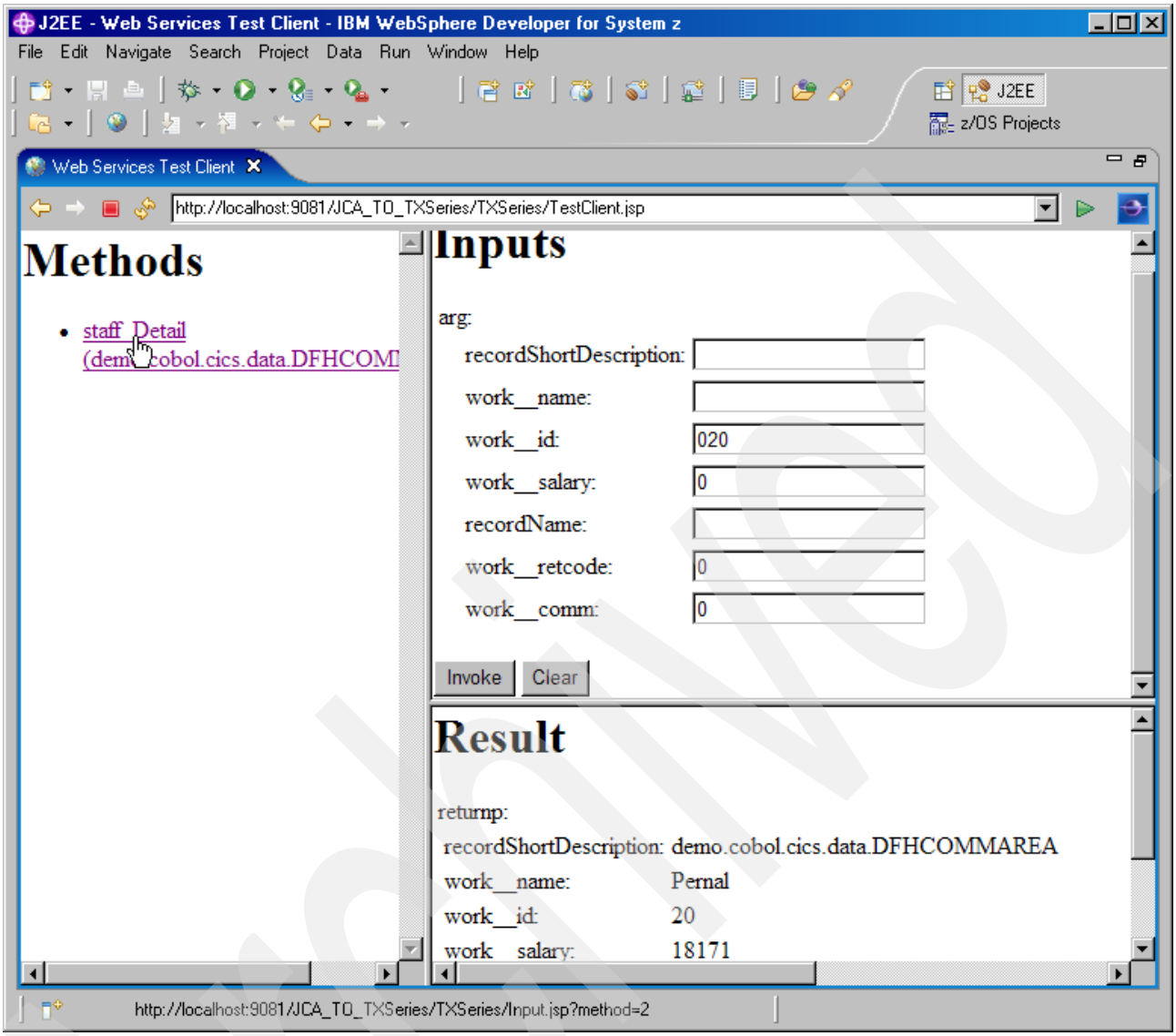


Figure 14-26 JSP Test Client in action

You also may try an invalid staff ID like 111. The work_retcode must be 100 (this is the SQL return code -100).

Note that if you do not enter 0 in the numeric fields, the parsing will malfunction and you will have a Java exception:

```
exception: java.lang.NumberFormatException: For input string: " "
```

Archive

At this point, you have successfully executed a JSP application that used the J2C Java Bean that was created to invoke the COBOL CICS program deployed to TXSeries. This COBOL CICS program retrieved the data from DB2 and returned it to the J2C Java Bean, which also passed the date to the JSP that displayed in the browser, as shown in Figure 14-27.

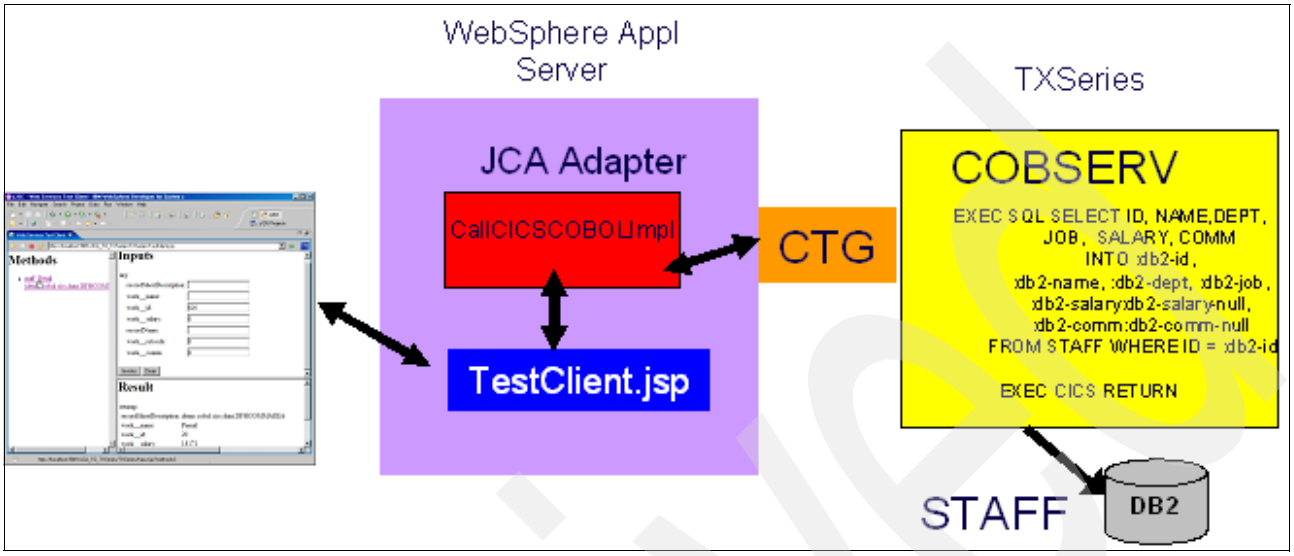


Figure 14-27 Test client invoking the COBOL/CICS Server through a JCA Adapter

14.4 Use WD/z to generate a Web service implementation from the sample application and test it using the Web Services Explorer

In the previous sections, you created a J2C Java Bean that invokes a COBOL/CICS Server and you also created a simple JSP test client that tested your J2C Java Bean. In this section, we create a Web service that uses the J2C Java Bean then test our Web service using the generated WSDL file.

The main steps to be executed are:

1. Create a Web service that invokes the J2C Java Bean.
2. Test the Web service.

14.4.1 Create a Web service that invokes the J2C Java Bean

Do these steps:

1. Using the J2EE perspective and the Project Explorer view, navigate to the J2C Java Bean implementation class. Right-click **CallCICSImpl.java** and select **New** → **Other**.
2. In the New wizard, type J2C and select **Web Page, Web Service, or EJB from J2C Java Bean** and click **Next**, as shown in Figure 14-28 on page 517.

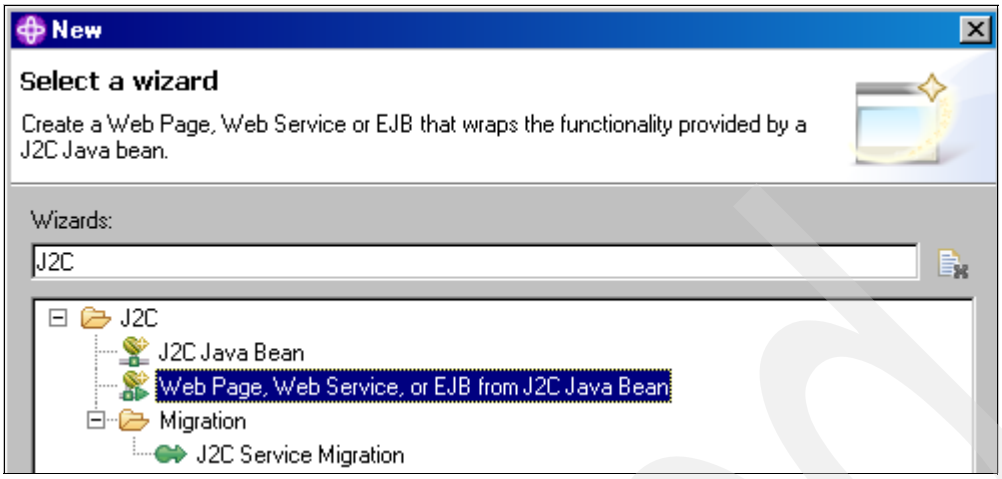


Figure 14-28 Wizard selection

3. Click **Browse** to look up the J2C Bean Implementation class, type C, select the **CallCICSImpl** class in the demo.cobol.cics package, and click **OK**. This returns the fully qualified implementation class in the window. Click **Next** to continue.
4. Select **Web Service** as the J2EE resource type. Leave **Configure Resource Adapter Deployment** unchecked. Click **Next** and then **Finish**. The J2EE Resource from the J2C Java Bean wizard will now create the necessary J2EE resources for a Web service implementation.
5. Examine the generated J2EE resources for Web Services. Expand **JCA_TO_TXSeries** → **WebContent** → **WEB-INF** → **wsdl** → **CallCOBOLCICSImpl.wsdl**.

   Note the generated WSDL file and associated XML mapping file.
6. Double click the **CallCICSCOBOLImpl.wsdl** file to open the WSDL editor. Click the **Design** tab to observe the visual representation of the WSDL file in the design view. you can also use the source view, if you prefer.

Note that the Web service operation is staff_Detail and that the operation has one input and one output, as shown in Figure 14-29.

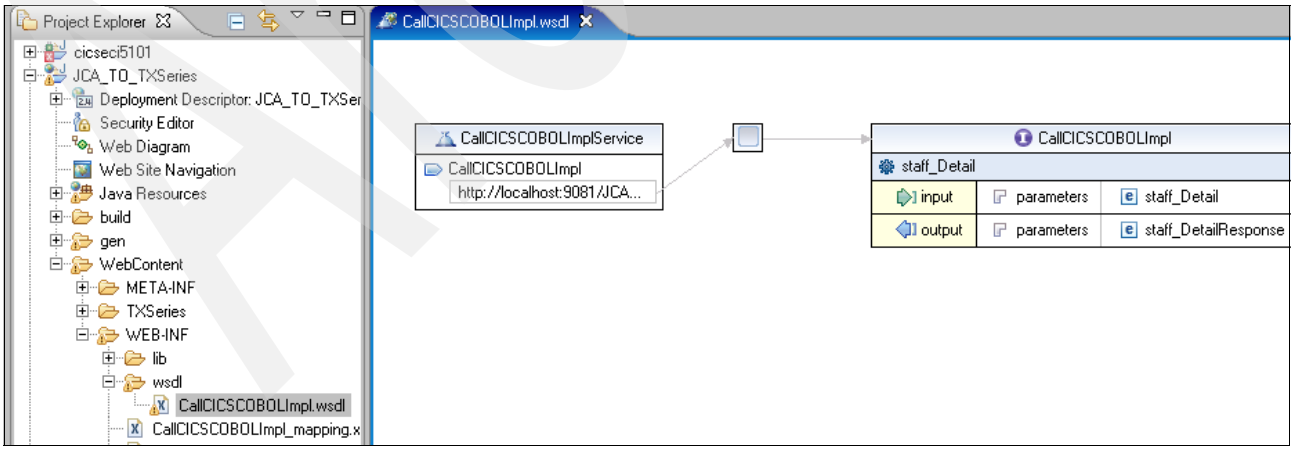


Figure 14-29 WSDL generated

Note that the generated URL usually is http://localhost:9080..., but if you have multiple application servers on your workstation, the port number could be 9080x instead. If you still have issues during the tests, you might need to change this port number. If this is necessary, select **http://localhost:9080....**, click the **Properties** view, change from 9080 to 908x, and save the changes.

14.4.2 Test the Web service

Do these steps:

1. Make sure the WebSphere Application Server is started. You test your Web service using the WebSphere Application Server test environment. If it not started, start it using the Servers view.
2. In the Project Explorer view, right click the **CallCICSCOBOL.wsdl** file and select **Web Services → Test with Web Services Explorer**.

   The Web Services Explorer window will launch. There is a Navigator view and an Actions view. The Actions view specifies both the operations and endpoints for the Web service. The endpoint is defined as:

   ```
   http://localhost:908x/JCA_TO_TXSeries/services/CallCICSCOBOLImpl
   ```

   Note that this is the location of the Web service and that it points to the WebSphere Application Server running on our local workstation. the default configuration listens on port 908x.
3. Double-click the **staff_Detail** operation to invoke it. Scroll down and enter the necessary information to invoke the COBOL CICS Server program. This will be the same data used for the TestClient.jsp.

   For example, type a valid ID, such as 020, in the work_id field and 0 in the work_salary, work_comm, and work_retcode fields and click **Go** to run the Web service.

   The Web service executes and the results are displayed in the Status window shown in Figure 14-30 on page 519. Scroll through the Status window or resize the window, as necessary, to see the results.

Archived

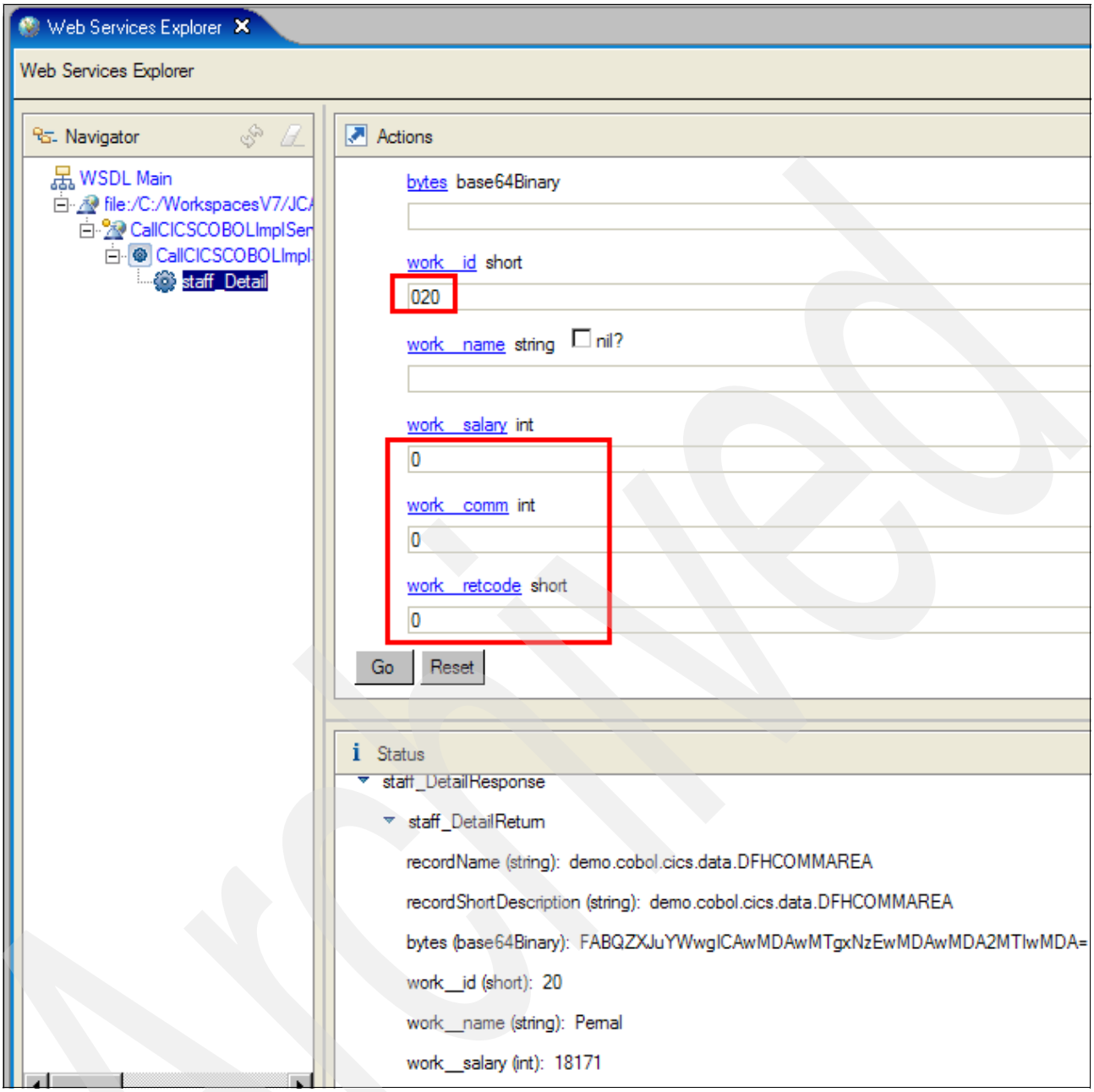


Figure 14-30 Using Web Services Explorer to test a Web service

Archived

4. If you are using SOAP as your binding type, you will have the option of toggling the Status pane between the Source and Form views. The Source view shows the SOAP request and response envelopes, including the parameters you have entered to test the service and the returned value. Click the **Source** link to display the Web service results in the Form view, as shown in Figure 14-31.

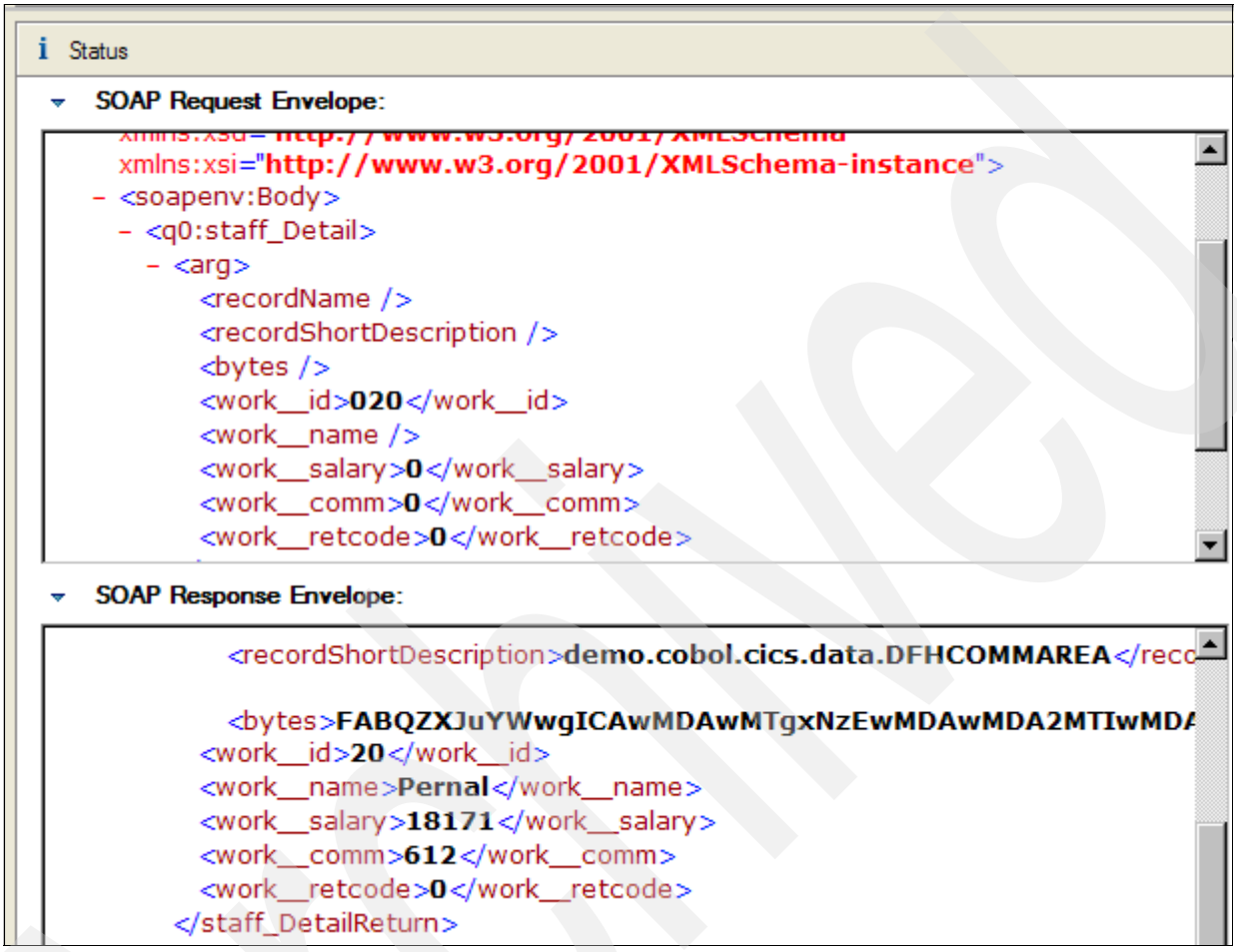


Figure 14-31 SOAP Request and Response Envelope

You may also try an non-existent ID in the DB2 table, such as 111, which will give you 100 in the field work_retcode, or an invalid ID, such as xxx, which will give you an error about validating parameters, because the ID must be a valid numeric input.

J2C Connection Factory Pool in WebSphere Application Server

The code generated in this example used the non-managed connection, but this code would run most efficiently if it takes advantage of WebSphere Application Sever's capability to pool connections to a CTG. Refer to the Help menu for information about how to create a managed connection. When creating managed connections, the application server must be configured to support it. This is the recommended way to move your code to production.

Archived

Conclusion

WebSphere Developer for System z can help developers that want to do some tests using J2C applications and do not want to use the mainframe for the development tasks. All the wizards shown here are part of the Rational Application Developer; without the ability to generate the executables from COBOL code and test the code using CICS TS and the CTG, you would not be able to execute this example.

Archived

Archived

15

Using IBM WebSphere Developer for System z with local COBOL, TXSeries, and VSAM (SFS)

In this chapter, we describe, develop, and test offloaded mainframe development on Windows workstations with TXSeries. Quite often, when mainframe connectivity is not available, local development and testing environments have to be self-contained.

There are also cases where local deployment is required for production purposes as well. In a case where DB2 is the database, workstation development and deployment with COBOL or PL/I for both online (with TXSeries/CICS) and batch are quite straightforward. In the case of VSAM data, we have to use Structured File System (SFS) files, which is a VSAM emulator on Windows. When using the SFS to store data on Windows, the CICS application continues to use conventional CICS file control commands, such as EXEC CICS READ / WRITE / DELETE, and TXSeries converts them into native SFS calls. The SFS files can be used from both COBOL as well as PL/I under TXSeries, but not in non-CICS (that is, batch programs), as there is no support for the External File Handler (EXTFH), which is required to access SFS files from COBOL or PL/I. The local VSAM option in batch mode is the STL file, which is supported by WebSphere Developer for System z (WD/z)-COBOL as well as PL/I. The STL file unfortunately has only a primary index and does not support any secondary indices. Otherwise, it can be used without any changes to the program code.

© Copyright IBM Corp. 2009. All rights reserved.

15.1 Preparation

Note: Refer to the SG247482-Chapter 14.zip file for more information. You can download this file by following the instructions in Appendix B, “Additional material” on page 645.

Note: For the purpose of building our sample to demonstrate the use of SFS on Windows, we make copies of the following programs and maps:

- Copy COBCLI.cbl and rename it as COBCLI2.cbl.
- Copy COBSERV.cbl and rename it as COBSERV2.cbl.
- Copy CLIMAP.bms and rename it as CLIMAP2.bms.
- Store these modules in the same repository as the others.

We import these programs, map them into WD/z, and then modify them to suit our purposes.

We assume that TXSeries and CICS Transaction Gateway (CTG) have been set up and configured. We also use the same procedure as in Local_Development_1 to define the region. We use the artifacts from Local_Development_1 and modify them for our purposes. Although CTG is not needed to demonstrate the use of SFS files as VSAM on Windows, we include CTG to build simple Client-Server applications, which may be then extended or modified to suit our requirements, and moving the server program to the mainframe will not require any changes. We assume that the sample database in DB2 V8.2 has been generated, so that we can use the sample tables for our examples. For the sake of simplicity, we use one of the sample database tables, STAFF, as our reference for the VSAM (SFS) file. Some of the TXSeries’ utilities will be mentioned here for the sake of completeness. We also use some of these utilities to generate SFS files.

15.1.1 Generating an SFS file

The sfsadmin utility, as well as the CICS SFS Diagnostic Tool (cicssdt), may be used to create, query, and maintain SFS files. Both sfsadmin and cicssdt can be used in interactive mode as well as in a Windows’ bat file for these purposes. The following cicssdt example shows how to create a clustered SFS file named STAFF on SFS server /.:/cics/sfs/BITT60, with the key values sorted in an ascending order and where BITT60 is the host name.

We shall define and create a VSAM/SFS file “STAFF” to match the definition of the sample DB2 table with the same name, so that we could use the same MAP definition in CICS to display the data. We shall use cicssdt in an interactive mode as follows:

1. The TXSeries region (JCACICS, in our case) has to be started and the SFS server activated. This may be done from the TXSeries Administration Console, as shown in Figure 15-1 and Figure 15-2 on page 526.

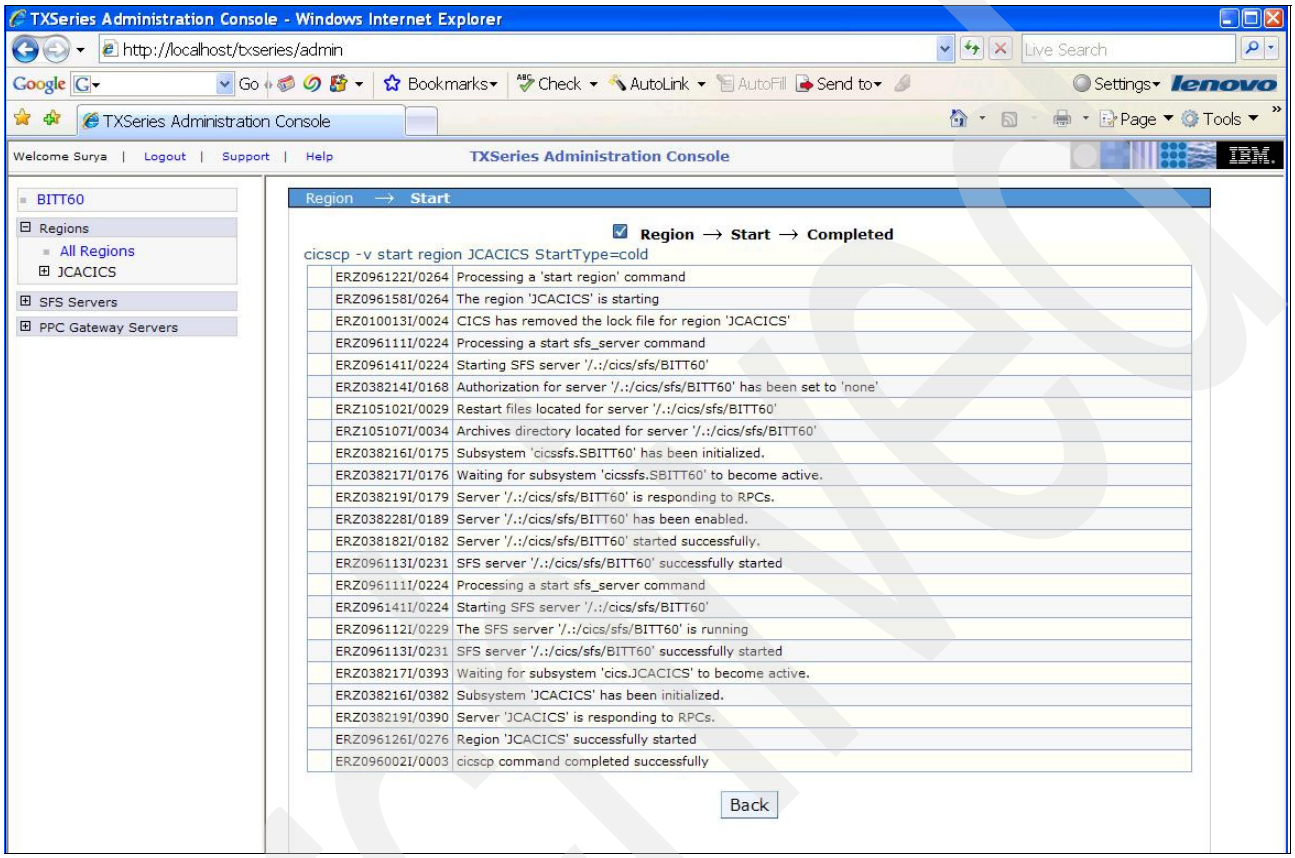


Figure 15-1 The TXSeries Administration Console starting SFS servers

Archive

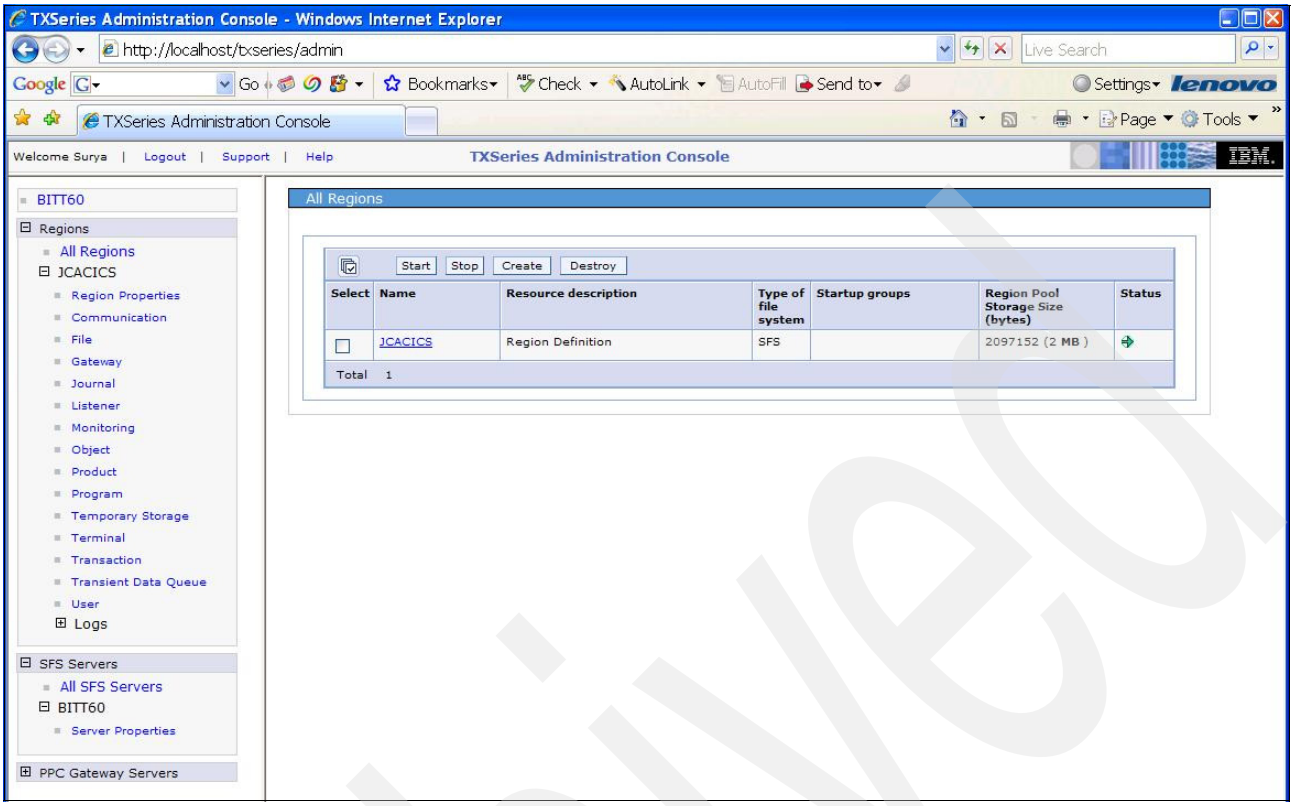


Figure 15-2 The TXSeries Administration Console starting SFS servers

2. A command line session has to be opened up to use the **cicssdt** command.
3. Run **C:\> cicssdt –s /.:/cics/sfs/BITT60**, where BITT60 (SFS server) is the host name in this case.
4. Run **cicssdt:-> create**. The the server has been connected to.

Archive

We now show how to create a clustered SFS File (equivalent to a VSAM KSDS) named STAFF, as shown in Figure 15-3. For detailed information about the use of cicssdt and other utilities, refer to *CICS SFS Server and PPC Gateway Server – Advanced Administration V6.1*, SC34-6627.

```
C:\>cicssdt -s /.:/cics/sfs/BITT60

ERZ037068I/0500: CICSSDT talking to: '/.:/cics/sfs/BITT60'
ERZ037069I/0501: Version [2.1 : 12-07-1995]
ERZ037070I/0502: Contacting Server ...
ERZ037103I/0704: (Server OK)
cicssdt: -> create

[File  Name.....: STAFF

[SFS Server Volume Name .....: sfs_SBITT60
[File Organisation [E/R/B] ..: B[treeClustered]
                    (Taking default: BtreeClustered)
[Field 01: Name .....: ID
[Field 01: Type .....: unsignedInt16
[Field 02: Name .....: NAME
[Field 02: Type .....: byteArray
[Field 02: Size .....: 9
[Field 03: Name .....: DEPT
[Field 03: Type .....: unsignedInt16
[Field 04: Name .....: JOB
[Field 04: Type .....: byteArray
[Field 04: Size .....: 5
[Field 05: Name .....: YEARS
[Field 05: Type .....: unsignedInt16
[Field 06: Name .....: SALARY
[Field 06: Type .....: decimal
[Field 07: Name .....: COMM
[Field 07: Type .....: decimal
[Field 08: Name .....:
[Maximum Number Of Records ..: SFS_NATURAL_RECORD_LIMIT
[Primary Index Name .....: PRIMARY_KEY
[Is Index Unique ? [Y]/N ....: Y
[Index Field 01: Field Name .: ID
[Index Field 01: Ordering ...: a[scending]
[Index Field 02: Field Name .:
[Number Of Pages To Allocate : 1
```

Figure 15-3 Creating the SFS file STAFF using the cicssdt command

Archived

5. Add a secondary / alternate index (any time after the creation of the SFS file), as shown in Example 15-1.

Example 15-1 Adding a secondary index name using the cicssdt command

```
cicssdt: -> addindex

[File  Name.................: STAFF
[Secondary Index Name........: NAME
[Is Index Unique ? [Y]/N ....: N
[Alternate Rec. Spec. ? Y/[N]: N
[Index Field 01: Field Name .: NAME
[Index Field 01: Ordering ...: a[scending]
[Index Field 02: Field Name .:
cicssdt: ->
```

6. There are alternate methods of creating SFS files. You may also use the sfsadmin utility in a bat file to create the SFS files.
7. Generating test data for the SFS file STAFF.

   For a limited number of records, you may use the **cicssdt** command to feed in test data interactively. But to generate a substantial number of records, it is worthwhile to use a simple program to read a flat file and load the SFS file. If you would like to offload mainframe VSAM data on to the SFS file, the CICS supplied transaction “CALF” could be used. This would, of course, require connection to CICS on the mainframe. The export/import facilities can also be used to utilize batch processing for the transfer of data, as shown in Figure 15-4.

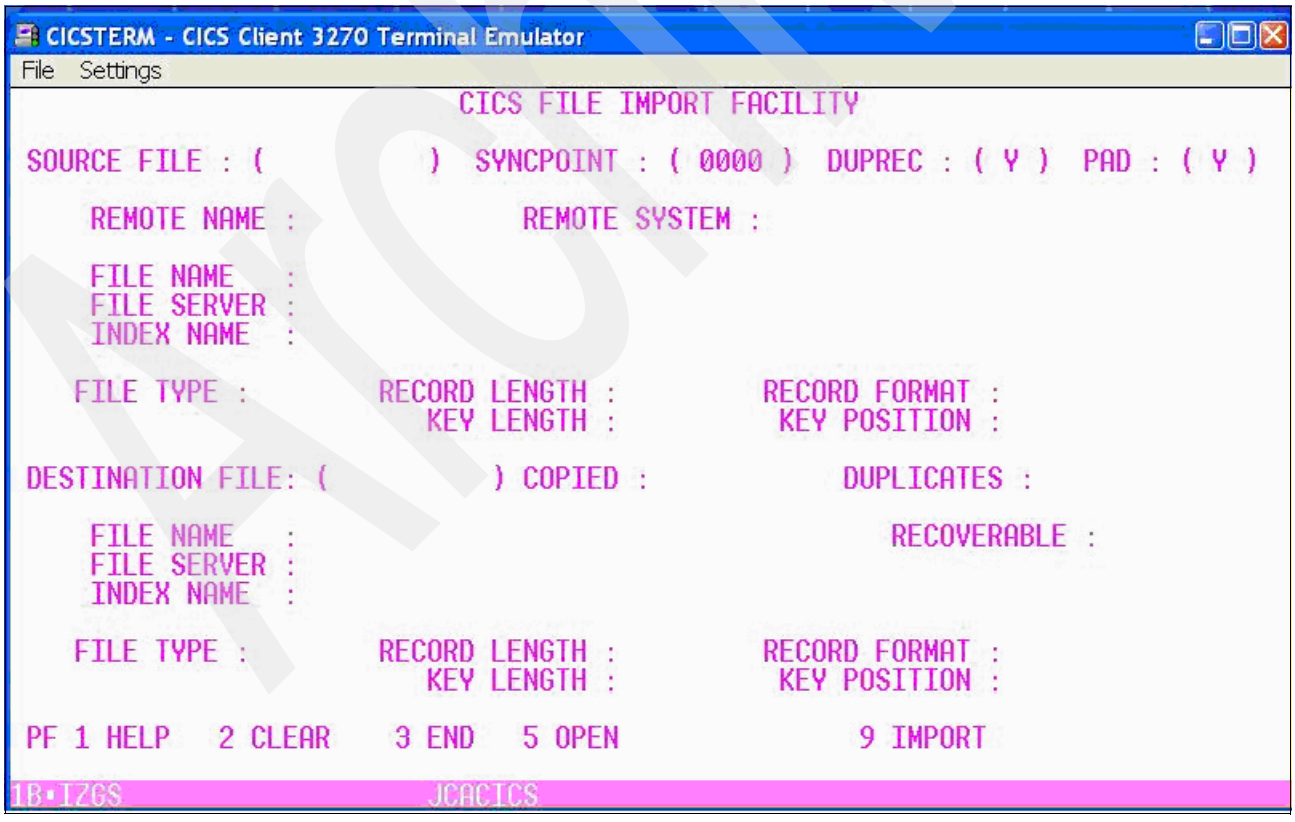


Figure 15-4 The IBM CICS supplied transaction CALF

8. Displaying information about the SFS file STAFF.

   The TXSeries Administration Console can be used to modify and display properties of the SFS files, as shown in Figure 15-5.

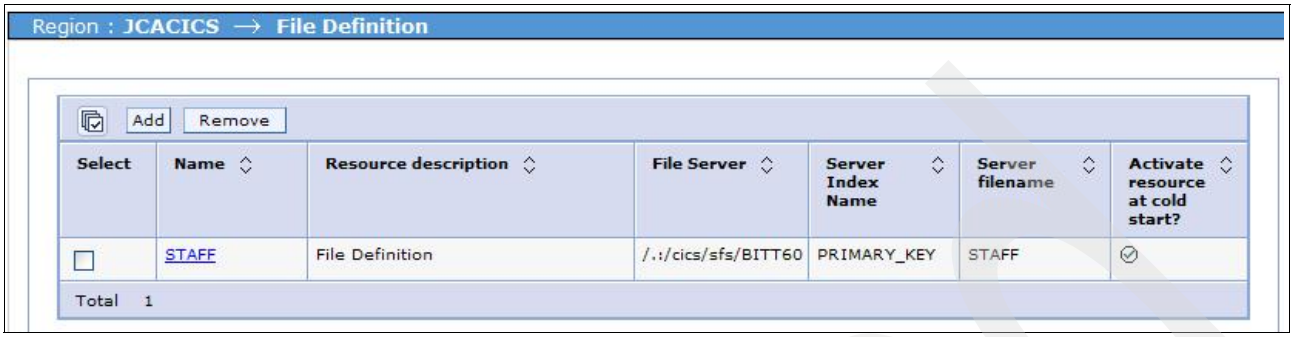


Figure 15-5 The TXSeries Administration Console: File Definition

Figure 15-6 shows the following properties set for STAFF:

- Server filename: STAFF
- File Server: /.:/cics/sfs/BITT60, where the host name is BITT60
- Server Index Name: PRIMARY_KEY, which is the name of the primary index

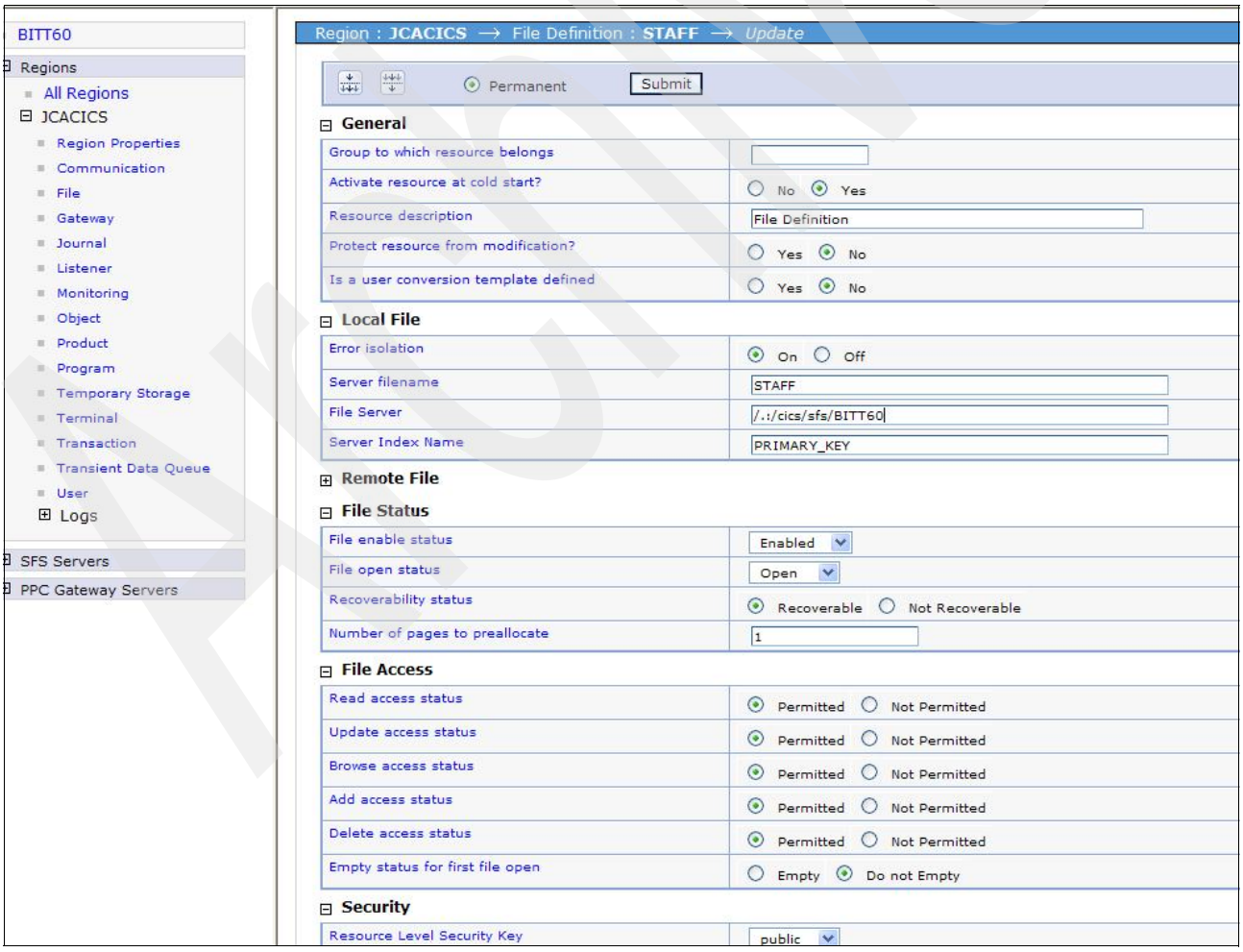


Figure 15-6 The TXSeries Administration Console: File Definition expanded

9. Use the cicssdt utility to get information about the SFS files, as shown in Example 15-2.

Example 15-2 Displaying information about SFS file STAFF

```
C:\>cicssdt -s /.:/cics/sfs/BITT60   BITT60 is the hostname in this case

ERZ037068I/0500: CICSSDT talking to: '/.:/cics/sfs/BITT60'
ERZ037069I/0501: Version [2.1 : 12-07-1995]
ERZ037070I/0502: Contacting Server ...
ERZ037103I/0704: (Server OK)

cicssdt: -> info STAFF

[Information For File: STAFF                                                  ]
------------------------------------------------------------------------------
[File Organisation          ] btreeClustered (KSDS)
[Primary Index Name         ] PRIMARY_KEY (Unique index)
[Primary Index Field(s)     ] ID
[Secondary Index Names      ] NAME
                              [Index Field(s)] NAME
[Number Of Records In File  ] 36
[Number Of Fields Per Record] 7
[Field 001: ID                          ][unsignedInt16      ][Size:      2]
[Field 002: NAME                        ][byteArray          ][Size:      9]
[Field 003: DEPT                        ][unsignedInt16      ][Size:      2]
[Field 004: JOB                         ][byteArray          ][Size:      5]
[Field 005: YEARS                       ][unsignedInt16      ][Size:      2]
[Field 006: SALARY                      ][decimal            ][Size:     18]
[Field 007: COMM                        ][decimal            ][Size:     18]
------------------------------------------------------------------------------
[No OFD information found for this file.                                      ]
------------------------------------------------------------------------------
```

10. Use the CICS Transaction CEMT to display or modify the SFS file status CEMT I FILE (STAFF), as shown in Figure 15-7.

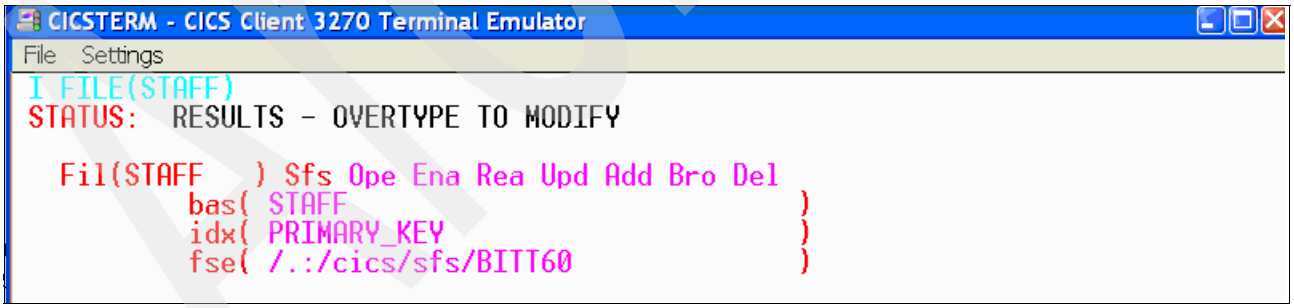


Figure 15-7 CICS terminal

15.1.2 Modifying COBOL/CICS programs and BMS maps

We shall, for the sake of continuity and simplicity, build on the existing COBOL/CICS programs and modify them to handle local SFS files. We also modify an existing Map for the purpose of accessing SFS data. Figure 15-8 shows the structure of the client server application. The server program COBSERV2 can reside on the mainframe and use VSAM or run on Windows using SFS without changing the program code. We modify the COBOL programs and the BMS Map with WebSphere Developer for System z (WD/z) and deploy them to TXSeries. We then debug the programs in the conventional form with CEDF as well as with the debugger in WD/z, as shown in Figure 15-8.

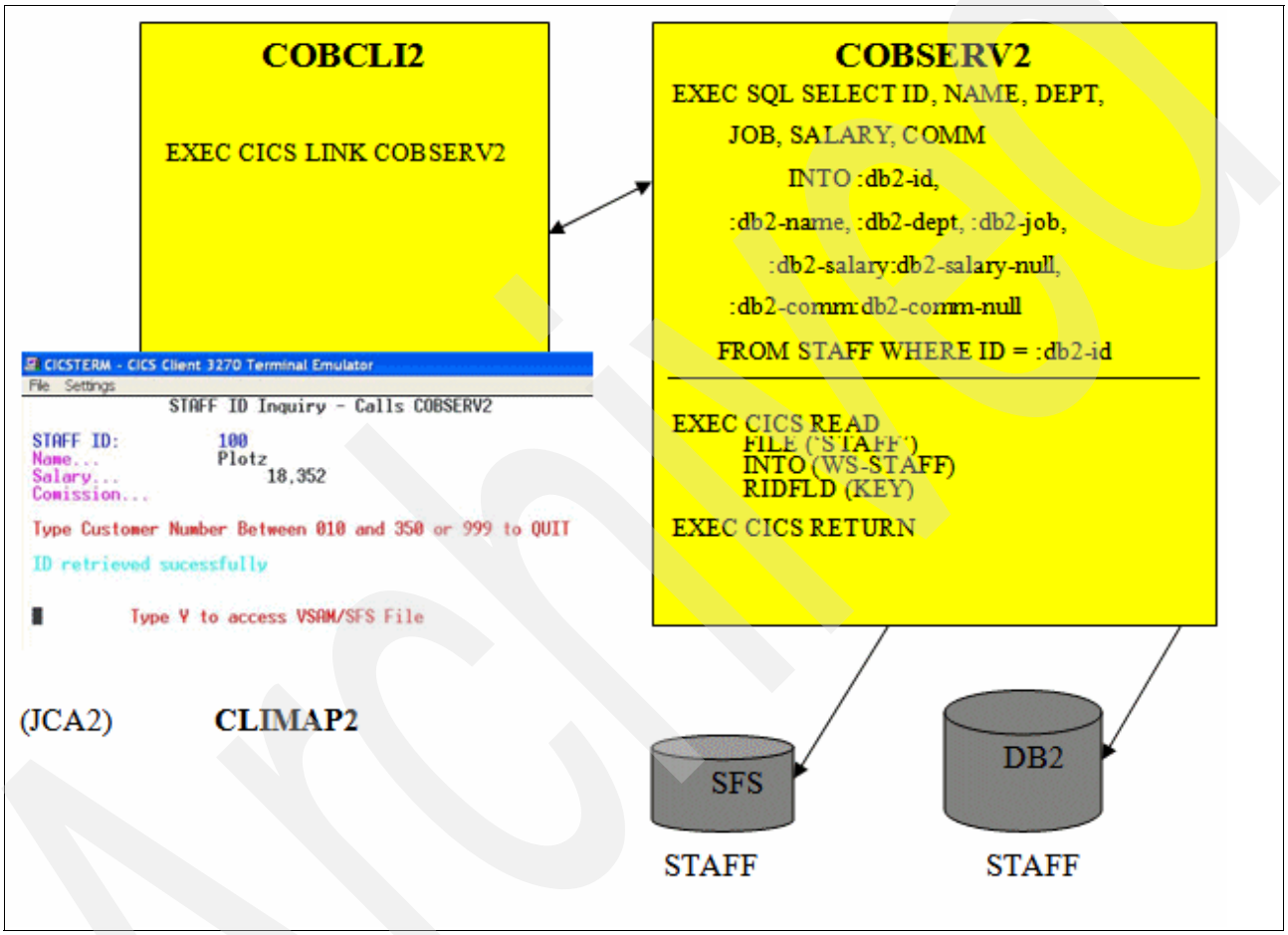


Figure 15-8 Scenario

Archive

Do these steps:

1. As mentioned earlier, we make a copy of CLIMAP.bms and rename it to CLIMAP2.bms. We import CLIMAP2.bms and use the BMS Map Editor in WD/z to modify the map. We create an input field to enter “Y” or “y” to access the SFS file STAFF. If the field is empty or does not contain a “Y” or a “y”, then the DB2 database will be accessed. The Design mode makes it quite easy to add or modify fields and their attributes. The modification can be immediately checked in the Preview mode. In the Source mode, the code may be modified and field names added or changed, as shown in Figure 15-9 and Figure 15-10 on page 533.

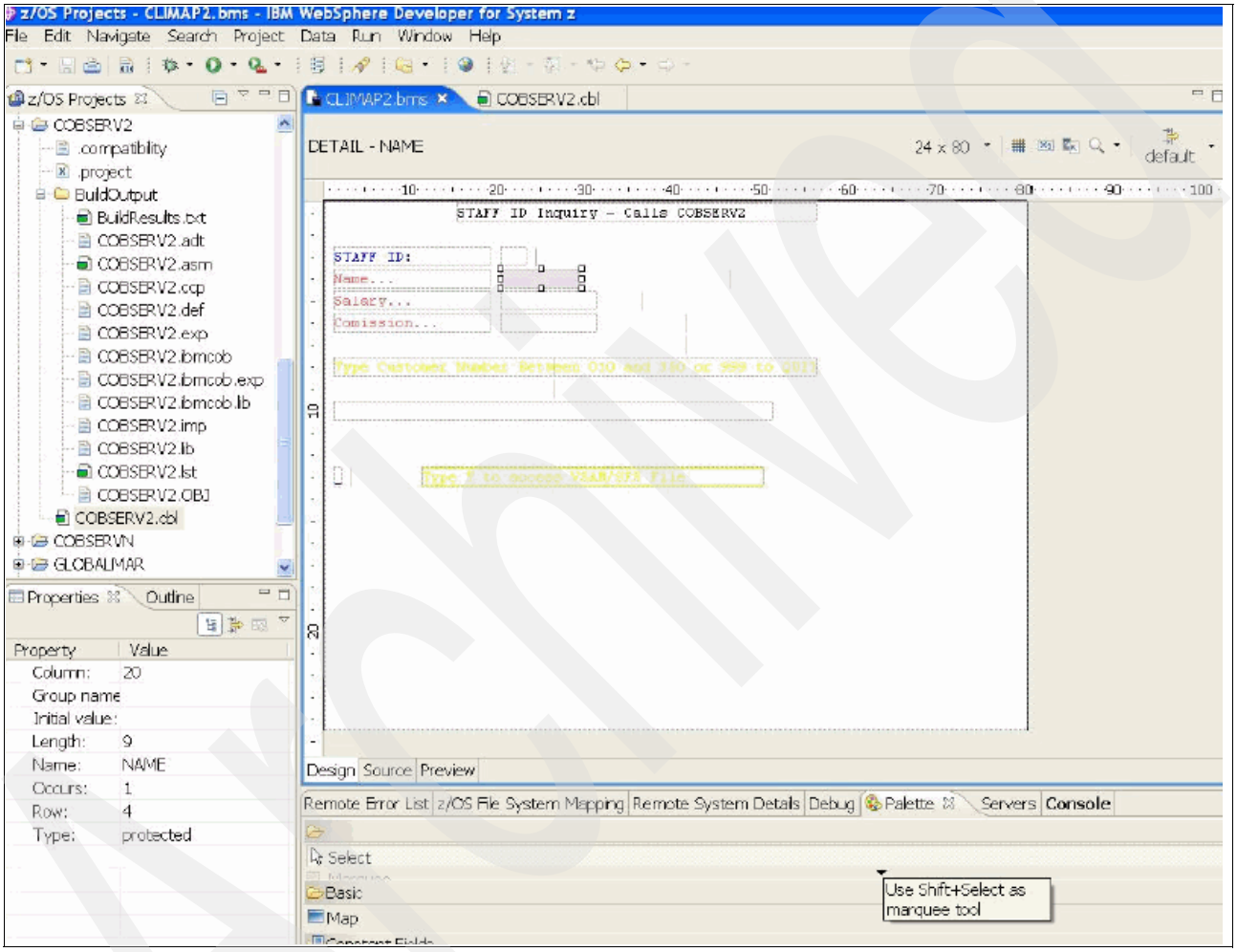


Figure 15-9 The BMS Map Editor in Design mode

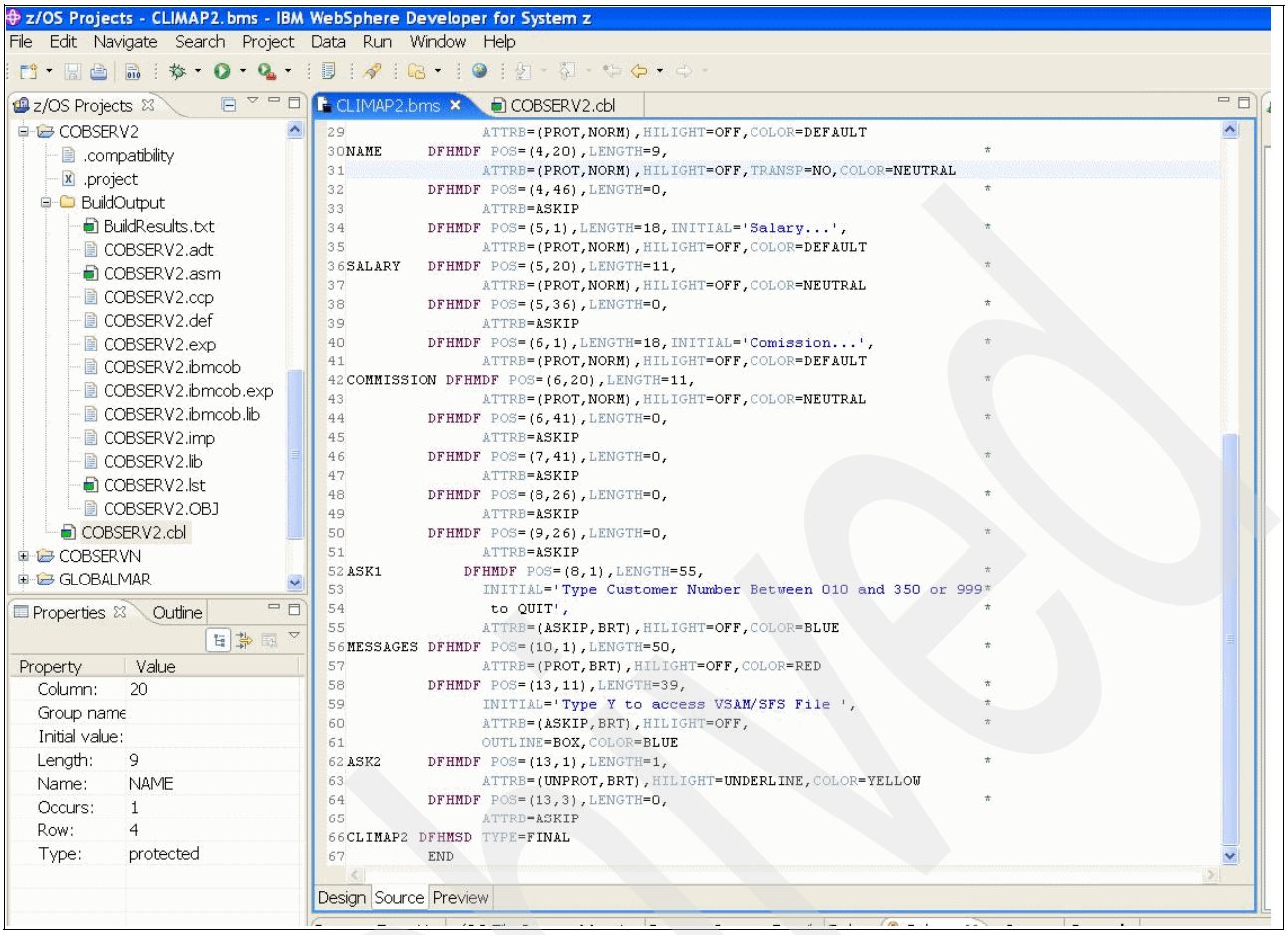


Figure 15-10 The BMS Map Editor in Source mode

The extract from the source shown in Example 15-3 shows the addition/modification done to the original source code. The field “ASK2” has been added as an Input field for entering “Y” or “y”.

The following text has also been added, `'Type Y to access VSAM/SFS File`**”,** as shown in Example 15-3.

Example 15-3 Source

```
MESSAGES DFHMDF POS=(10,1),LENGTH=50,                                  *
               ATTRB=(PROT,BRT),HILIGHT=OFF,COLOR=RED
         DFHMDF POS=(13,11),LENGTH=39,                                 *
               INITIAL='Type Y to access VSAM/SFS File ',              *
               ATTRB=(ASKIP,BRT),HILIGHT=OFF,                          *
               OUTLINE=BOX,COLOR=BLUE
ASK2     DFHMDF POS=(13,1),LENGTH=1,                                   *
               ATTRB=(UNPROT,BRT),HILIGHT=UNDERLINE,COLOR=YELLOW
         DFHMDF POS=(13,3),LENGTH=0,                                   *
               ATTRB=ASKIP
CLIMAP2 DFHMSD TYPE=FINAL
         END
```

Archived

2. We now deploy the BMS Map to TXSeries.

   a. After saving the changes, we export CLIMAP2.bms to a work directory and execute the command **`cicsmap CLIMAP2.bms`**.

   b. We then move the map created to the TXSeries location using the command **`xcopy CLIMAP2M.map C:\opt\cics\maps\EN_US /y`**.

   c. We use WD/z to modify the Client COBOL program COBCLI2 and the Server COBOL program COBSERV2 to access VSAM files (or SFS on Windows). Figure 15-11 shows an extract from the program COBCLI2 to show some of the changes done to accommodate the changes in the Map CLIMAP2. The entire source code for the programs, maps, and copybooks are included in the additional material for this book (refer to Appendix B, “Additional material” on page 645for information about obtaining this code).

```
 Line 102     Column 61    Insert
      ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
000079                MOVE 'ID must be 3 numeric digits' TO WS-MESSAGES
000080                MOVE 0 TO WS-ID
000081             END-IF
000082             IF  WS-ID > 350 OR  WS-ID < 10 then
000083                move 'ID must be between 010 and 350' to WS-MESSAGES
000084       *         MOVE 0 TO WS-ID
000085             END-IF
000086       * check if DB2 or VSAM /SFS is to be queried
000087             IF ASK2I = 'Y' OR ASK2I = 'y'
000088                MOVE 'Y' TO WS-VSAM
000089             ELSE
000090                MOVE ' ' TO WS-VSAM
000091             END-IF
000092             IF  WS-ID = 99  OR  WS-ID = 999 then
000093                MOVE 'NO ' TO WS-SEND-MAP
000094                MOVE 0 TO WS-ID
000095                MOVE 'Program COBCLI2 ended.         ' TO WS-MESSAGES
000096                MOVE SPACES to ASK1O
000097                END-IF .
000098       *     ---------------------------------------------------------------
000099        0300-INVOKE-CICS-SERVER.
000100             INITIALIZE WORK-COMMAREA
000101             MOVE  IDI  to WORK-ID  IN WORK-COMMAREA
000102             MOVE WS-VSAM TO WORK-VSAM
000103
000104             EXEC CICS LINK PROGRAM  ('COBSERV2')
000105                  COMMAREA (WORK-COMMAREA)
000106             END-EXEC
000107       *
000108             IF WORK-RETCODE = 0
000109               MOVE  WS-IDNUM                      to  IDO
000110               MOVE  WORK-NAME   IN WORK-COMMAREA  to  NAMEO
000111               MOVE  WORK-SALARY  IN WORK-COMMAREA  to  WS-SALARYZ
```

Figure 15-11 Extract from COBCLI2

Figure 15-12 shows an extract from the program COBSERV2 that shows how the EXEC CICS command is used to access VSAM .

```
Line 136     Column 50    Insert
      ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
000104              IF db2-comm-null is NOT NEGATIVE
000105                 MOVE db2-comm                    TO WORK-COMM
000106              END-IF
000107
000108            ELSE
000109              MOVE SQLCODE to WORK-RETCODE
000110            END-IF.
000111       * ------------------------------------------------------------- *
000112       * Get Data from local SFS (VSAM) File                           *
000113        2000-GET-CUSTOMER-DATA.
000114       * ------------------------------------------------------------- *
000115            MOVE WORK-ID  TO T-KEY.
000116            MOVE 56       TO T-LENGTH.
000117            MOVE 2        TO T-KEYLENGTH.
000118
000119            EXEC CICS HANDLE CONDITION NOTFND (NOTFND-ERROR)
000120            END-EXEC.
000121
000122            EXEC CICS READ
000123                 INTO (VS-STAFF)
000124                 FILE ('STAFF')
000125                 RIDFLD (T-KEY)
000126            END-EXEC.
000127
000128            MOVE VS-NAME TO WORK-NAME.
000129            MOVE VS-SALARY TO WORK-SALARY.
000130            MOVE VS-COMM   TO WORK-COMM.
000131
000132       * TRAP ERRORS
000133        NOTFND-ERROR. MOVE 100 TO WORK-RETCODE
000134                      MOVE 'ID NOTFND ' TO WORK-VSAM-ERROR.
000135
000136                      EXEC CICS HANDLE CONDITION ERROR END-EXEC.
```

Figure 15-12 Extract from COBSERV2

Archived

3. Install the generated programs and map them into TXSeries.

   After having created the modified or new TXSeries components, we need to define the TXSeries resources using the TXSeries Administration Console. We need to add the two COBOL programs COBCLI2 and COBSERV2 and the Map CLIMAP2 as TXSeries resources, as shown in Figure 15-13.

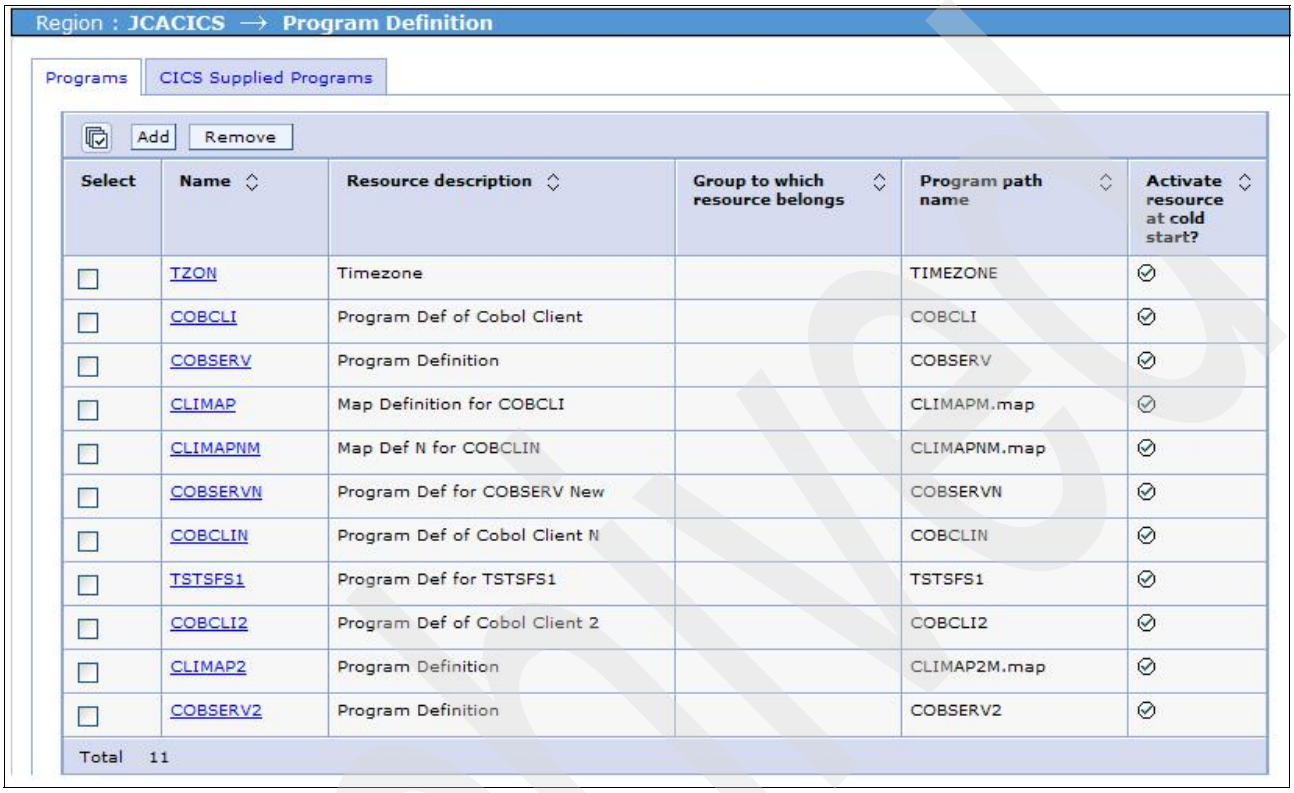


Figure 15-13 Adding or modifying the program and map definitions

Archive

We also need to add the new transaction (JCA2) that will invoke the COBOL client program COBCLI2, as shown in Figure 15-14.

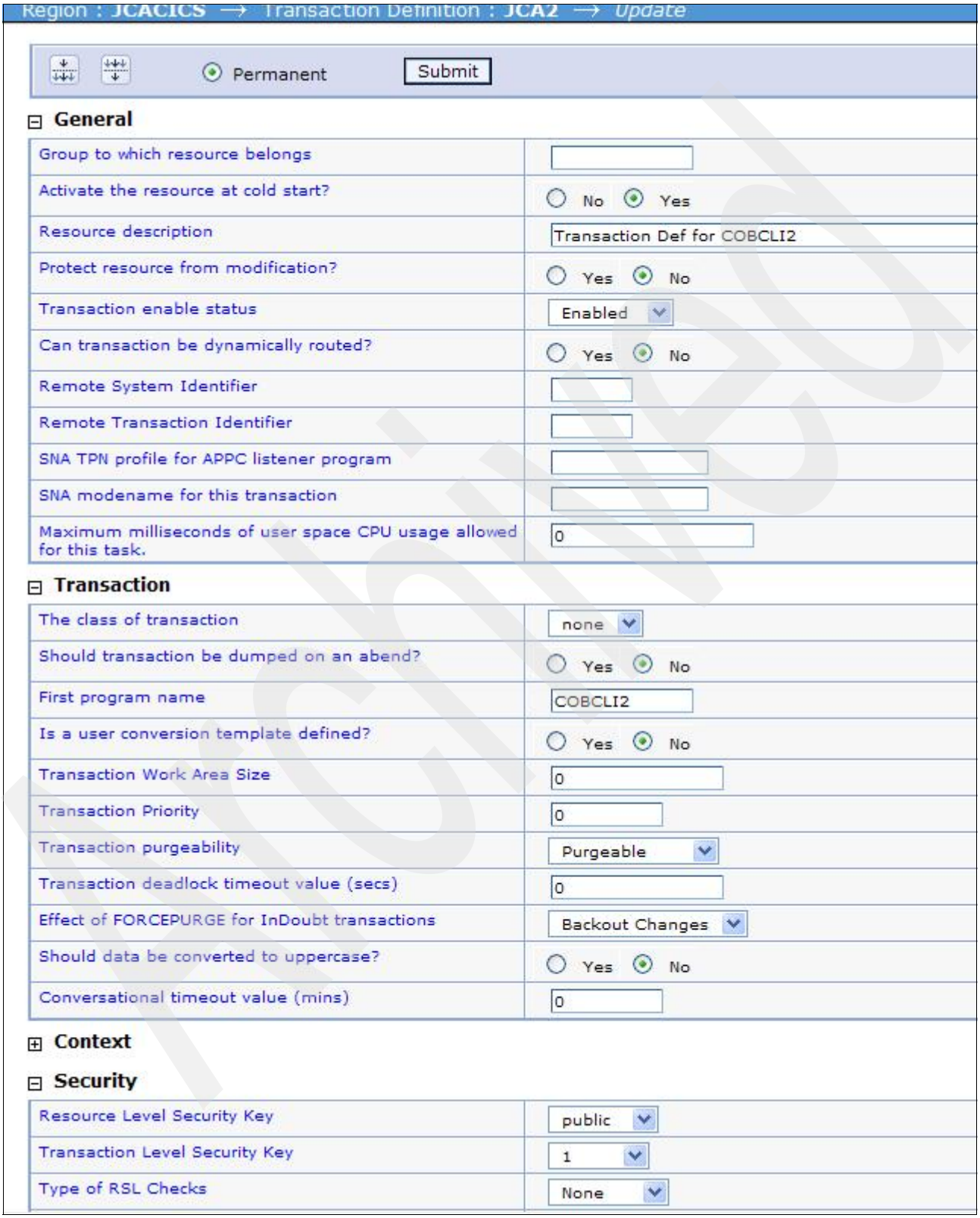


Figure 15-14 The TXSeries Administration Console

4. Cold start the CICS region (JCACICS). Before we can test the deployed resources, we need to stop and start the region JCACICS (the region that we have been using for our examples). A cold start is necessary, as we have added new programs and maps. Using a CTG CICS Terminal or the local terminal, we can test our transaction by typing in JCA2. If everything has been correctly done, we should see the map and can access both DB2 as well as the SFS file.

15.2 Debugging COBOL/CICS with VSAM/SFS

We shall debug our Client/Server application both in the conventional CEDF mode as well as using the integrated debugger in WD/z to debug the source code.

Figure 15-15 through Figure 15-26 on page 543 show a typical CEDF session.

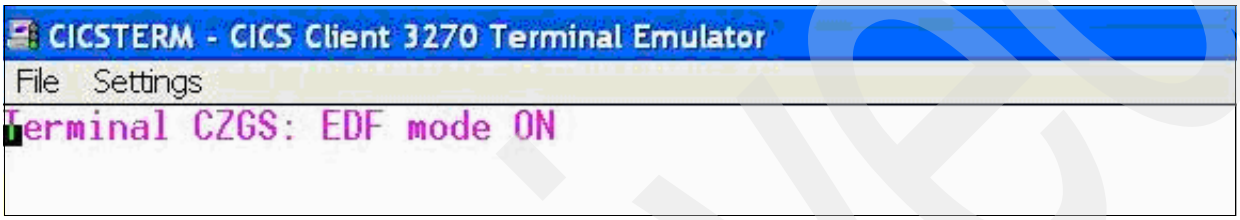


Figure 15-15 Starting a CEDF session

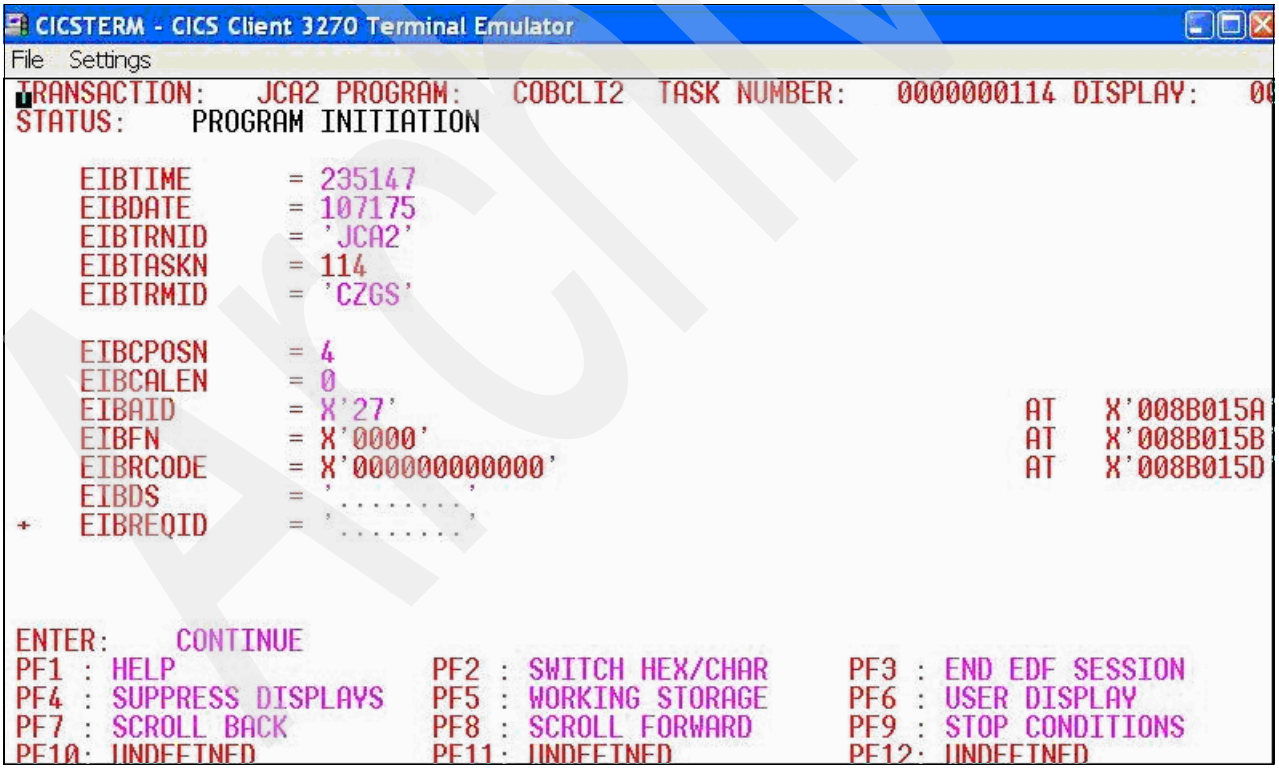


Figure 15-16 Transaction JCA2 has been activated

Archived

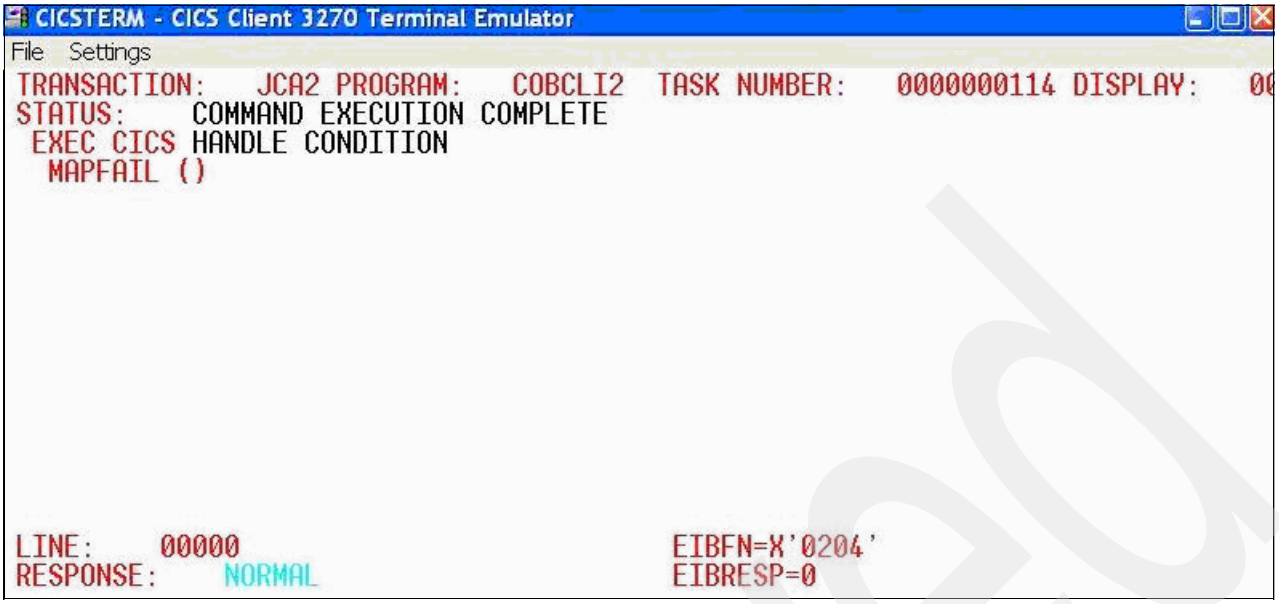


Figure 15-17 Testing handle condition

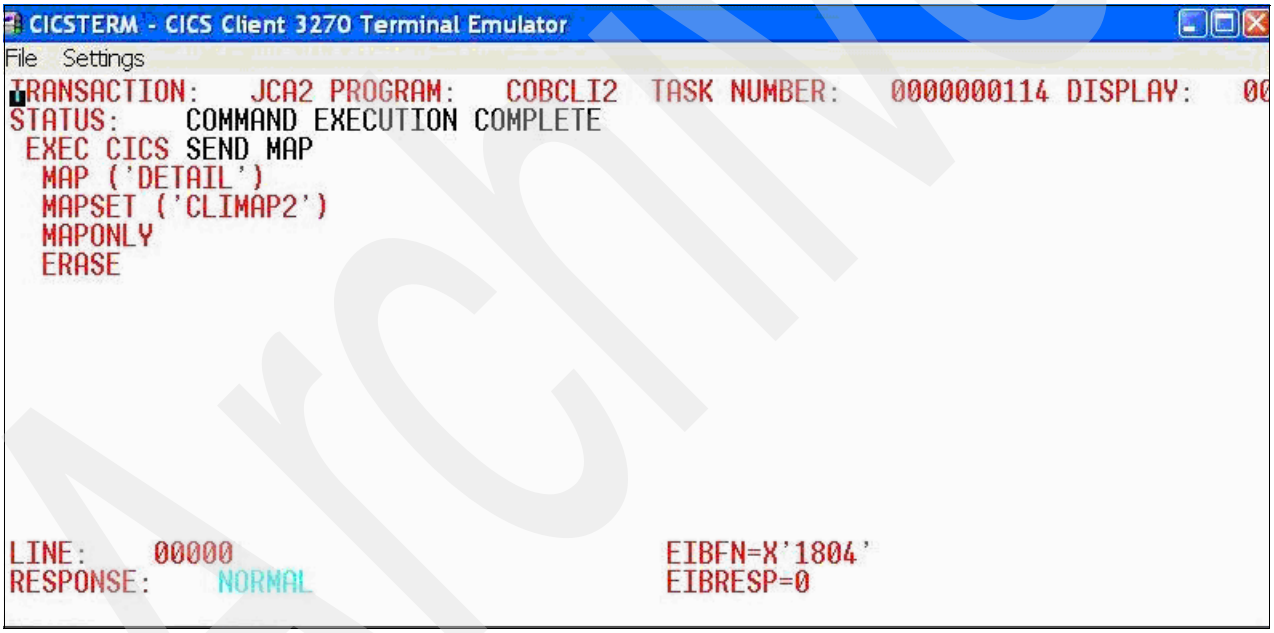


Figure 15-18 Preparing to send map

Archived

We have entered a “Y” so that the SFS file is accessed for the ID number 070, as shown in Figure 15-19.

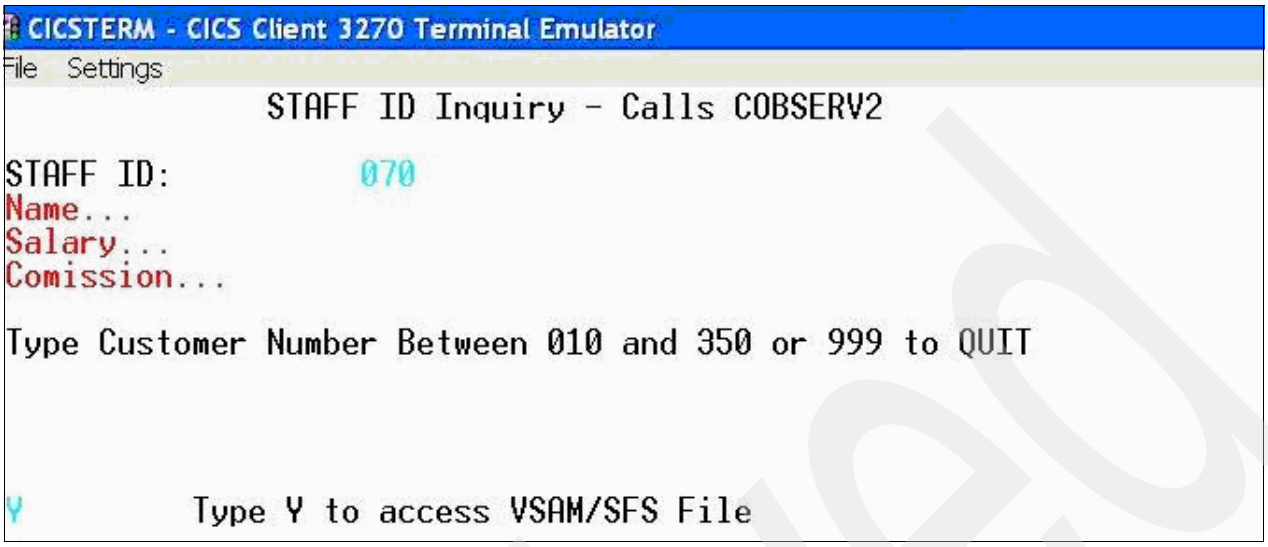


Figure 15-19 Map has been sent and data has been typed in

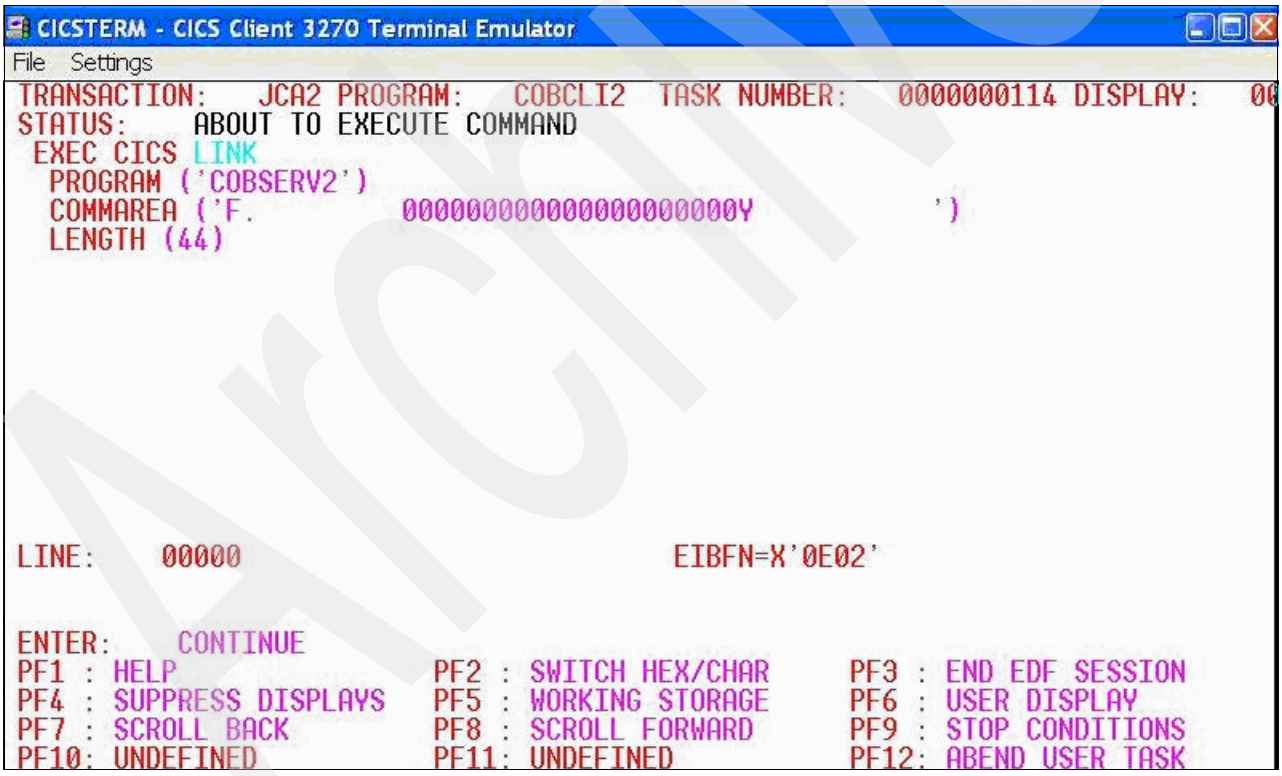


Figure 15-20 Client program COBCLI2 passes data on to the server program

Archived

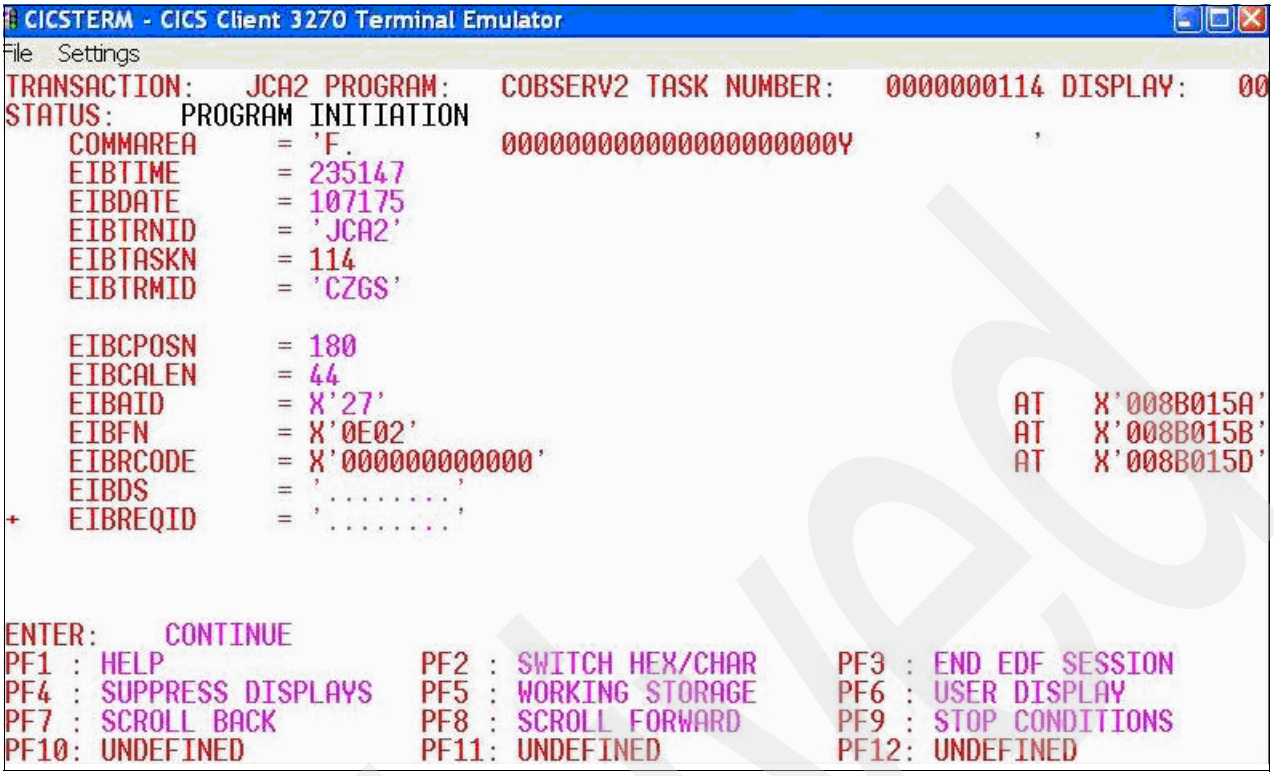


Figure 15-21 Commarea is being passed to the server program COBSERV2

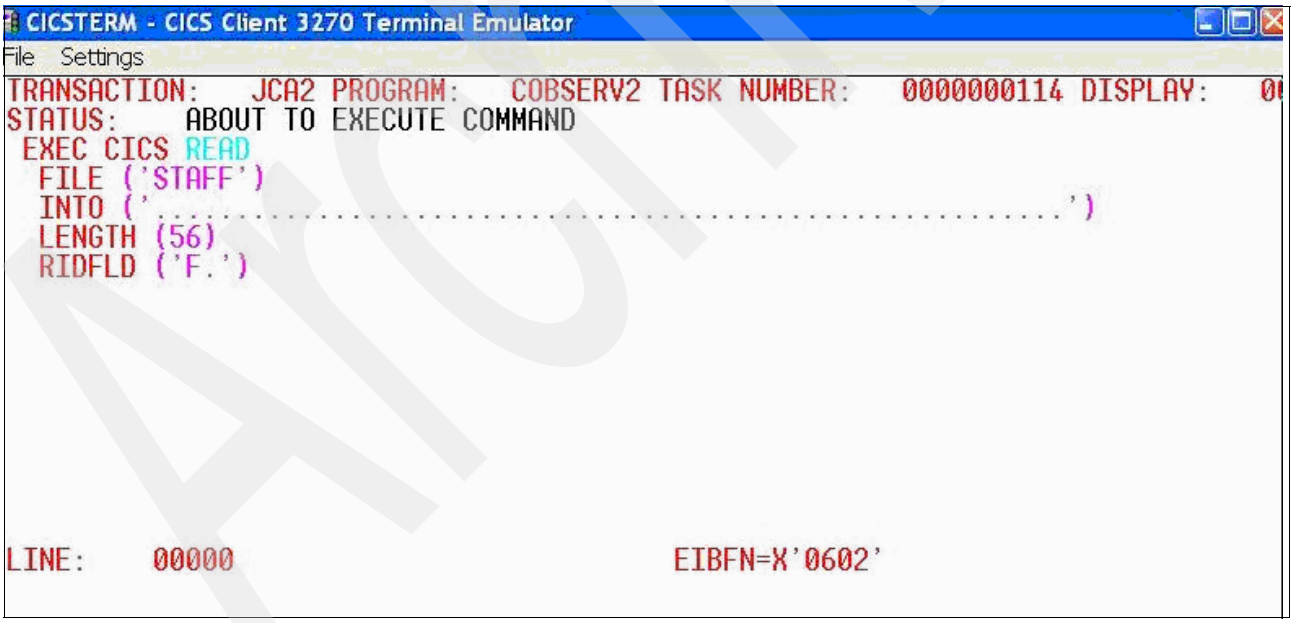


Figure 15-22 Read

Archived

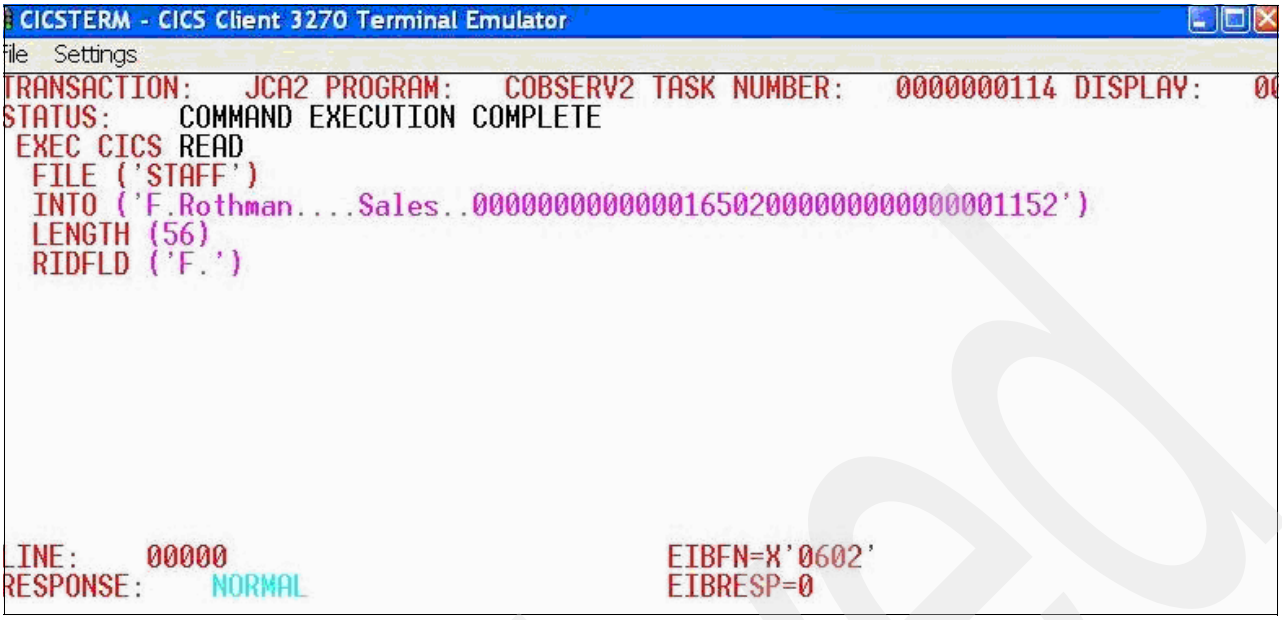


Figure 15-23 EXEC CICS READ has been successful and data received

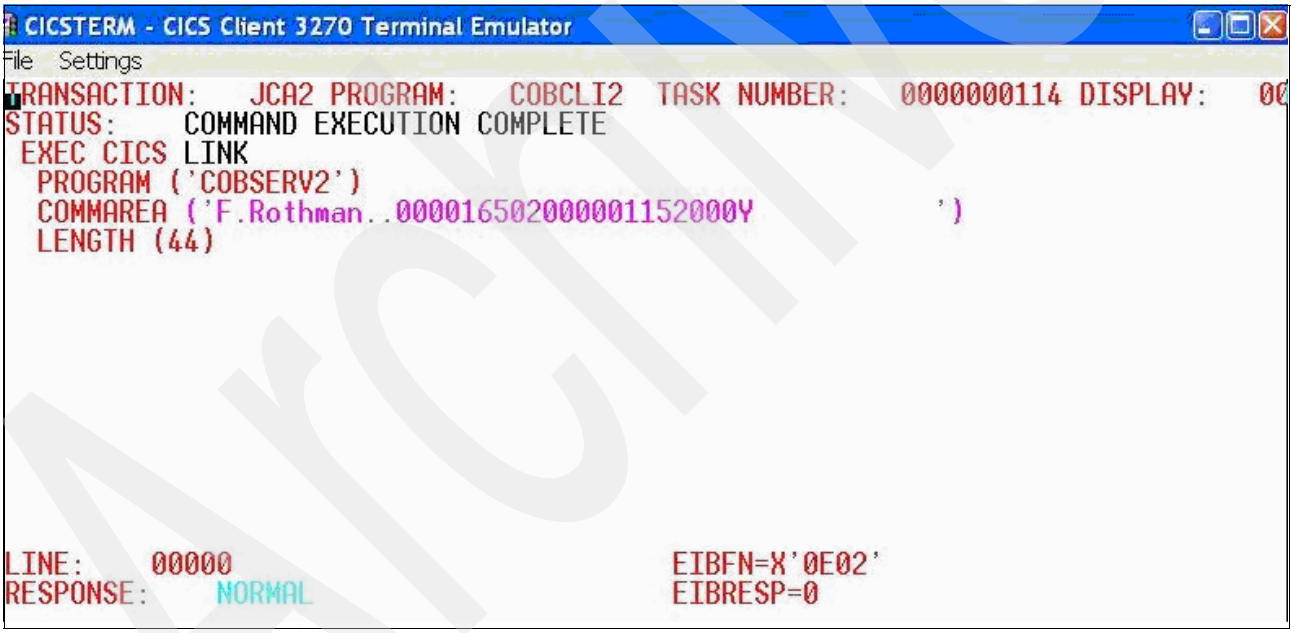


Figure 15-24 Control is being passed back to the client program

Archived

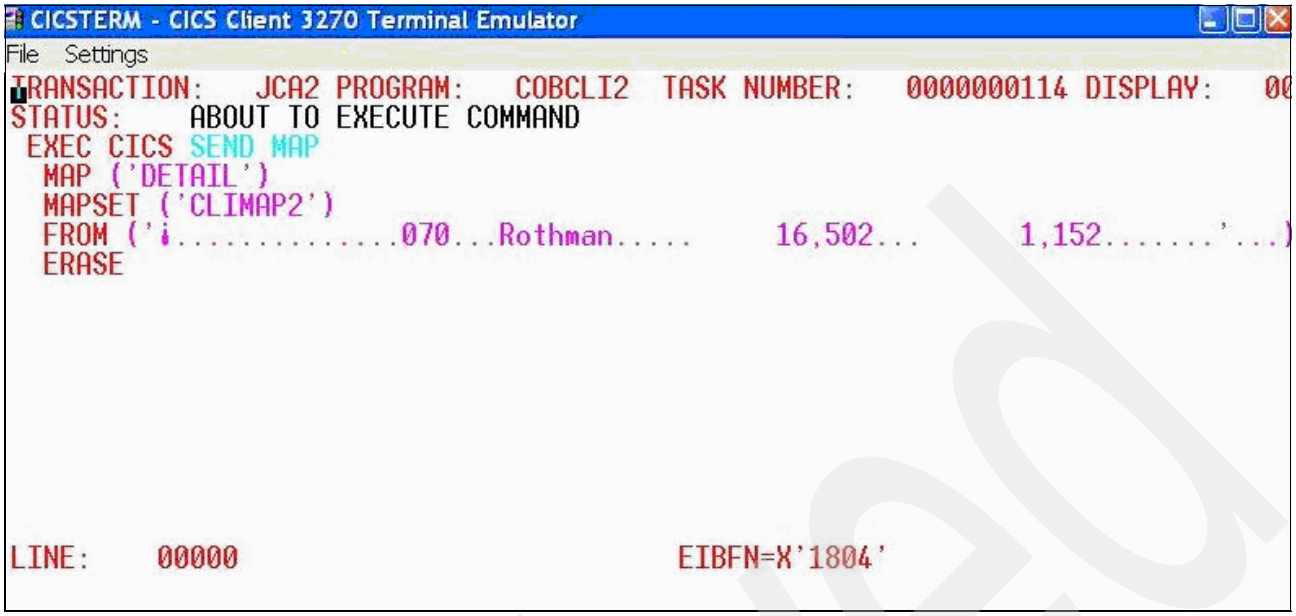


Figure 15-25 Client program preparing to send retrieved data to the Map

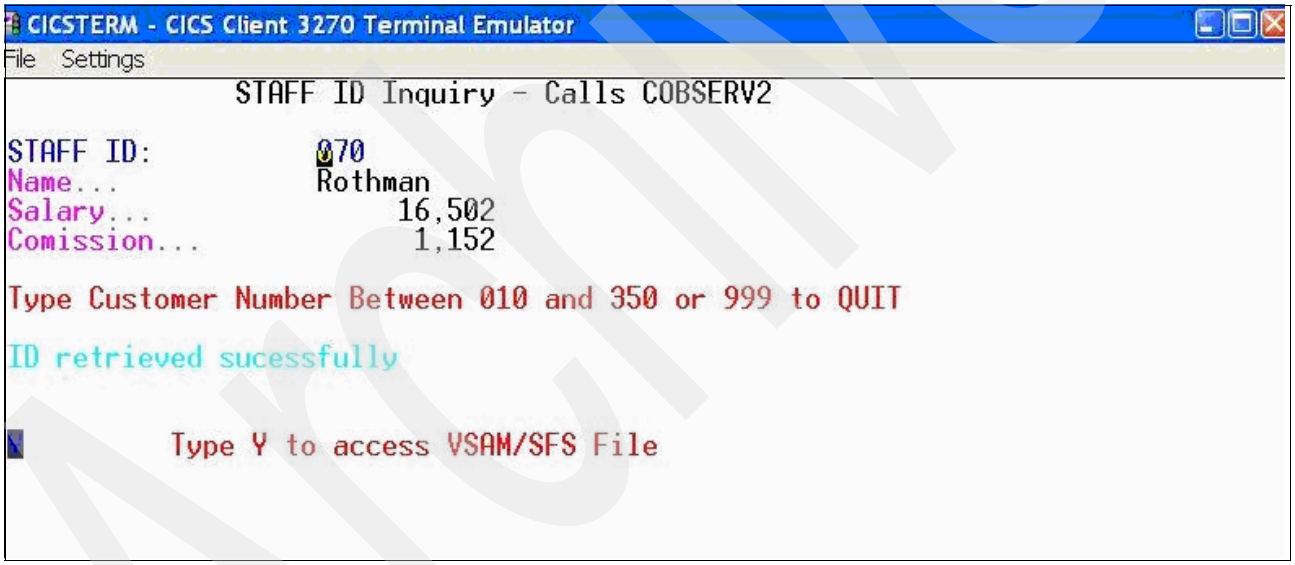


Figure 15-26 Data shown on the map has been retrieved from the SFS file STAFF

Archived

We now use the integrated debugger in WD/z to debug the source code. The following series of screen captures show the different steps that have to be followed to set up the debug environment to connect WD/z and TXSeries. To avoid repetition, we have left out some of the initial steps, such as opening the Debug Perspective, selecting the menu for launching a new configuration, and attaching to a running process. We concentrate on access to the VSAM / SFS file STAFF, as shown in Figure 15-27.

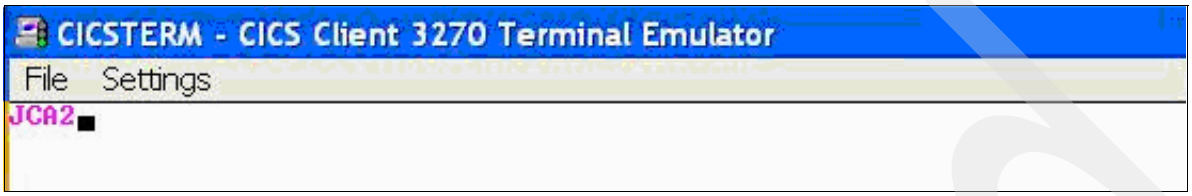


Figure 15-27 The transaction ID is JCA2

The Debug Engine message window appears, indicating that a debug a session has been started. We click the **OK** button to continue, as shown in Figure 15-28.

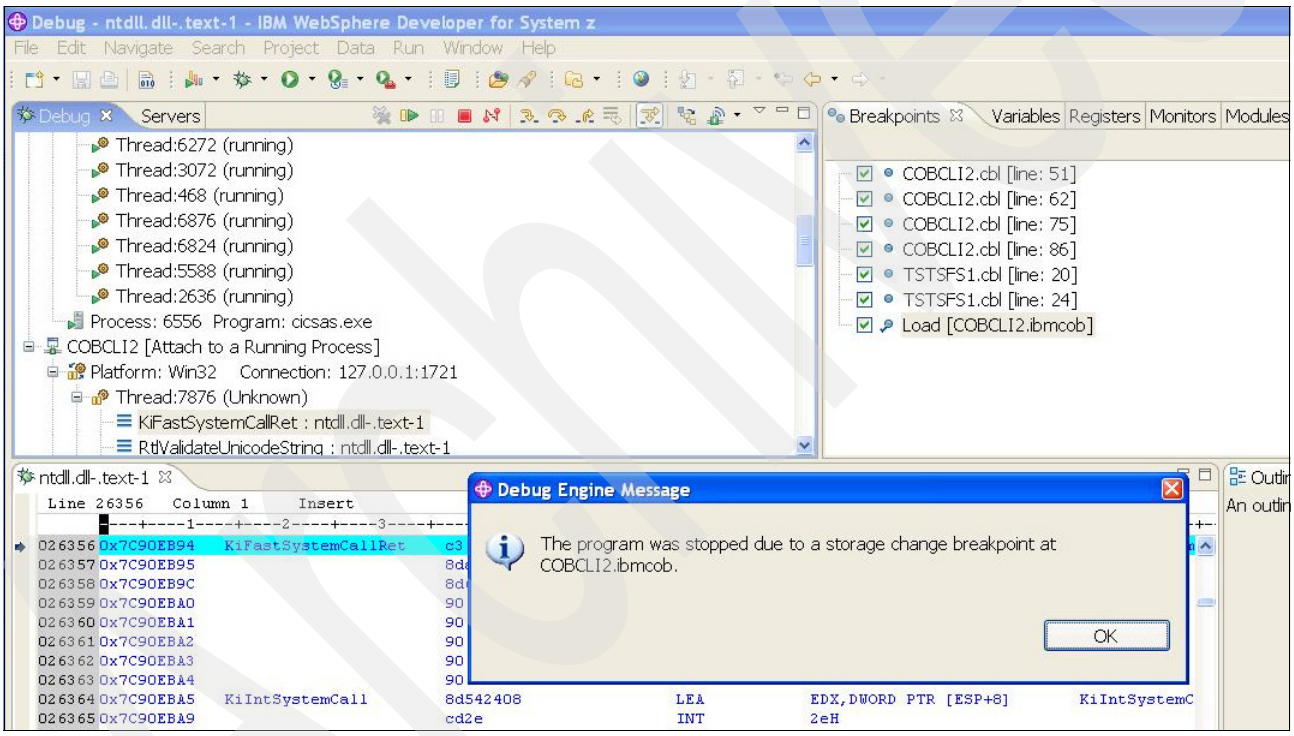


Figure 15-28 Debug Engine

In the Modules view, we expand the objects to get to the COBCLI2 source listing. Then we double-click **COBCLI2**. This opens the source listing for the program COBCLI2, as shown in Figure 15-29.

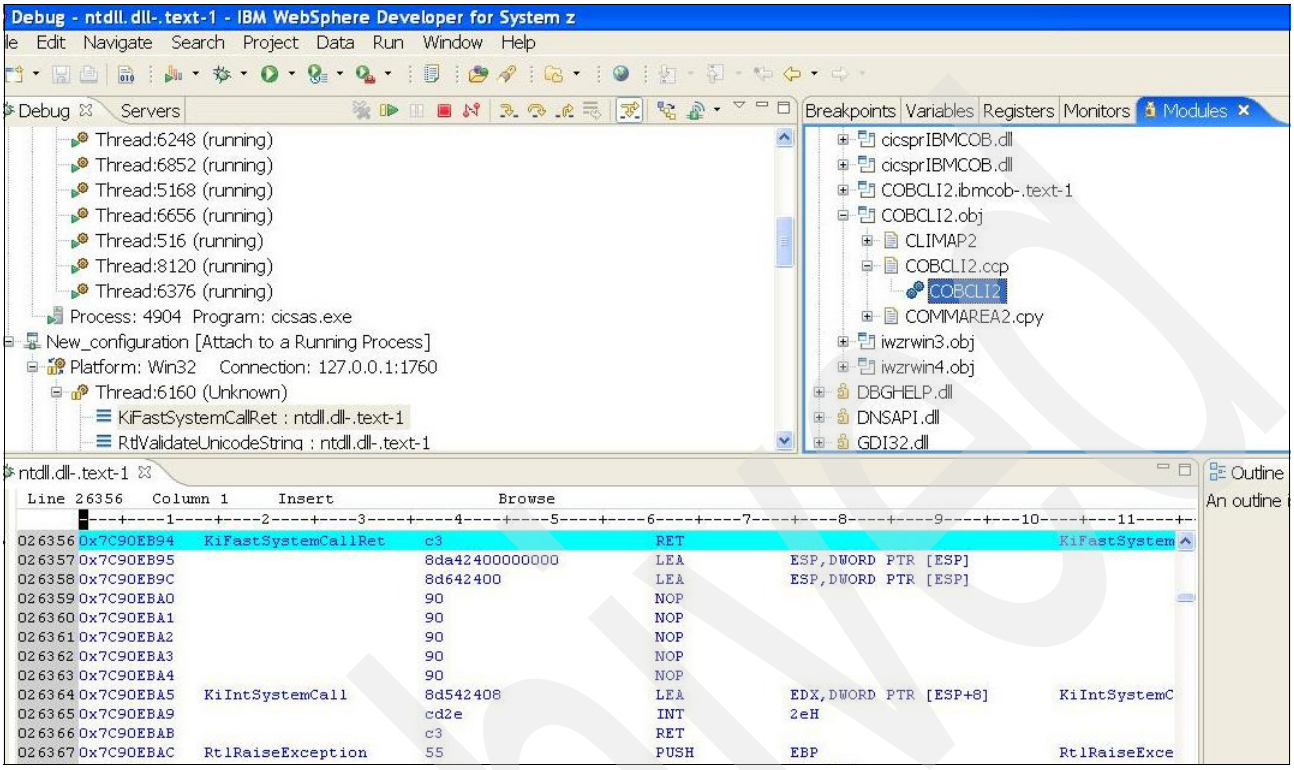


Figure 15-29 Modules view

Archive

The COBCLI2.ccp listing appears, as shown in Figure 15-30.

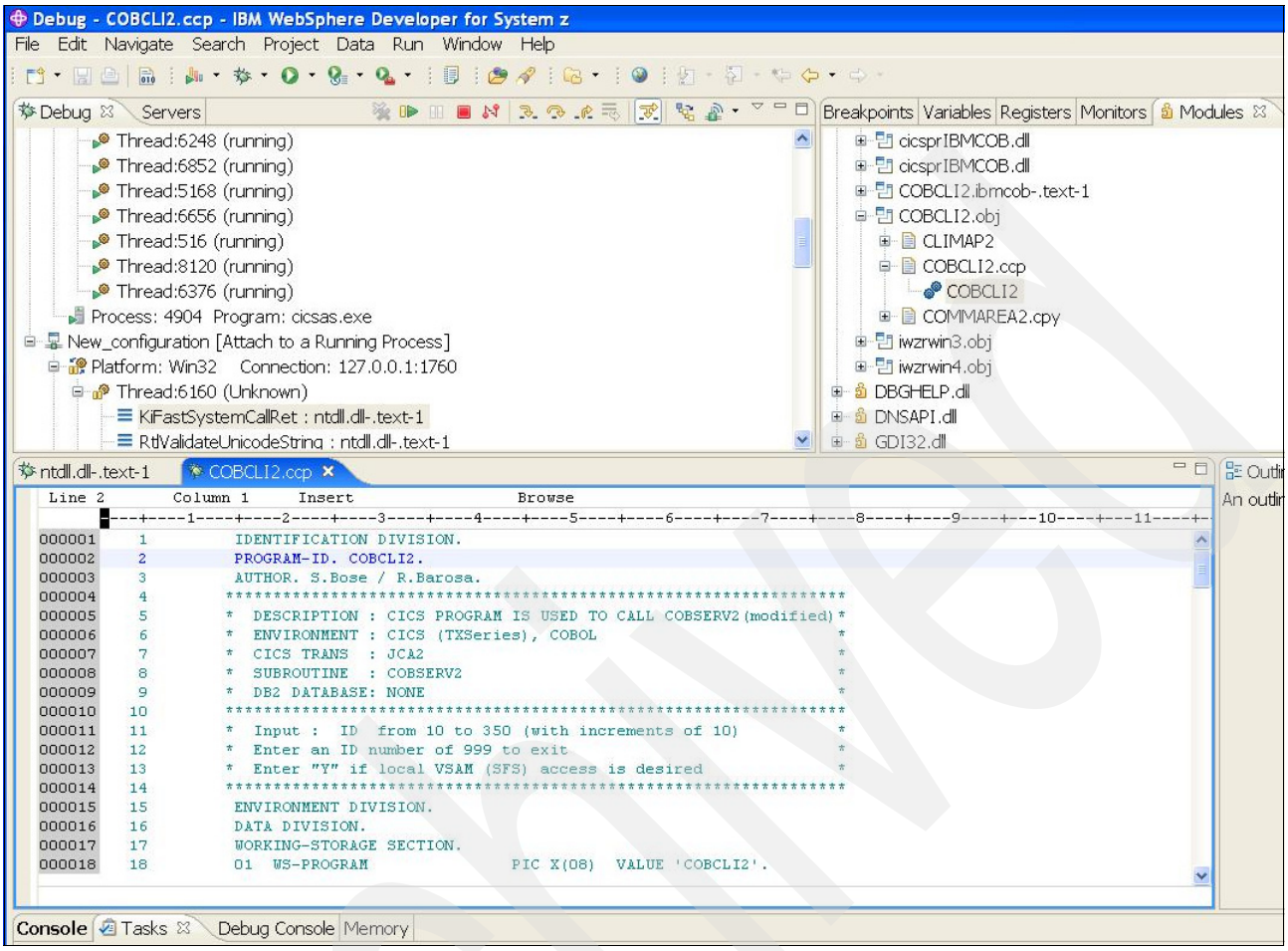


Figure 15-30 Listing

Archive

Here we select **Add a Load Breakpoint** and enter COBSERV2.ibmcob as the Library name and are about to click **Finish**. The file COBSERV2.ibmcob is the load module that we deploy to CICS and that represents the COBSERV2.cbl program. The breakpoint will force a stop of the CICS transaction JCA2 when it first enters the COBSERV2 program, as shown in Figure 15-31.

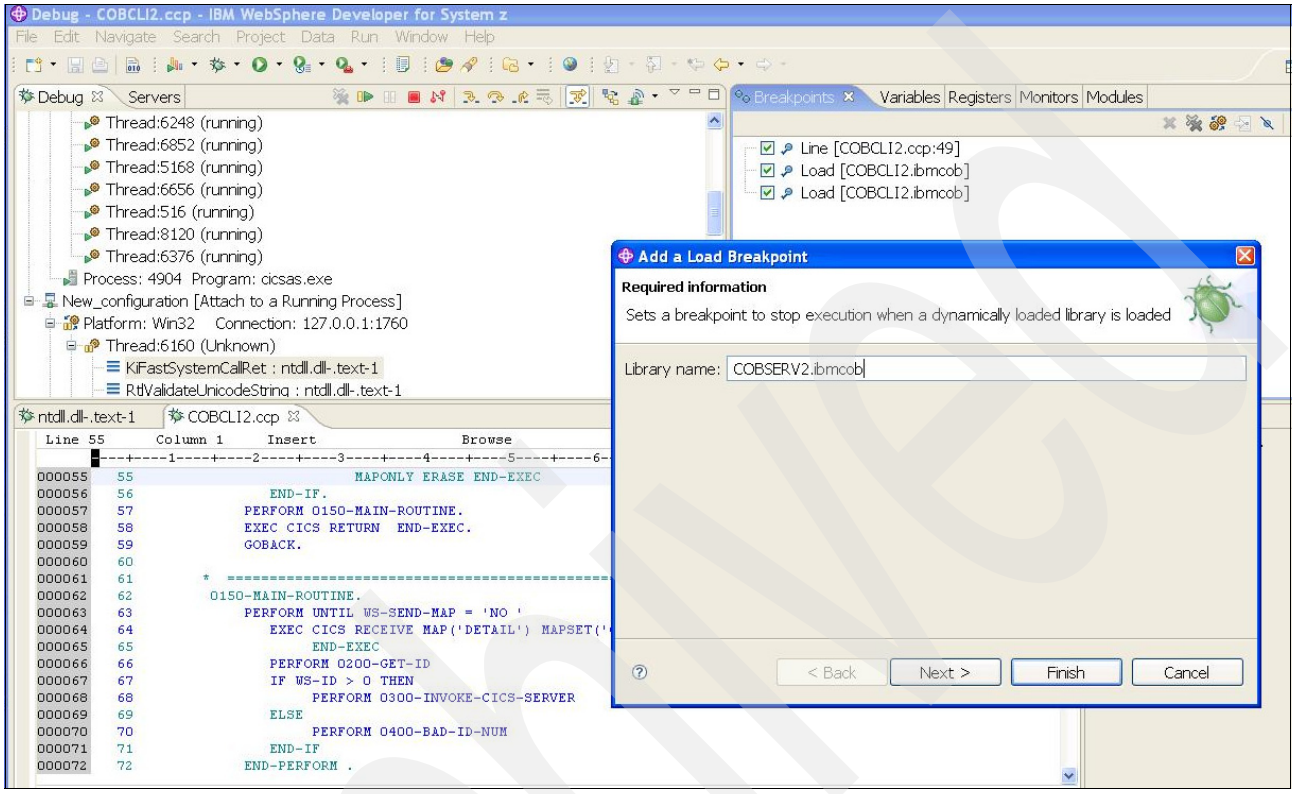


Figure 15-31 Breakpoint

Archived

We are still in the program COBCLI2 and have just passed the first breakpoint on line 49. This is just before the map is sent, as shown in Figure 15-32.

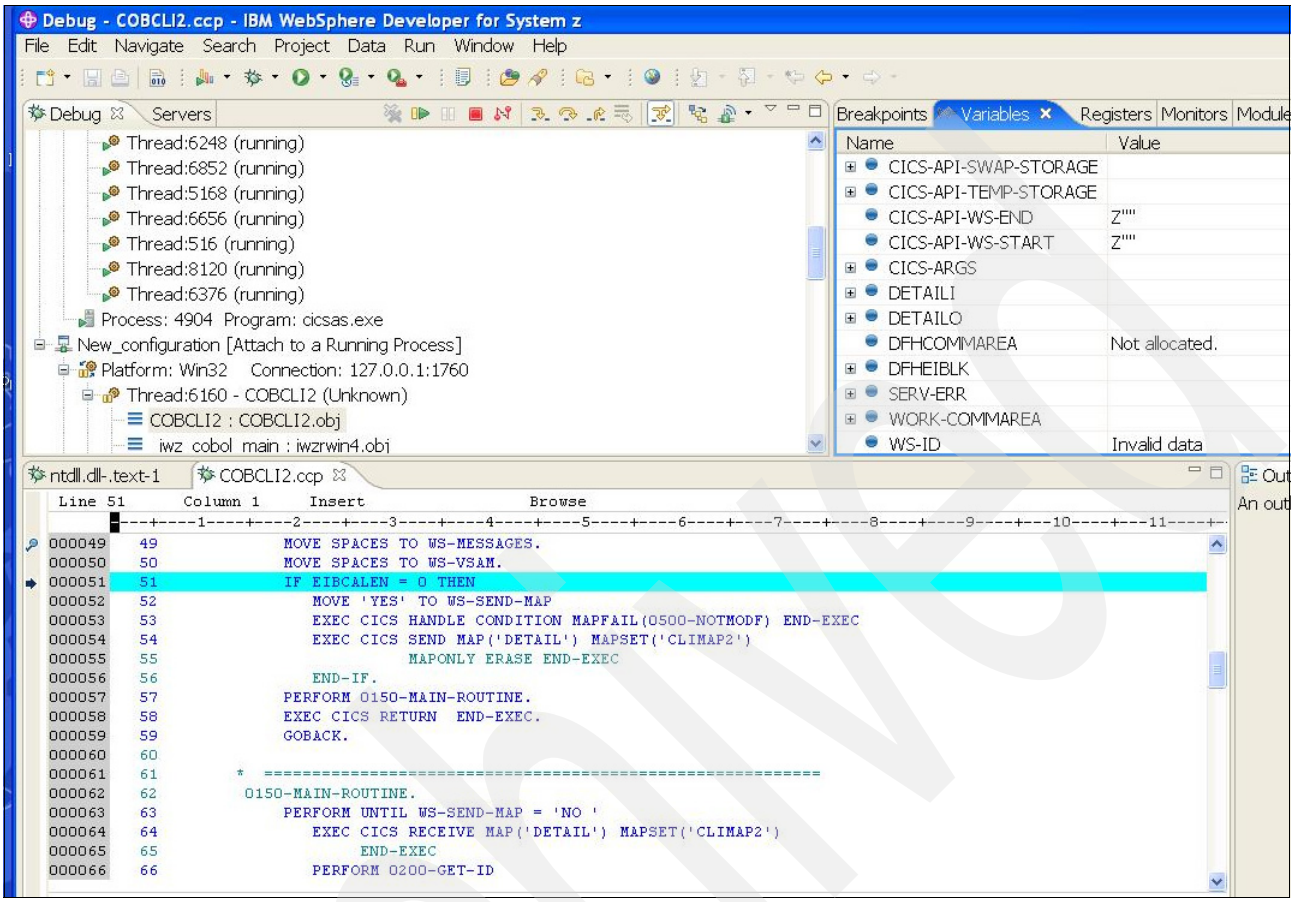


Figure 15-32 Map

Archive

The map has been sent and is awaiting input for accessing either the DB2 table STAFF or the SFS file STAFF with the ID as a Key, as shown in Figure 15-33.

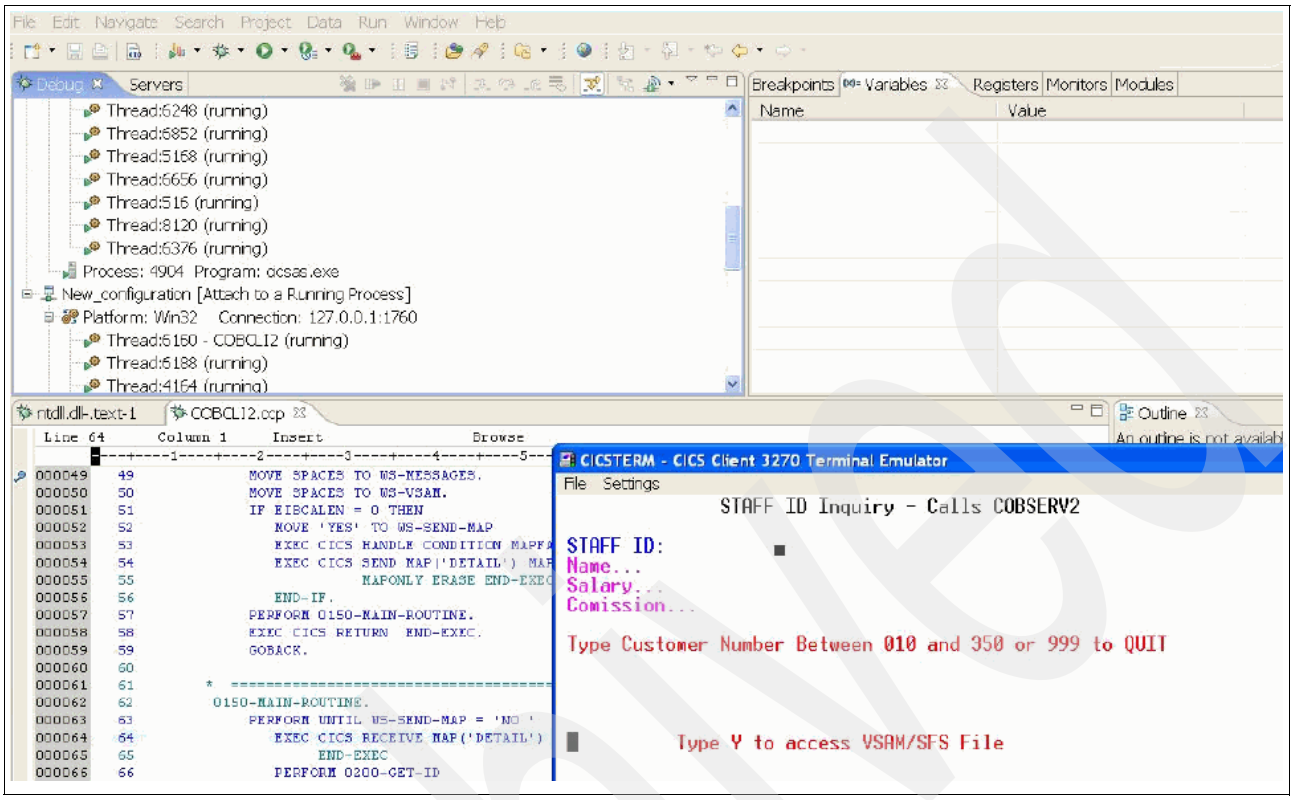


Figure 15-33 Map

Archived

We type in 070 as the ID and “Y” to access the SFS file, as shown in Figure 15-34.

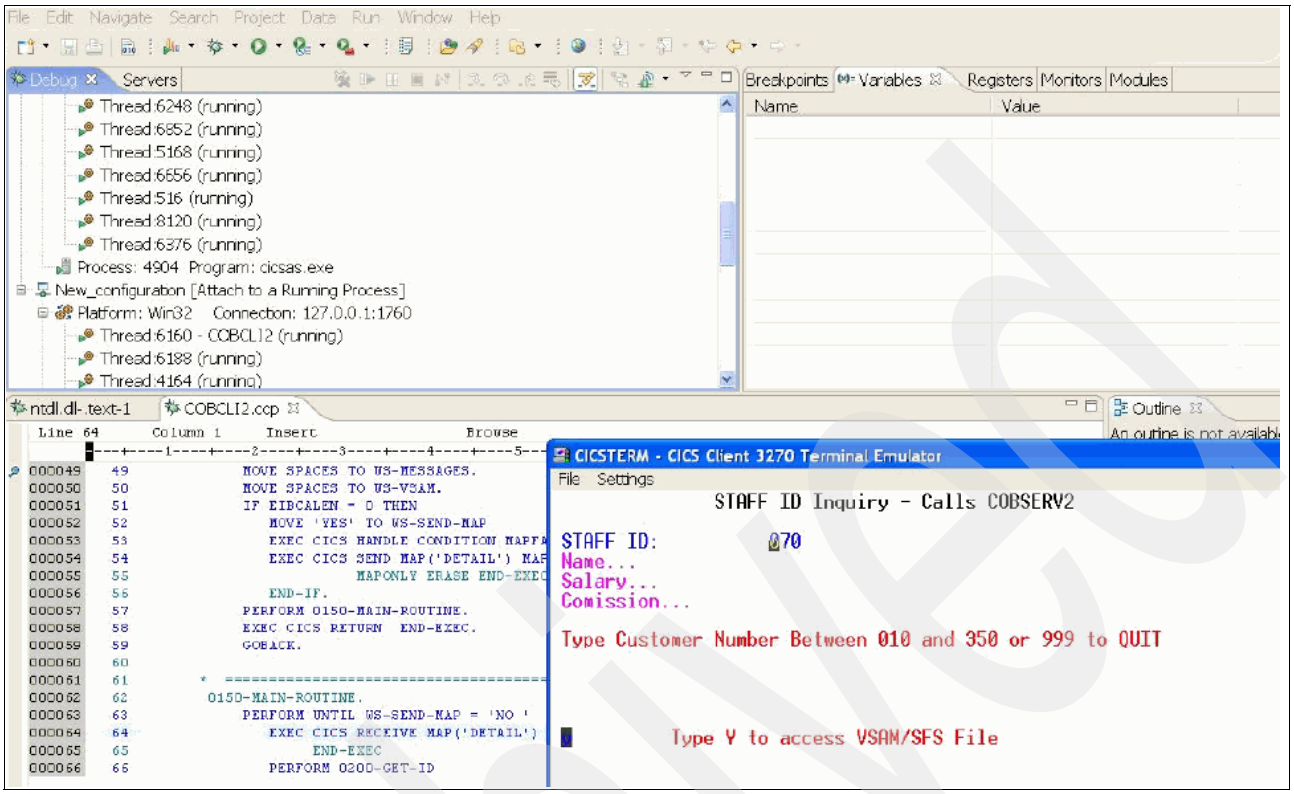


Figure 15-34 Map

Archived

In Figure 15-35 and Figure 15-36 on page 552, you notice, in the Variables view, that the ID is 070 and “Y” is the value for the variable WS-VSAM. We can modify the value if we want to.

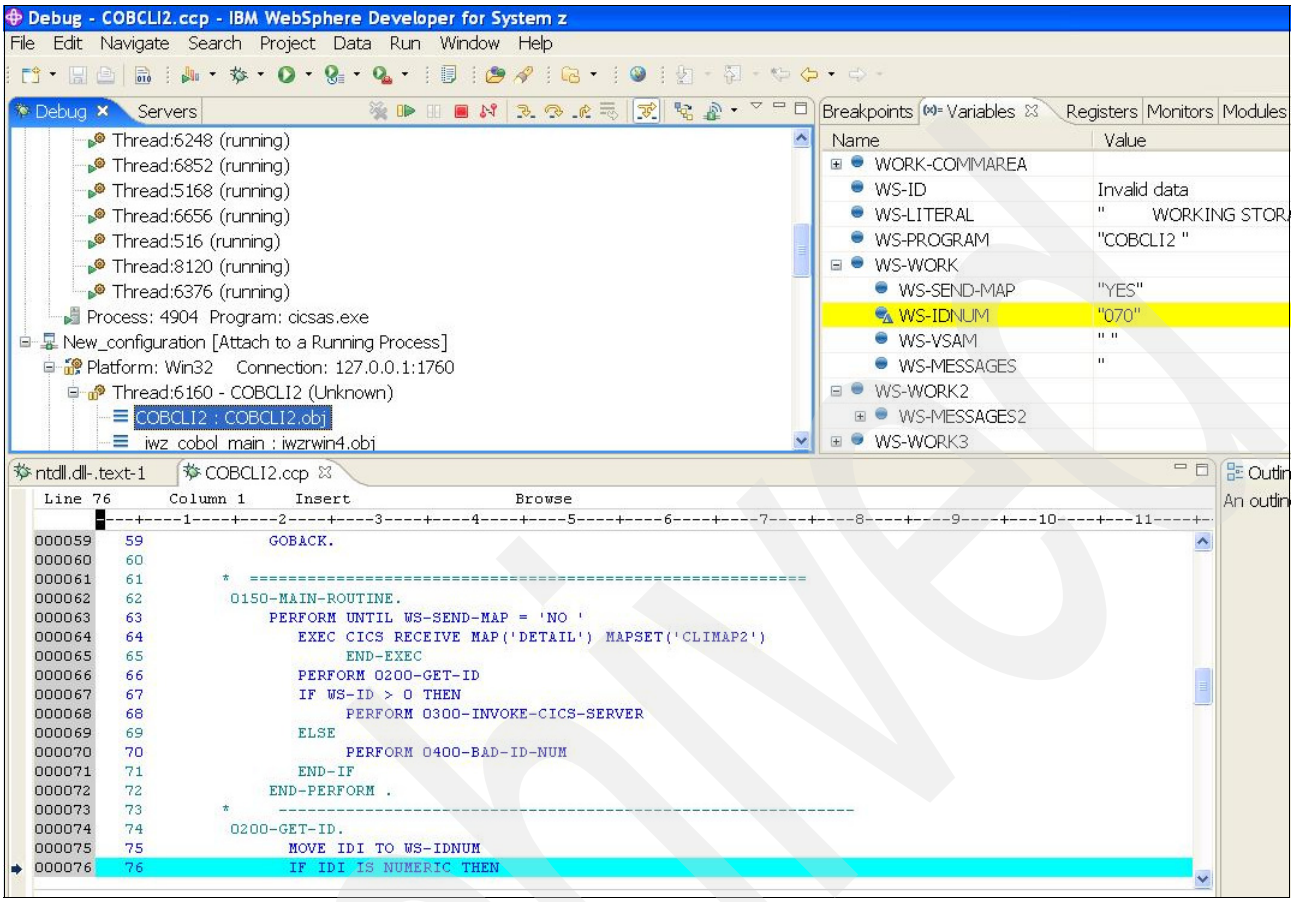


Figure 15-35 Listing

Archive

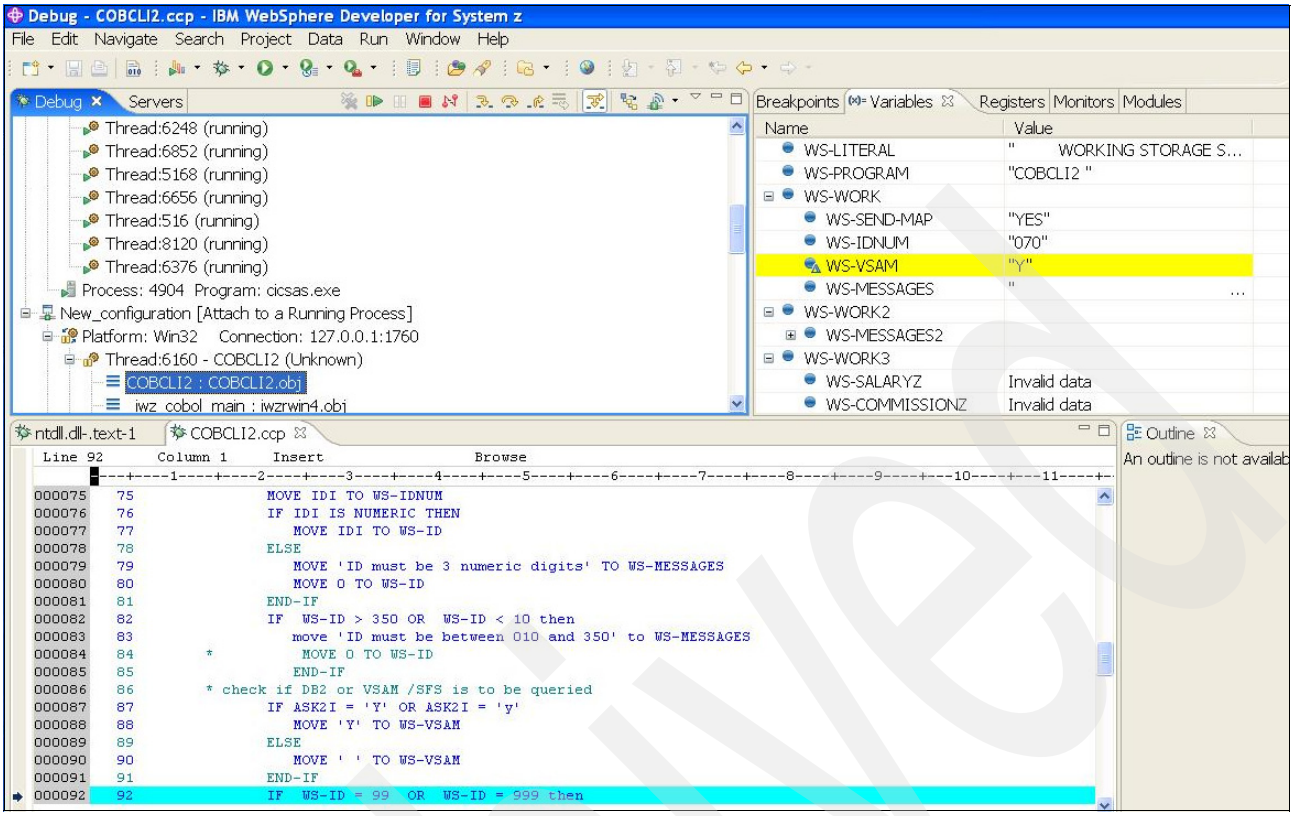


Figure 15-36 Listing

Archive

Control is being transferred through EXEC CICS LINK to COBSERV2, as shown in Figure 15-37.

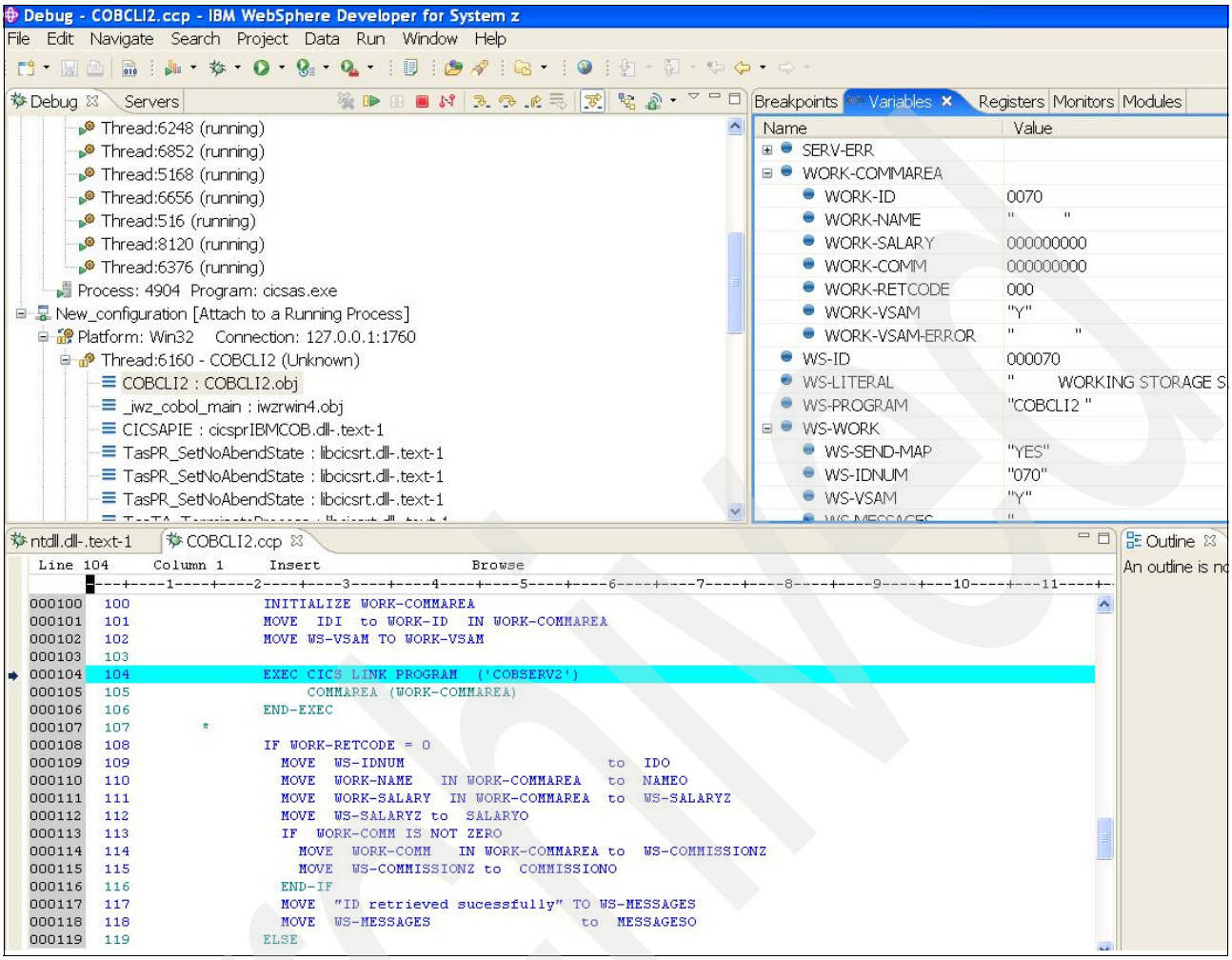


Figure 15-37 Listing

Archive

The server program COBSERV2 has now been loaded and the COBSERV2.ccp listing opens, as shown in Figure 15-38.

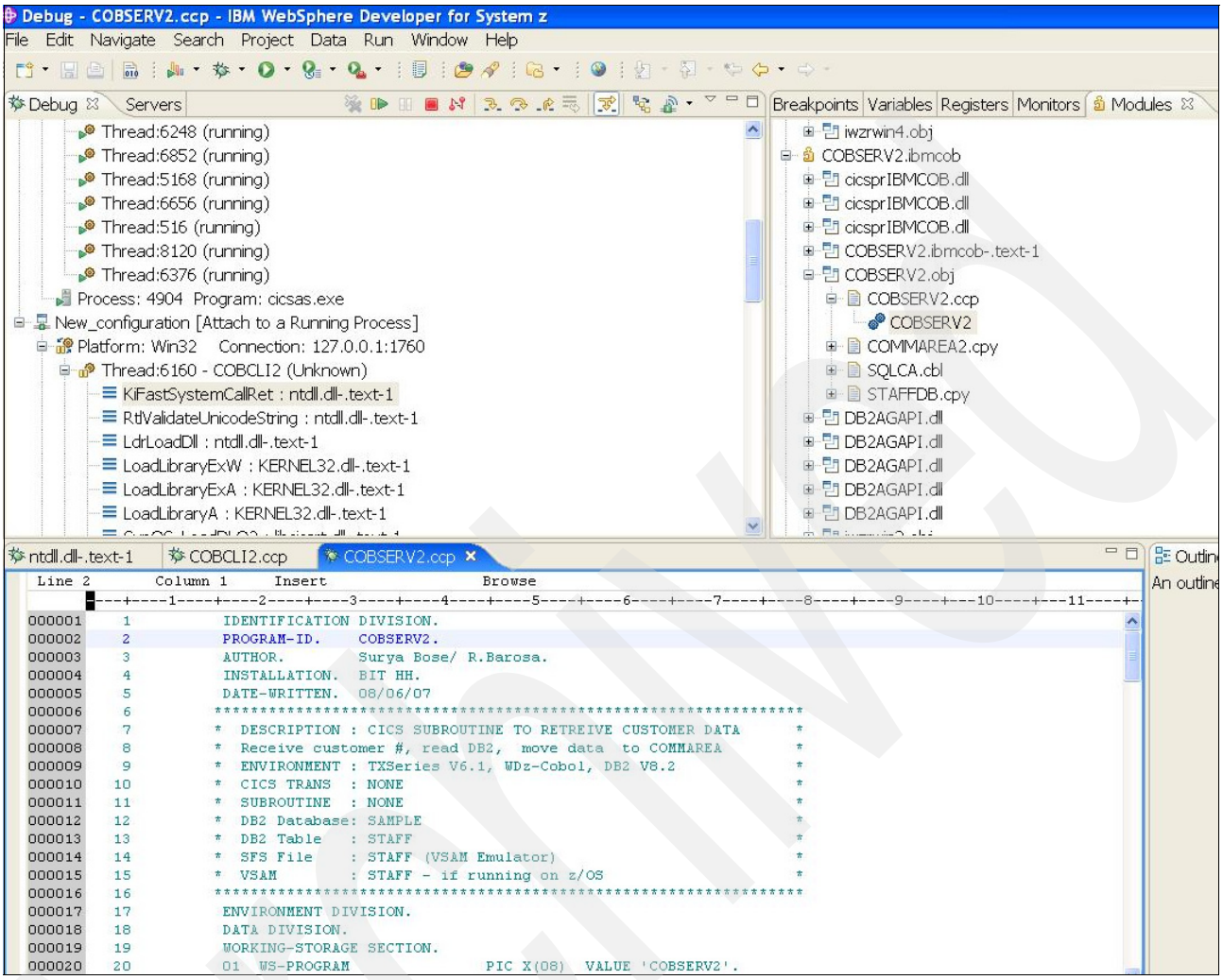


Figure 15-38 Listing

Archive

We are now in COBSERV2 and can add breakpoints, as shown in Figure 15-39.

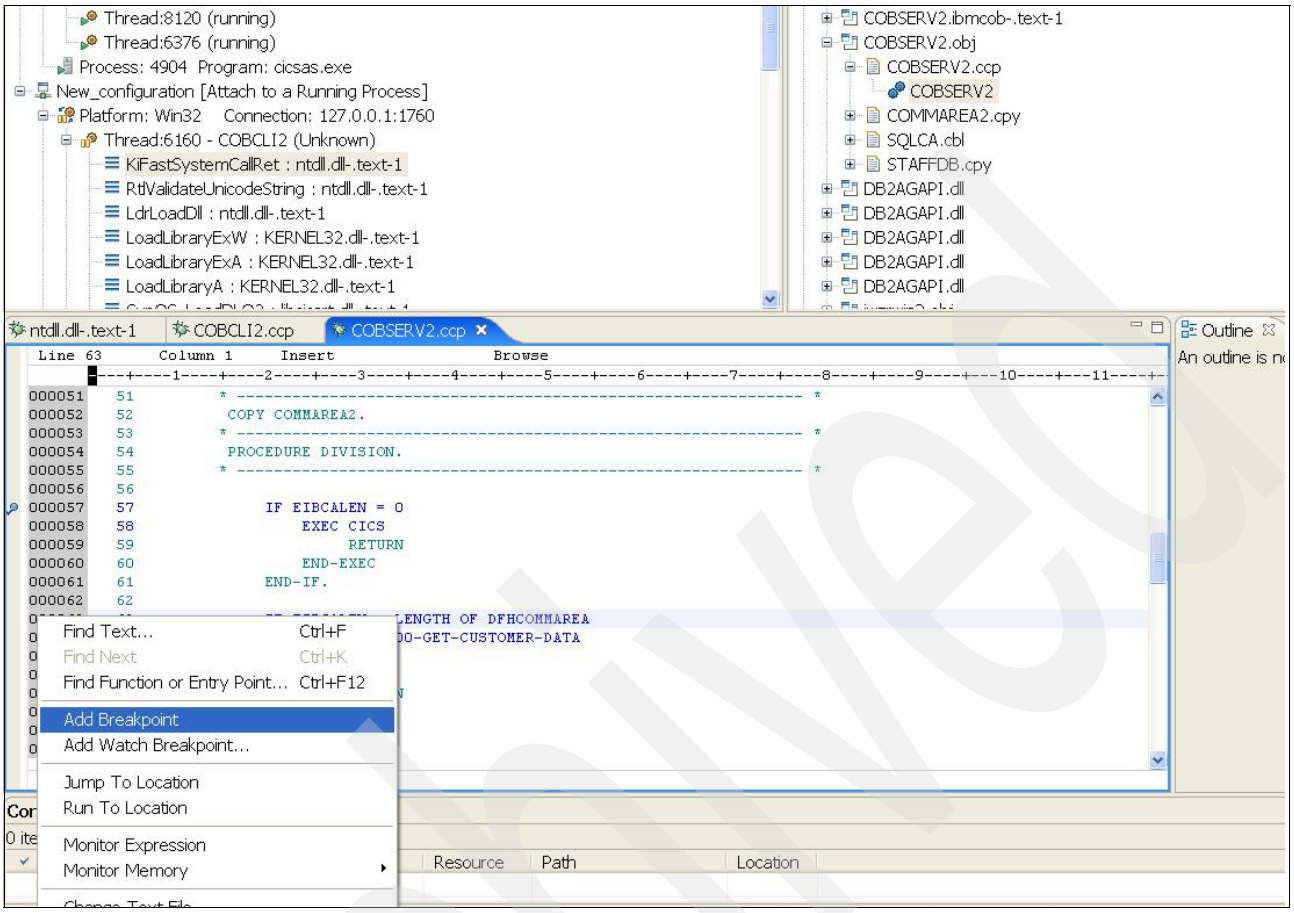


Figure 15-39 Adding a breakpoint

Archive

In the Variables view, we can see the values in the Commarea, as shown in Figure 15-40.

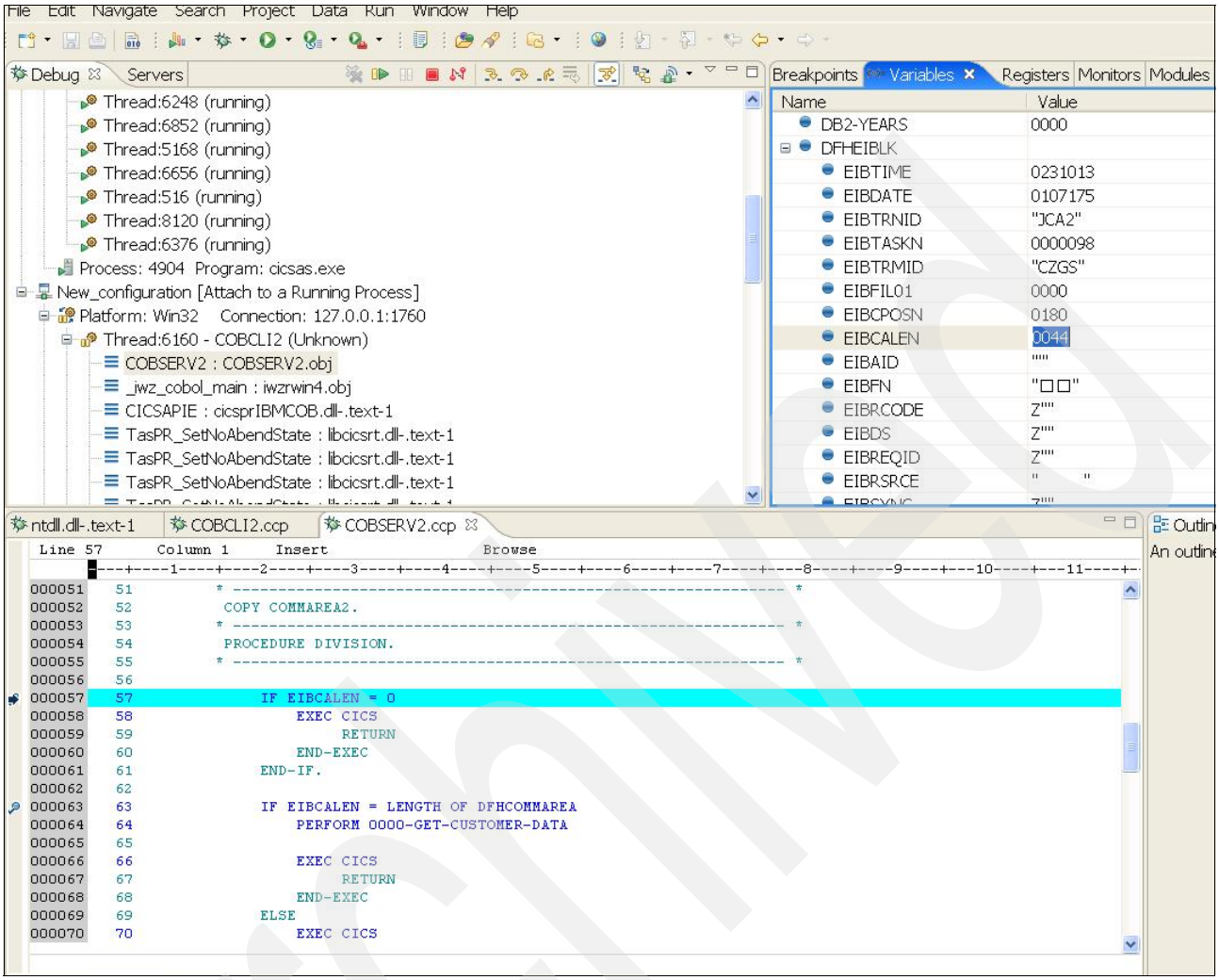


Figure 15-40 Variables view

Archived

Because WORK-VSAM = “Y”, the program will access SFS and not DB2 in this case, as shown in Figure 15-41.

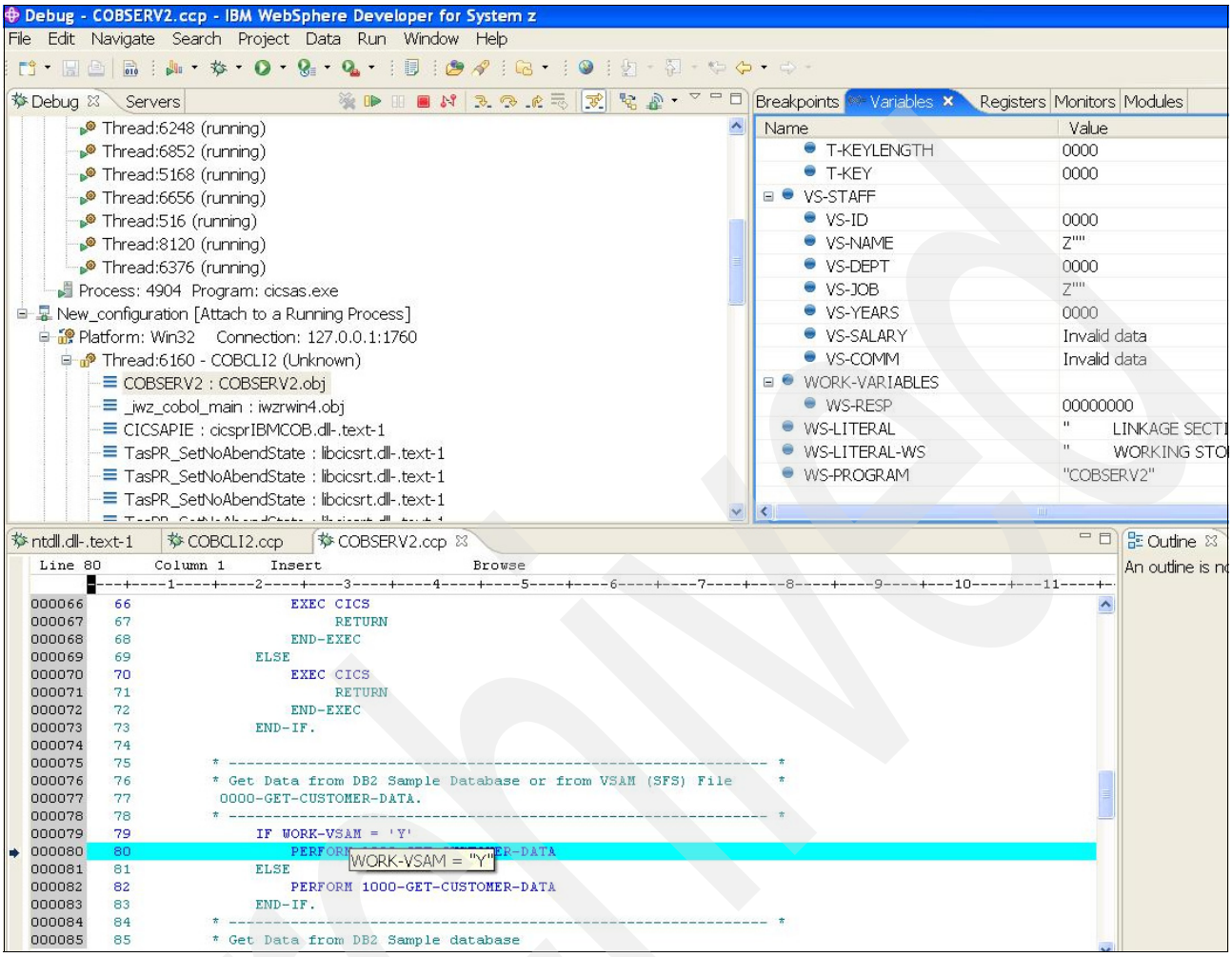


Figure 15-41 Listing

While preparing the EXEC CICS READ FILE (STAFF) command, we can see, in the Variables view, that the key is 070, as shown in Figure 15-42.

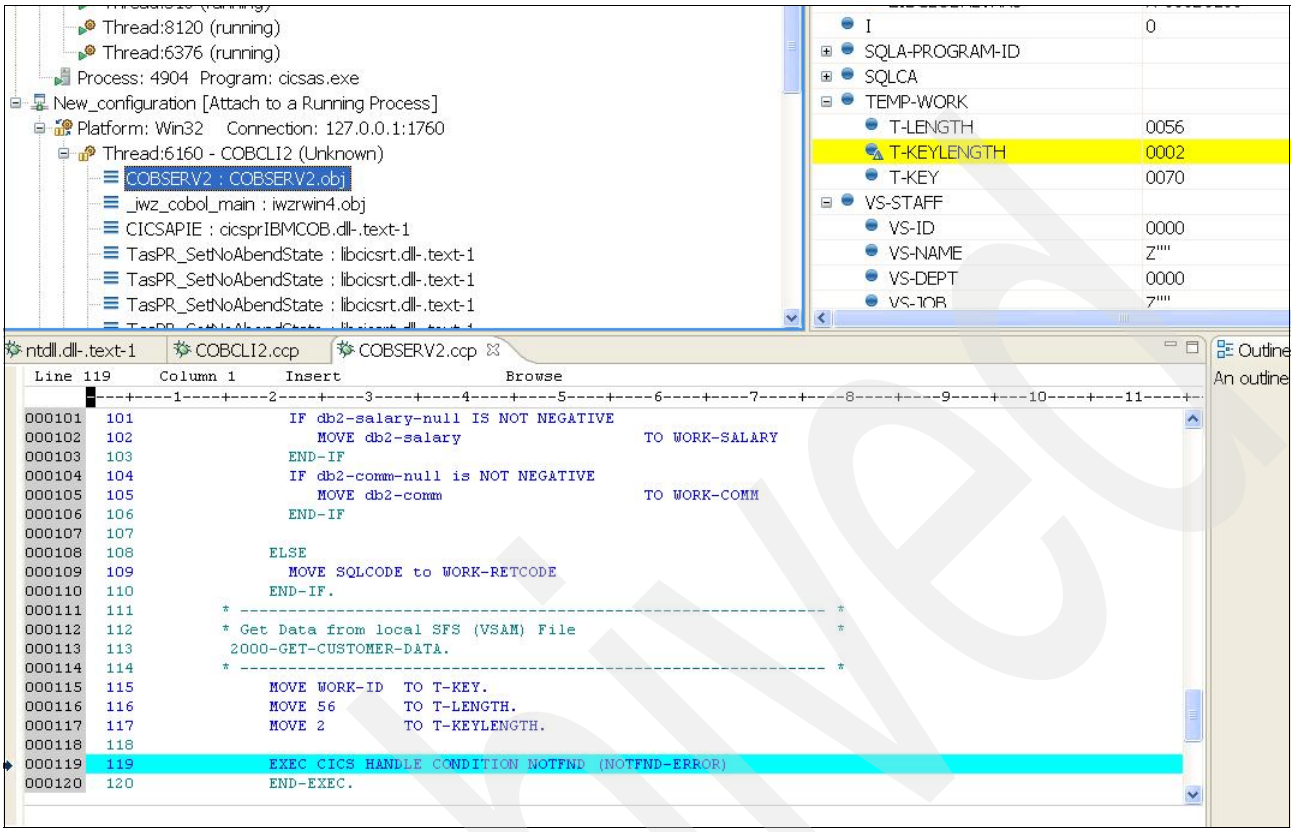


Figure 15-42 Listing

Archive

The SFS File STAFF is read with key (= 070), as shown in Figure 15-43.

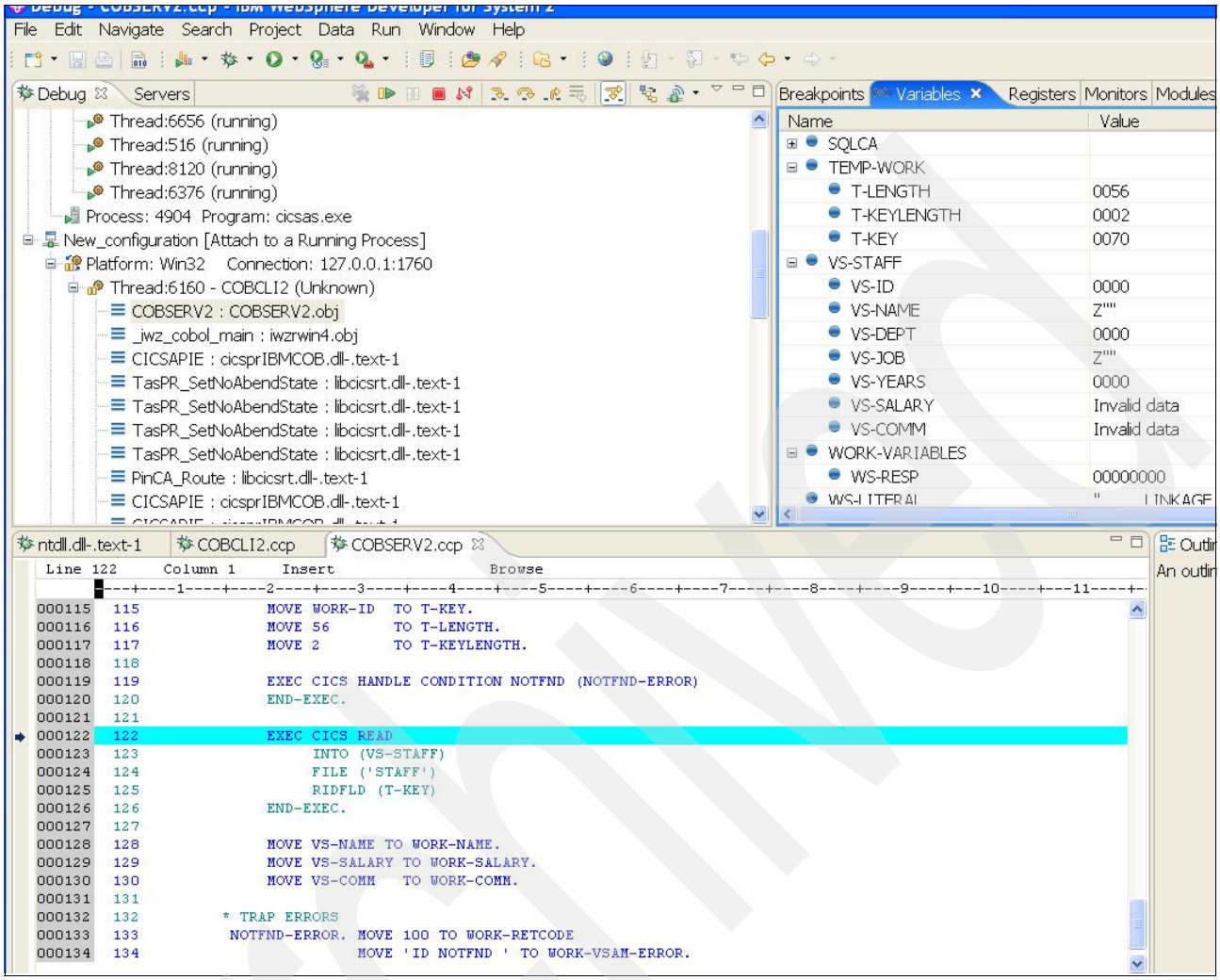


Figure 15-43 Listing

Archive

The VSAM / SFS Read has been successful. The retrieved data can be seen in the Variables view, as shown in Figure 15-44.

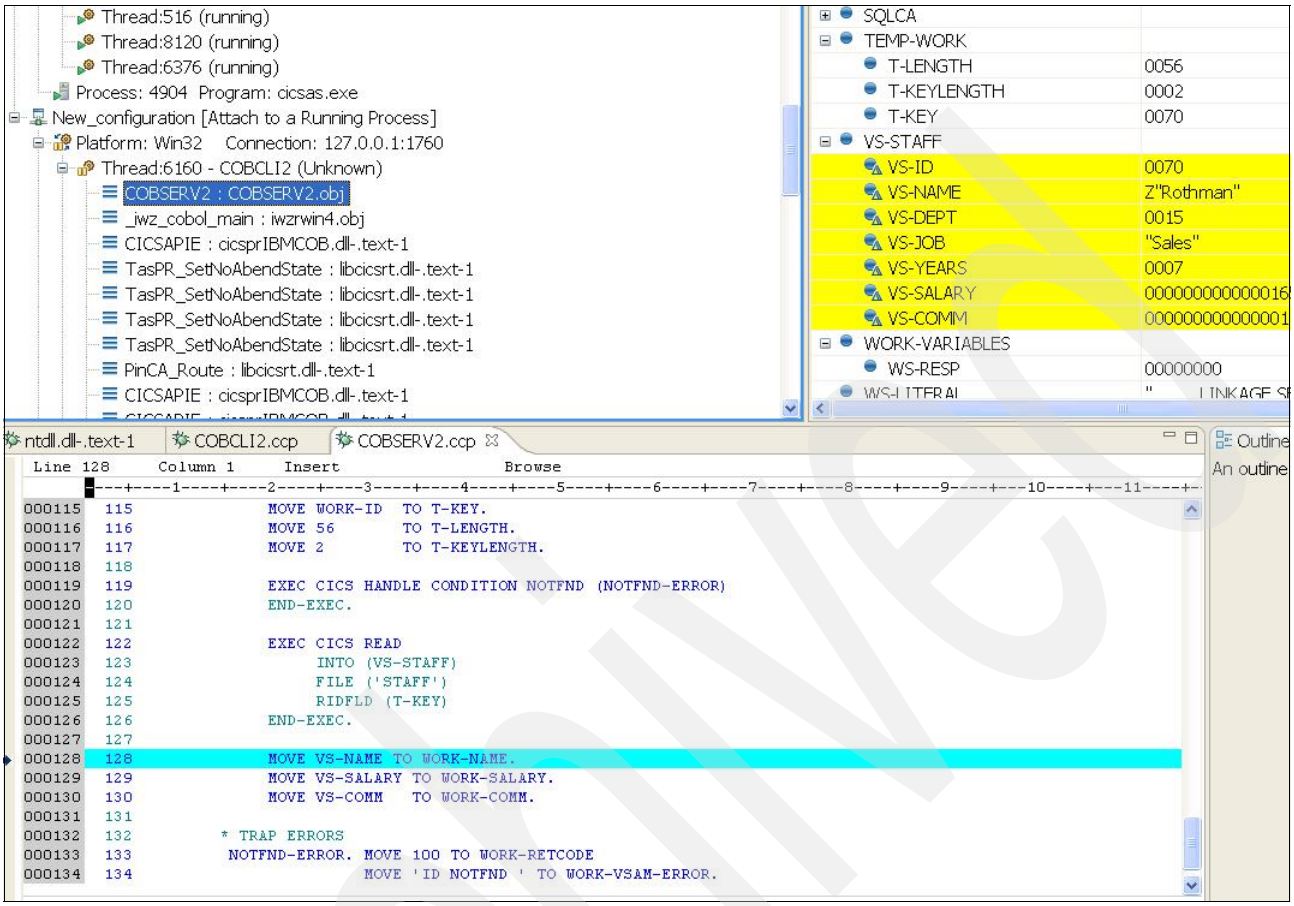


Figure 15-44 Listing

Archive

The result has been transferred to the client program COBCLI2 and displayed on the Map, as shown in Figure 15-45.

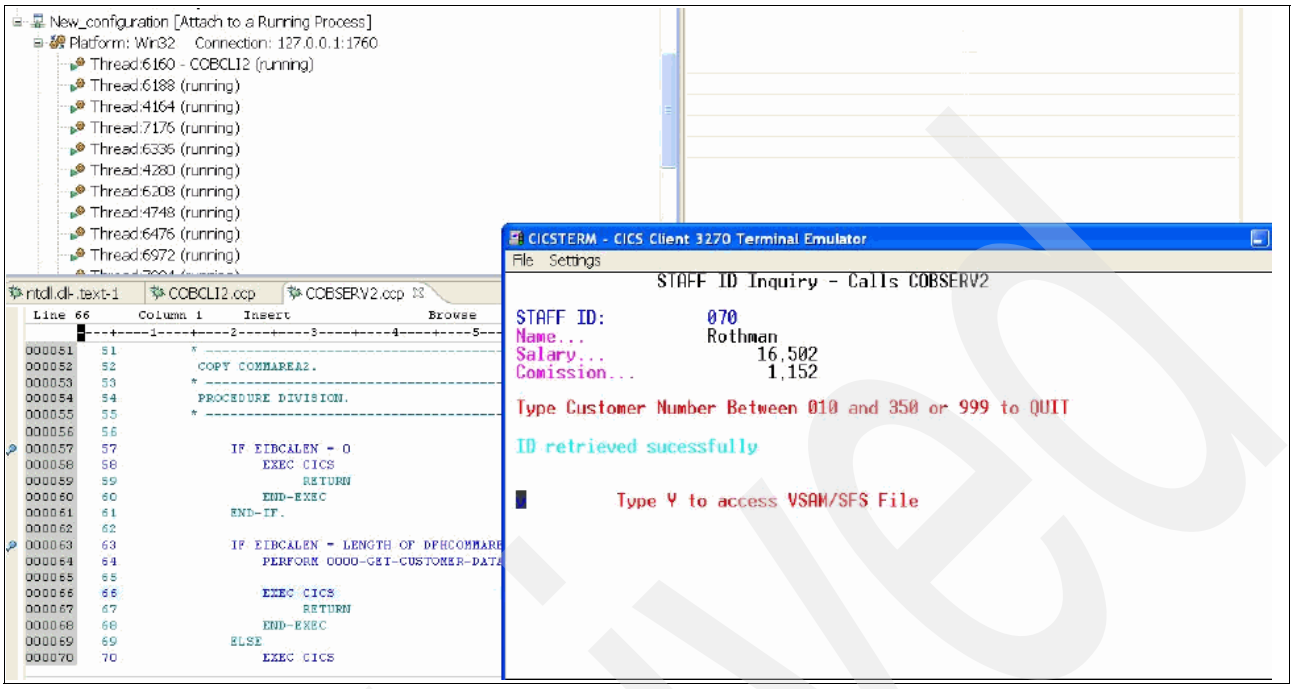


Figure 15-45 Map

Conclusion

We have seen that EXEC CICS commands for VSAM on the mainframe do not have to be changed to access SFS on Windows. Even when mainframe connectivity is a problem, development and testing can still be done with WD/z and TXSeries on Windows with DB2 and SFS, thereby providing uninterrupted development support.

Archived

Archived

16

z/OS Database Application Generator

This chapter discusses a feature that was introduced in WebSphere Developer for System z Version 7 named z/OS Database Application Generator, also known as System z Application Pattern Generator when it was a technical preview.

The z/OS Database Application Generator wizard automatically generates CICS COBOL programs that access DB2. Additionally, a Web service is provided, as well as the JCL to assist in the deployment of the code generated.

In this chapter, we give you a small introduction to this feature and provide two example scenarios where you could use this feature. The following topics are discussed in this chapter:

- An introduction of z/OS Database Application Generator.
- Scenario 1 Introduction: Working with the existing z/OS DB2 table.
  - How to create a DB2 table to be used by the Database Application Generator using WD/z.
  - Using the Database Application Generator wizard to create a Create, Read, Update, Delete (CRUD) and List applications.
  - Understanding the generated assets.
  - How to deploy the generated assets to a z/OS system.
  - How to configure the CICS Web Services on a z/OS system.
  - How to install the z/OS CICS Services.
  - How to test and debug the deployed CICS Web Services.
- Scenario 2 Introduction: Working with UML.
- How to create and test a simple Web Client using Faces JSP (JSF) that invokes the CICS Service deployed.

© Copyright IBM Corp. 2009. All rights reserved.

Note: Refer to the following files that are included as Web material with this book:

- SG247482-Chapter 14.zip file (scenario code)
- SG247482_Viewlet_SCENARIO_1.zip (audio self-paced overview of Scenario 1
- SG247482_Viewlet_SCENARIO_2.zip (audio self-paced overview of Scenario 2)

Information about downloading these files can be found in Appendix B, “Additional material” on page 645.

Archived

16.1 z/OS Database Application Generator

The z/OS Database Application Generator wizard automatically generates CICS COBOL code that accesses DB2. Additionally, a Web service is provided, as well the JCL that will help you implement the generated code.

The Database Application Generator provides Create, Read, Update, and Delete (CRUD) and list operations for the COBOL programs generated.

The generated COBOL code consists of two programs per DB2 table. The first is the standard CRUD operations set acting on a single record. The second program handles the search/list operation for multiple rows.

The Database Application Generator enables you to use an existing DB schema to rapidly create a z/OS data access layer for Web Services. In addition, you can work directly from a Unified Modeling Language (UML) model.

After the transformation is generated, configuration information is stored in the transformation parameters file (TPM file), which can be modified with the Transformation Parameters Editor (TPM Editor). The transformation parameters file allows you to customize the transformation to source code.

For full CICS functionality, Database Application Generator should be used with CICS Version 3.1 or later.

Figure 16-1 shows the Database Application Generator architecture.

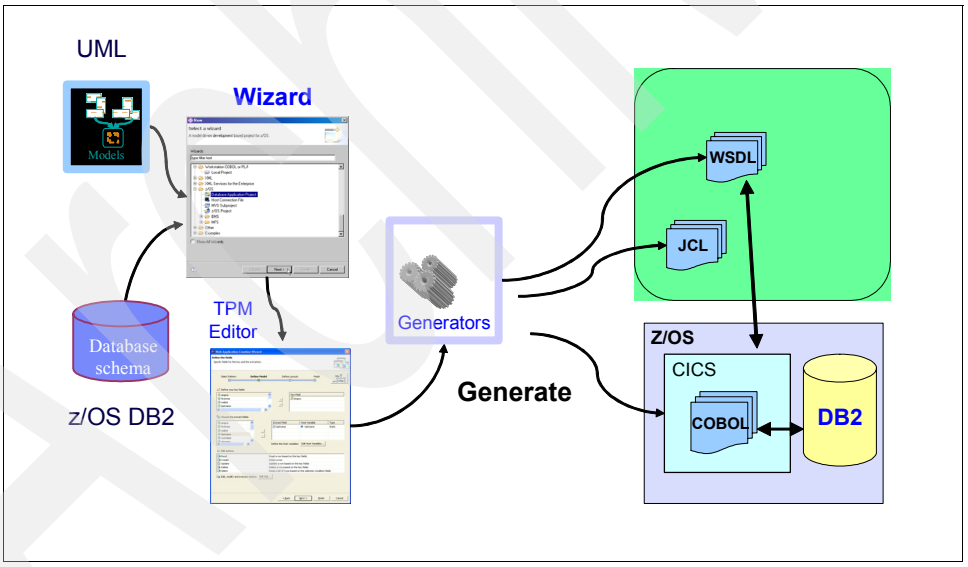


Figure 16-1 Database Application Generator architecture

Archived

16.2 Using the Database Application Generator

There are two paths for using the Database Application Generator:

- The first path, which we believe it is the main path, is where the wizard will be used to connect to an existing database to retrieve the table(s) schema and populate our model for users to modify within the TPM editor. The transformations are always done from the TPM editor.
- The second path, where you start from a UML2 (*.uml) model (probably exported from Rational Software Architect) and open the TPM editor. In this second scenario, we make assumptions that all UML classes in the model correspond to a DB2 table that may or may not have been created yet. You must manually or through some other means create the DB2 table if it has not been created yet.

16.3 Working with the UML model

When doing model driven development, you can work from the UML model that corresponds to a DB2 table that may or may not have been created yet.

In this scenario, you need to have the z/OS DB2 connections, as in the first scenario, because the DB2 table might not be created already, but later on, to test the generated assets, you must manually or through some other means create the table.

The UML model provides abstract classes and interfaces. You can configure the column properties of these abstract classes and interfaces, unless they are set as private. The reason for doing this is that these properties are generated for concrete classes within the UML that inherit from those abstract classes and interfaces.

We will return to this subject when working with this scenario.

16.4 Working with the existing z/OS DB2 table

We believe that this is the most common scenario, where you want to create the COBOL assets that use an existing z/OS DB2 table and generate CICS Web Services.

When you use an existing DB2 table, during Database Application Generator, this wizard asks you to connect to a z/OS DB2 database or use an existing database connection. You cannot generate the assets without having this database connection.

Also, when using this scenario, we recommend that you create a z/OS system connection in the Remote System Explorer and establish the z/OS remote connection. Otherwise, you might not be able to enter values in some of the Transformation Parameters Editor fields.

Archived

16.5 Scenario 1: Working with the existing z/OS DB2 table

In this scenario, we assume that there is an existing DB2 table in the z/OS and you want to use the Database Application Generator to create the assets. In this chapter, we show you the main steps that you must perform to make that happen.

Our sample DB2 table has various DB2 data types. Sometimes it is not an easy task to find the required COBOL definitions for an specific DB2 data type, and this is where this wizard might also help you.

The data types (using null and not null) that we have in our sample table are:

- INTEGER
- VARCHAR
- TIME
- TIMESTAMP
- SMALLINT
- CHAR
- FLOAT
- DECIMAL
- DATE

The generated COBOL code handles this DB2 table.

16.5.1 Creating a DB2 table to be used by the Database Application Generator

Usually a DB2 table is created by someone that has database authorizations, using DB2 tools, but here we show you some capabilities in WD/z that may help you with this task (these capabilities are not part of the wizard).

Before you start, you must be sure that you have access to the z/OS system, that WD/z is installed on your machine, and that you have the correct authorizations for the z/OS DB2.

Using the WD/z Data Perspective

The WD/z Data Perspective is the development environment for working with databases in many platforms, including System z. It allows you to connect to local or remote databases and perform database activities.

The default Data Perspective window contains many views, but four are very important and are explained below:

Data Project Explorer
: The Data Project Explorer is where data development projects appear. The Data Project Explorer provides features to manipulate these database objects and export them to the remote database. This view can be used to create a DB2 table.

Database Explorer
: This view contains views for specified databases. From this view, you can create and edit connections to DB2 databases, display the contents of database tables, and drop database tables and stored procedures.

Archived

Data Output View This view contains a table of all running and completed processes, details of a particular process, information about the completion of the process, the input and output parameter values when a stored procedure has been invoked by a run task, the result sets when a query is executed, and so on.

Remote Systems The Remote Systems view contains connections to remote systems and shows a directory of the data sets of the remote systems.

Figure 16-2 shows the WD/z Data Perspective layout.

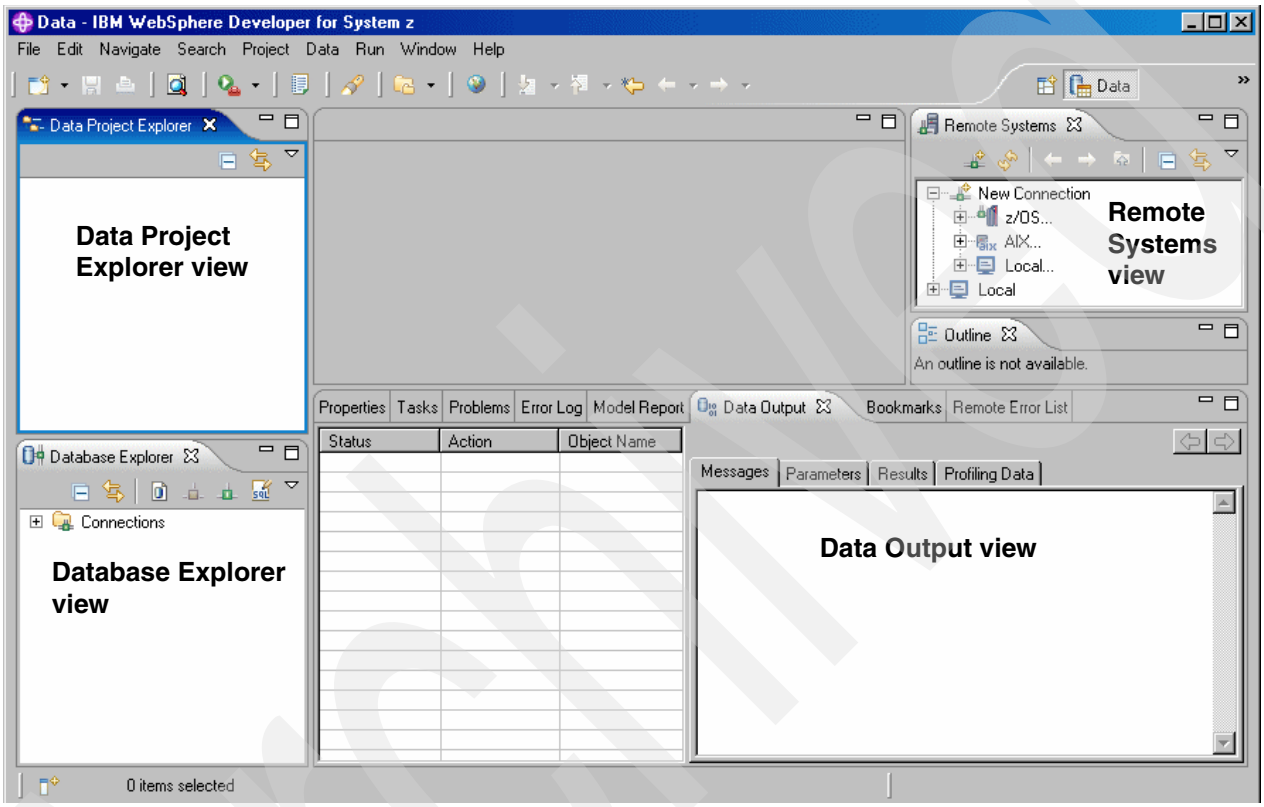


Figure 16-2 Data Perspective layout

We use this perspective to create, load, and work with the z/OS DB2 table.

Using a DB2 Data Design Project to create and load the DB2 table

A data design project is primarily used to store SQL modeling objects. You can store the following types of objects in a data design project:

- Logical data models
- Physical data models
- Domain models
- Glossary models
- SQL scripts, including Data Definition Language (DDL) scripts
- If you have Information Integrator installed: Mapping models and XML schemas

Any other file types, such as .doc files, text files, presentations, or spreadsheets, can also be stored in a data design project. Any files other than the files listed above or Eclipse-specific files (such as project files) are displayed in the Other files folder under a data design project in the Data Project Explorer.

Using the New Data Design Project wizard, you specify basic information about the data design project, including the name and the local directory in which to store files. You can also specify project references.

To create a data design project and load the DDL file, do these steps:

1. Select **File** → **New** → **Data Design Project**. As an alternative, right-click in any blank space in the Data Project Explorer and select **New** → **Project** → **Data Design Project**. The New Data Design Project wizard opens.
2. Complete the steps of the wizard, then click **Finish**. The data design project is displayed in the Data Project Explorer view
3. Using this project, we can import a DB2 DDL that creates the z/OS DB2 table in the SQL Scripts folder. Example 16-1 shows the DDL that we used to create this table, and Figure 16-3 on page 570 shows the Data Explorer View with this file imported.

Example 16-1 DDL used to create a DB2 COBOLTYPES table

```
-- create the table COBOLTYPES on zOS
DROP TABLE COBOLTYPES;
CREATE TABLE COBOLTYPES
( ID INTEGER NOT NULL,
 TITLE                          VARCHAR(250) NOT NULL,
 DESCRIPTION                    VARCHAR(1000) NOT NULL,
 OWNER                          INTEGER NOT NULL,
 AUTHOR                         VARCHAR(100),
 PUBLISH_DATE                   VARCHAR(32),
 PUBLISH_TIME                   TIME,
 PUBLISH_STAMP                  TIMESTAMP,
 PUBLISH_INT                    INTEGER,
 PUBLISH_SMALLINT               SMALLINT,
 PUBLISH_CHAR                   CHAR(5),
 PUBLISH_FLOAT1                 FLOAT(4),
 PUBLISH_FLOAT2                 FLOAT,
 PUBLISH_FLOAT3                 FLOAT,
 PUBLISH_REAL                   FLOAT(4),
 PUBS_NEXT                      DECIMAL(19, 0),
 USERS_NEXT                     DECIMAL(15, 2),
 TYPE                           VARCHAR(32) NOT NULL,
 ISBN                           VARCHAR(12),
 CUR_CHKOUT_USER                VARCHAR(100),
 CUR_CHKOUT_DATE                DATE,
 PRIMARY KEY (ID));

CREATE UNIQUE INDEX XCOBTYPE
    ON COBOLTYPES
      ( ID ASC);
```

Archived

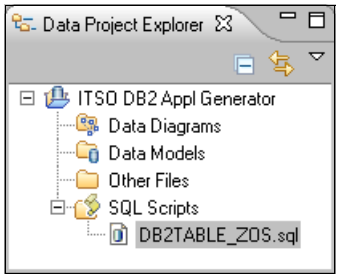


Figure 16-3 Data Project Explorer view with the DDL file to create the DB2 table

Running SQL statements

You can run and test SQL statements that you create. You need to create a database connection that points to the z/OS system that you use.

Here are the steps required to create the DB2 table using WD/z:

1. Using the Data Project Explorer view and using the SQL Scripts folder of the data design project, right-click the file **DB2TABLE_ZOS.sql** and then click **Run SQL**.
2. In the Connection Parameters window, you must select **Create a new connection** and click **Next** (or select an already existing database z/OS connection).
3. Select **DB2 UDB zSeries**, your DB2 version, your Location, your Port number, and z/OS user ID and password, as shown in our example n Figure 16-4 on page 571. Click **Test Connection** to verify that the connection works.
4. If the connection is successful, click **Next**.
5. In the Schema Filter, you can select **Disable filter** and use the expression “Contains the characters”, leaving your default user ID as the value and click **Finish**.
6. If you have the right authorizations, the z/OS DB2 table is created. You can use the Data Output view to check the messages, as shown in the Figure 16-5 on page 571. Note that we had a execution failure. This is because the DDL executed has a drop table and this table was not created causing a drop failure. This is normal and you may expect the same results if you use the DDL that we provided for your tests.

At this point, you are ready to start the tests using the Database Application Generator and an existing z/OS DB2 table. Note that our table example has no data; you need to insert data later when testing the executions.You might also use the Database Explorer view and verify the table created.

Archived

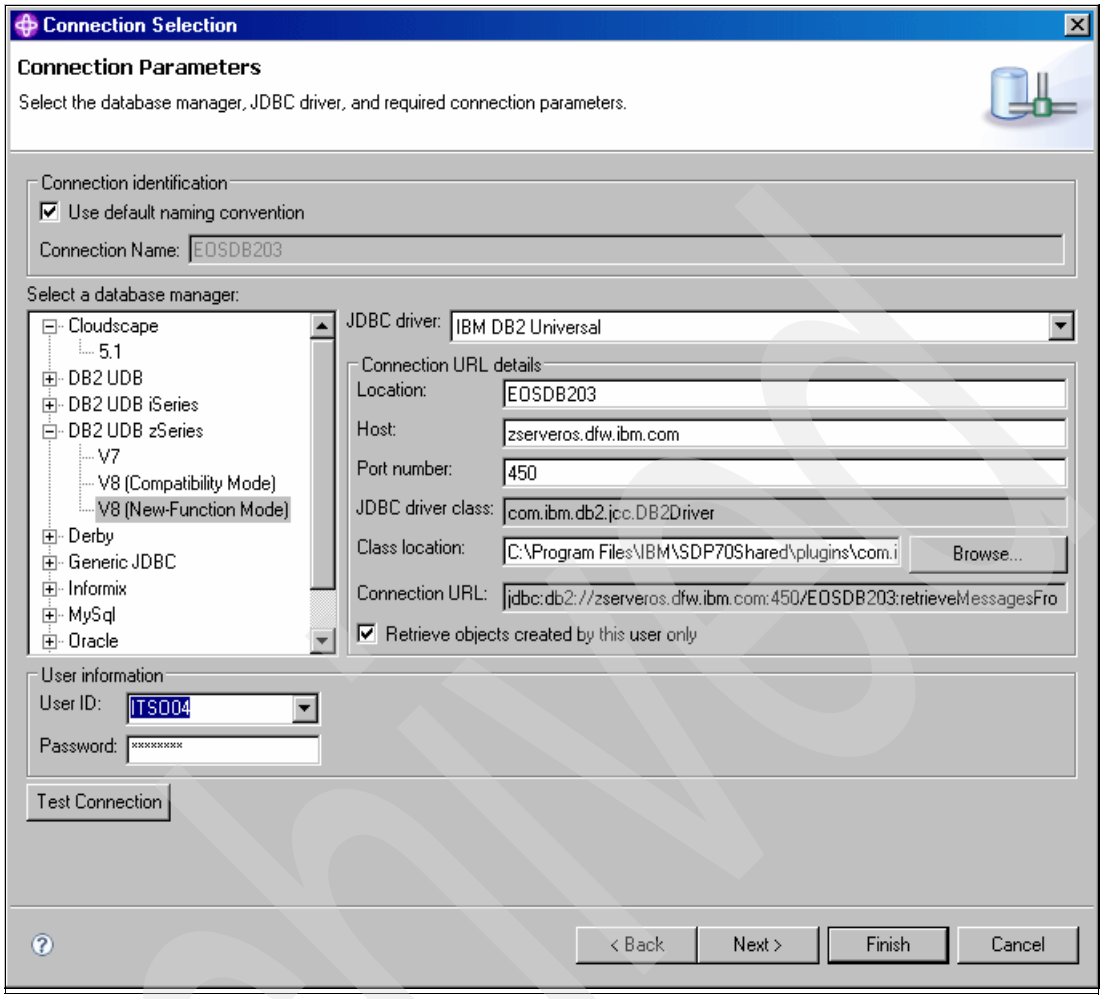


Figure 16-4 DB2 Connection Parameters used in our example

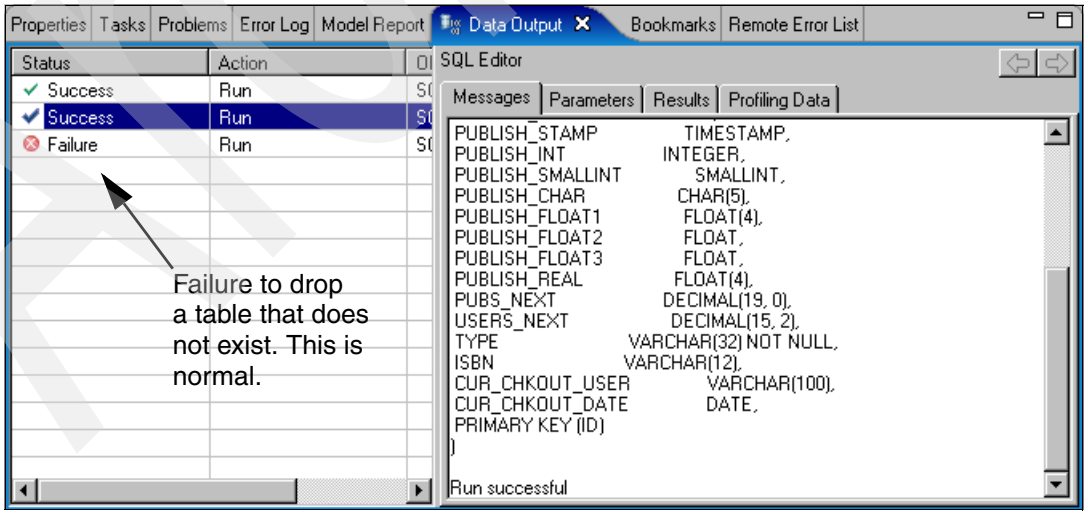


Figure 16-5 Data Output showing the DDL execution

16.5.2 Using the Database Application Generator wizard to create a CRUD and list applications

Before you start the Database Application Generator wizard, we recommended that you define a z/OS remote system and establish the connection. Otherwise, you might not be able to enter values in some of the Transformation Parameters Editor fields.

Defining a WD/z z/OS remote system

In our example, we defined a connection named *ITSO* following these steps:

1. Using the Remote System Explorer perspective or the z/OS Projects perspective, select the Remote Systems view, double-click **New Connection**, and select **z/OS**.
2. Right-click and choose **New Connection**. The New Remote z/OS System Connection window opens.
3. Select a profile name from the drop-down list. If this is your first time defining a remote connection, a window opens prompting you to create a new profile. In the New Remote z/OS System Connection window, the profile name you enter is displayed.
4. Enter the following values in the fields in this window:
   - Connection name, for example, ITSO
   - Host name, for example, zserveros.dfw.ibm.com
5. To define the connection using default values for the JES, MVS Files, and UNIX System Services Files subsystems, use the default properties and click **Finish**.
6. You could also set properties for these subsystems, but to simplify, you might just define properties for the MVS subsystem importing already defined properties that must match the z/OS data sets that you allocate. This might be done by right-clicking **MVS Files**, selecting **Import Properties**, and pointing to the provided properties file named **MVS_Files_Properties.xml**.

   You might need to adjust those properties to match your z/OS system, but because those properties are not used in the Data Base Application Generator wizard, you need to generate a JCL and do remote syntax checking or similar tasks.

Connecting to a z/OS remote system

To connect to a remote system, follow these steps:

1. Using the Remote Systems view, right-click a connection name, such as **ITSO** in our example, and select **Connect**.
2. In the Connect to System window, type your user ID and password and select **OK**.

You are connected to the z/OS system and are able to continue.

Allocating PDS on z/OS

You must have required data sets defined in the z/OS to hold the generated assets, such as COBOL source code, DBRMs, and so on. For our example, we have allocated data sets, as shown in Example 16-2 on page 573. You might have definitions with different characteristics.

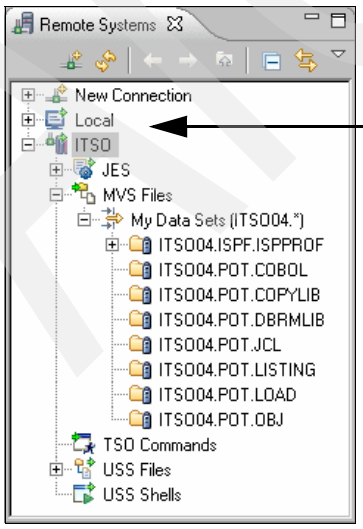
It is your decision about whether you will use the traditional JCL way to allocate data sets or use the WD/z wizards. If you want to use the wizards, you need to be connected to z/OS. Those wizards can be started from the z/OS Projects view or Remote Systems view. One way is by right-clicking **MVS Files** in the Remote Systems view and clicking **Allocate PDS**.

Archived

Example 16-2 JCL used to allocate the data sets in our example

```
//ITSO04A  JOB ,
// MSGCLASS=H,TIME=(,4),REGION=28M,COND=(16,LT)
//*  ************************************************
//*  ALLOCATE DATA SETS NEEDED FOR DB2 APPL GENERATOR WIZARD
//* ************************************************
//ALLOC EXEC PGM=IEFBR14,COND=(4,LT)
//JCLSRC DD DSN=ITSO04.POT.JCL,DISP=(NEW,CATLG,DELETE),
//      RECFM=FB,LRECL=80,BLKSIZE=8000,SPACE=(CYL,(1,1,20))
//COBSRC DD DSN=ITSO04.POT.COBOL,DISP=(NEW,CATLG,DELETE),
//      RECFM=FB,LRECL=80,BLKSIZE=8000,SPACE=(CYL,(2,1,20))
//COBCPY DD DSN=ITSO04.POT.COPYLIB,
//      RECFM=FB,LRECL=80,BLKSIZE=8000,SPACE=(CYL,(1,1,20)),
//      DISP=(NEW,CATLG,DELETE)
//COBOBJ DD DSN=ITSO04.POT.OBJ,
//      RECFM=FB,LRECL=80,BLKSIZE=8000,SPACE=(CYL,(1,1,20)),
//      DISP=(NEW,CATLG,DELETE)
//COBLST DD DSN=ITSO04.POT.LISTING,
//      RECFM=FBA,LRECL=133,BLKSIZE=1330,SPACE=(CYL,(2,1,20)),
//      DISP=(NEW,CATLG,DELETE)
//LOADL DD DSN=ITSO04.POT.LOAD,RECFM=U,
//      BLKSIZE=6233,SPACE=(CYL,(1,1,20)),
//      DISP=(NEW,CATLG,DELETE)
//LAB802 DD DSN=ITSO04.POT.DBRMLIB,DISP=(NEW,CATLG,DELETE),
//      RECFM=FB,LRECL=80,BLKSIZE=8000,SPACE=(CYL,(2,1,20))
/*
//
```

It is much easier to use WD/z than traditional TSO/ISPF for the data set allocations, either through wizards or through submitting JCL for execution. if you are defining through JCL and submitting for execution, then, using the Remote Systems view, edit the file (you can use Local Files) that has the JCL for the data sets allocations (shown in Example 16-3 on page 590 and submit the changes to the z/OS connected system using the command **`submit to ITSO`**. Be sure that the data sets are correctly allocated in your system. Figure 16-6 shows those data sets allocated on our system.



You can use the Local Files to point to your windows directory, edit the JCL and submit to ITSO for execution. Results can be seen in the JES subsystem.

Figure 16-6 z/OS data sets necessary for our example

Database Application Generator preferences

The Database Application Generator preferences enable you to specify deployment settings for specific target systems. You might optionally modify the preferences for your workspace.

To perform this step, select **Window** → **Preferences** → **z/OS Database Application**.

Figure 16-7 shows one example for the MVS Deployment. You also might define preferences for CICS, such as Transaction Name, DSN System, and so on, as well for the UNIX System Services, such as WSBIND file location, WSDL file location, and so on.

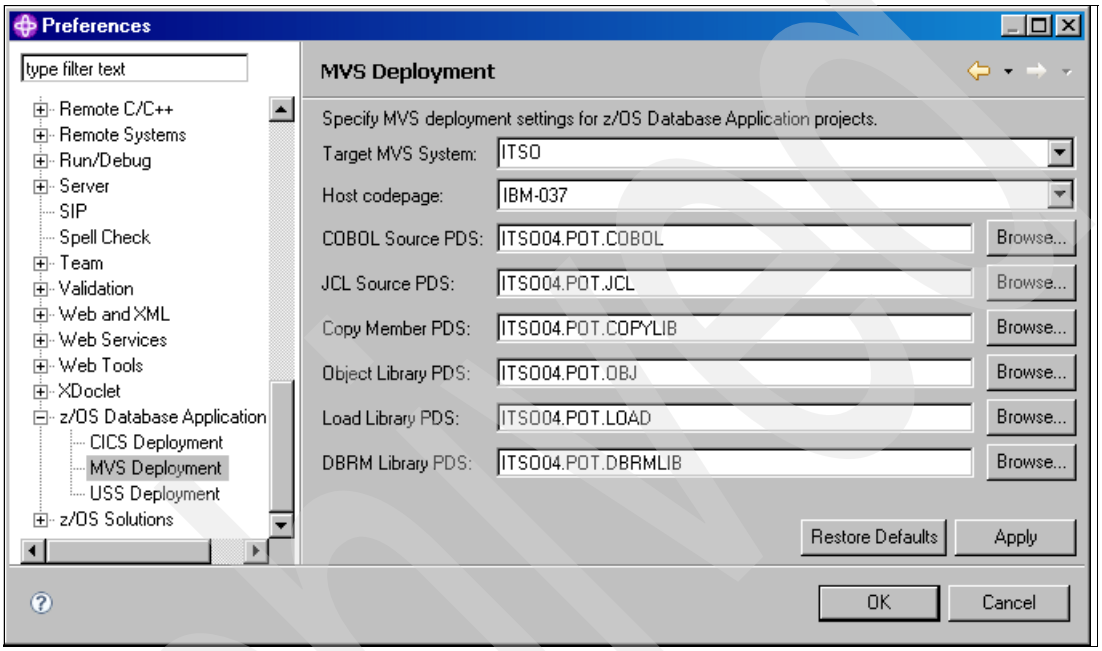


Figure 16-7 MVS Deployment preferences

Using the wizard

To create a new Database Application Generator project, do the following:

1. Select **File** → **New** → **Project**, or you can also use the Ctrl+N key combination.
2. Expand the **z/OS** category, select **Database Application Project**, and click **Next**.
3. In the New Database Application Project window, specify the Name of the new project, such as ITSO Appl Generator, use the default location, and click **Next** to continue.
4. In the Specify Database Connection Information window, you can use a previously created DB2 connection that exists in the workspace or create a new one. Because we have already created one, we select **Use an existing connection**.
5. The properties are displayed and you must click **Next** to continue. Also, for an existing connection, you might be prompted to enter a password.
6. The Specify Tables for Database Application window lists the available tables in the database. Select the tables for which you want to create Web Services transformations. In our example, it is ITSO04.COBOLTYPES, as shown in the Figure 16-8 on page 575. Click **Finish** to complete the project.

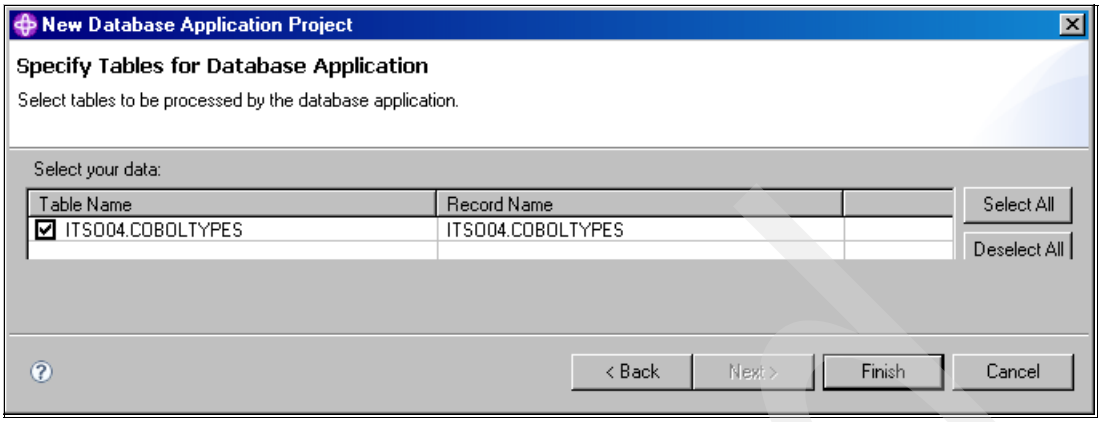


Figure 16-8 Specifying the DB2 Tables for the Database Application

You may notice that you do not see the project that has been created. You could change to Resource or Enterprise Service Tools perspectives or you may open a new view to be able to see it. To open a Project Explorer view, do these steps:

1. Select **Window** → **Show View** → **Other...**, expand **General** → **Project Explorer**, and click **OK**.
2. Drag and drop the new view on top of Data Project Explorer and expand **ITSO Appl Generator** and **models**, as shown in Figure 16-9.

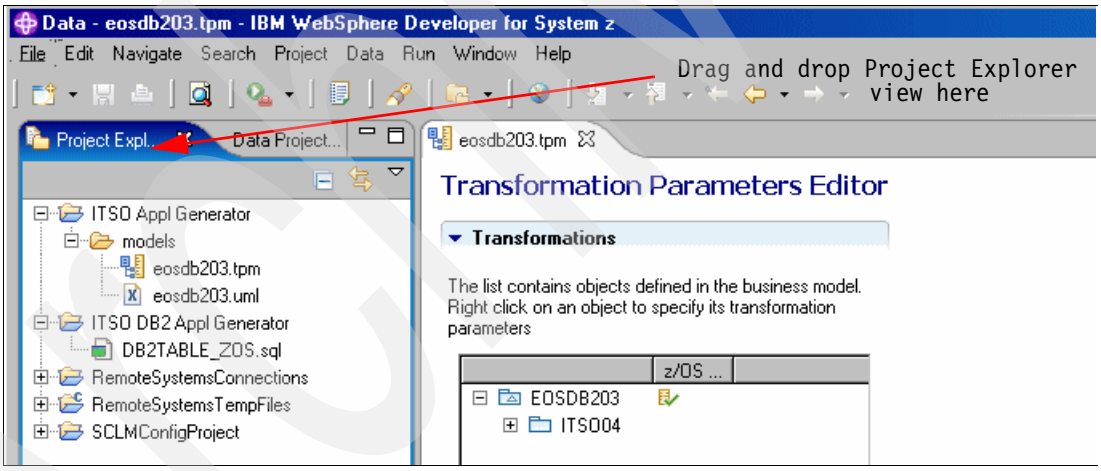


Figure 16-9 Project Explorer view showing the Database Application project created

Working with the Transformation Parameters Editor

The Transformation Parameters Editor enables you to create and modify parameters for model elements defined in an external target model, storing the transformation parameters in a .tpm file. You can add, remove, enable, and disable transformations that are applicable for particular model elements, as well as modify transformation properties for a particular element.

Changing the z/OS Database Application Transformation Editor layout

The editor displays a list of objects in the model. You can display the list in horizontal or vertical form using the icons, as shown in the Figure 16-10.

If required, you can also add, remove, or change the order of selected transformations by clicking the Manage Applied Transformations icon.

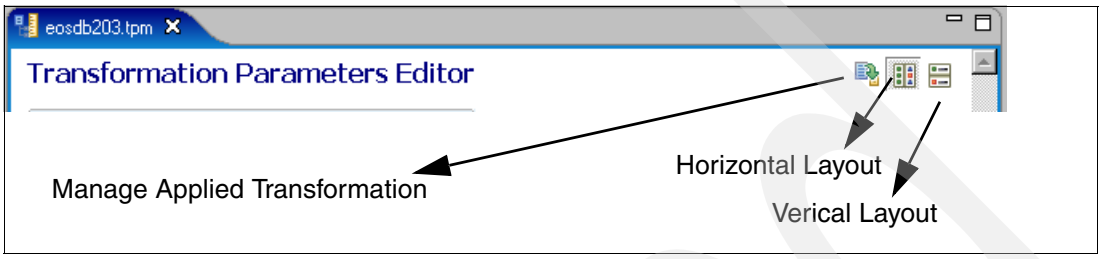


Figure 16-10 Icons to change the editor

If you need to remove an object from the transformation list, right-click and select **z/OS Data Access**. This step cannot be reversed. This applies when you have multiple transformations, but in our example the only possible transformation would be z/OS Data Access Transformation.

Configuring the z/OS DB2 Tables Application

To select a project to view and edit the transformation parameters, expand the objects and click the COBOLTYPES icon, as shown in Figure 16-11, to see and configure the z/OS Database Application Transformation. Note that you must click the icon that is located after the name COBOLTYPES.

You can use the defaults or change the fields as you desire. In our example, we use the default values.

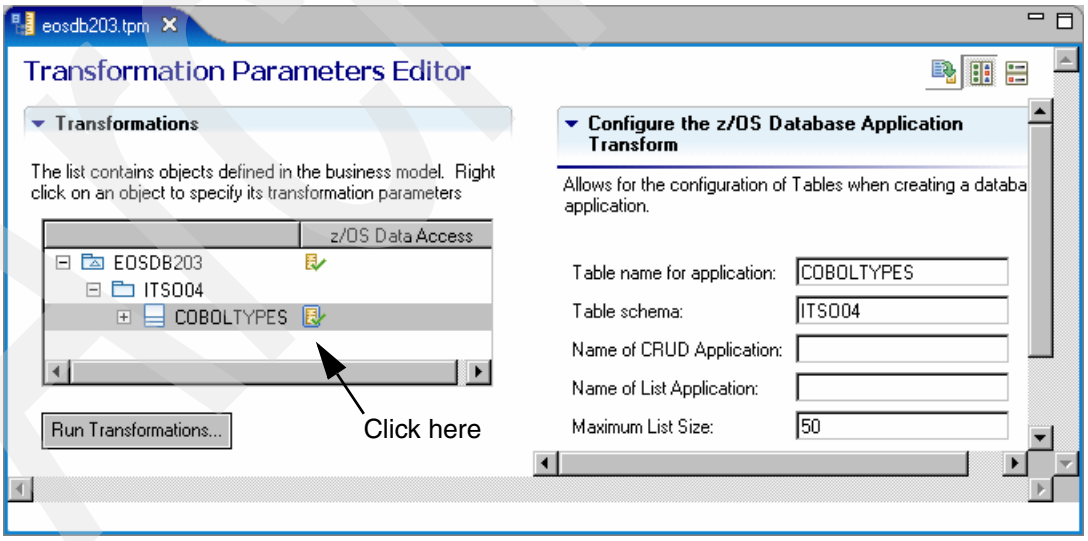


Figure 16-11 Configuring the Database Application Transformation

If you do not specify names for the COBOL applications, the default names will be XXXXCRUD for the Create, Read, Update, and Delete application, and XXXXLIST for the List application, where XXXX are the first four characters of the table name. In our example, because the table name is COBOLTYPES, those names will be COBOCRUD and COBOLIST.

Maximum List Size is the maximum number of DB2 rows that will be listed when the application xxxxLIST is executed. The default value is 50.

Configuring the DB2 Configuration columns

You can optionally select a column to view and edit the transformation parameters. In our example, we changed the Column name for the application from ID to CODE, but we kept all the other values the same, as shown in Figure 16-12.

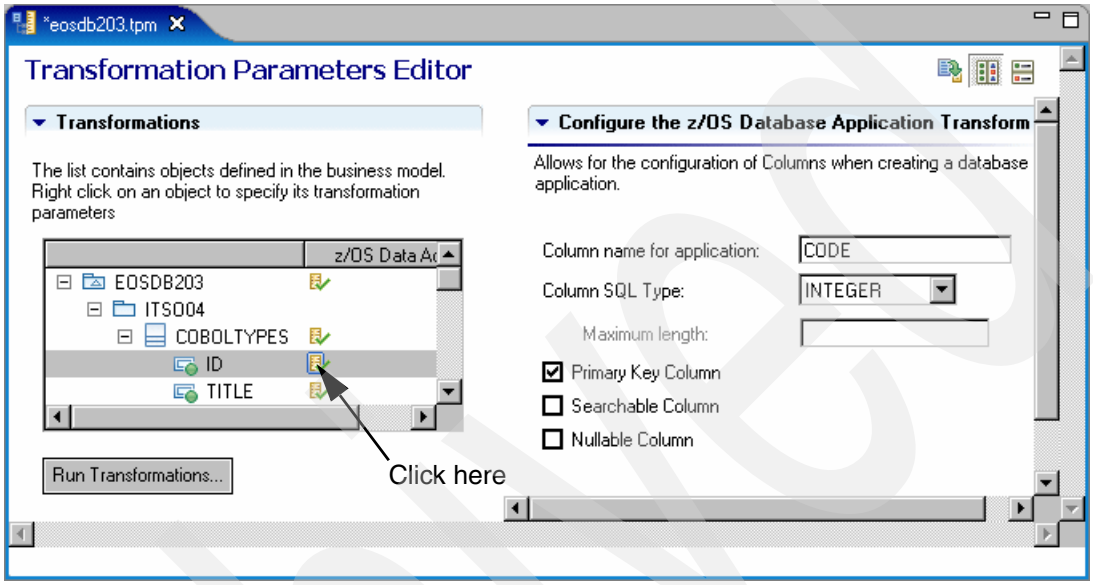


Figure 16-12 Configuring the DB2 Column

The values are self explanatory, and you might need to change some values, as shown below:

Searchable Column Specify whether this column should be searchable. If you specify a column as searchable, the wizard will make that column part of the interface to the “LIST” program that is generated, so that it can be used in the search. We did not use this in our example, but it is desirable to have this specified.

Nullable Column Specify whether this column can accept null values. This could match your database definitions.

To make it simpler, we made no changes, and just used the default values.

Configuring the JCL job parameters

The wizard will create a JCL to deploy the generated assets. You might need to change the defaults to be generated.

To configure the JCL job parameters, click the EOSDB203 icon and click the **Configure JOB Card** link under Project Settings. You could modify “%U%”, which would represent the User value specified in the CICS settings. This value can also be modified in the Preferences window. We made no changes in our example.

Configuring the CICS settings

In the window, the asterisk indicates parameters that are required for automatic deployment in order to generate artifacts.The only change that we made in our example was the URI Prefix:

```
http://zserveros.dfw.ibm.com:3091/u/itso04/wdzv7pot/wspickup/provider/
```

This parameter specifies the Uniform Resource Identifier namespace and the associated prefix that will be used by the generated WSDL to invoke the CICS Service. Note also that u/ITSO04/wdzv7pot/wspickup/provider is where the WSBIND file will be moved to, because this is the pickup directory that was defined for our CICS. Also, the port that we are using is 3091. You must verify these values in your installation. Figure 16-13 shows our example.

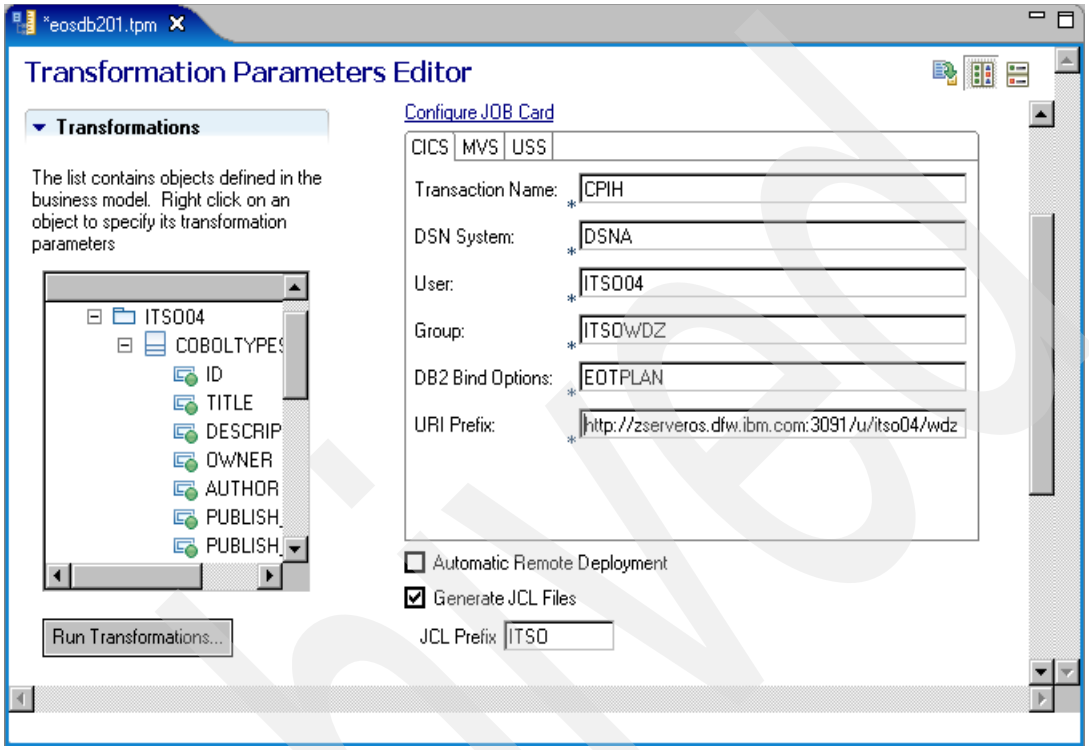


Figure 16-13 Configuring CICS settings

Most of the values are self explanatory, and you can see more details using the WD/z Help menu. These values are:

- The *transaction name* for the default pipeline alias transaction ID used for inbound HTTP Web service requests is CPIH. If you change this transaction, you need to create another transaction definition similar to the already defined CPIH, as well customize the pipeline configuration file; this is not a trivial task.
- *DB2 Bind Options* (EOTPLAN) will be the name of the DB2 Bind Plan and package that are created within the JCL definitions. This plan will be associated with the transaction CPIH that is the default transaction provided for CICS Web services.
- *JCL Prefix* is the prefix name of the generated JCL. In our example, the generated JCL will be named ITSOxxx.JCL. If you do not specify the JCL Prefix, the wizard uses the first four characters of the connection name. In our example, the connection name is EOSDB203, and if a JCL prefix was not specified, the JCL generated would be EOSDxxxx.JCL.
- *User* is the value that the wizard will pick for the job card. It will replace the %U% that is specified in the Configure JOB Card link or in the z/OS Database Application Preferences. In our example, we use ITSO04, so the generated JOB name will be //ITSO04C JOB.

Configuring the MVS settings

This is the z/OS system that compiles the generated COBOL code and runs the programs.

If the Automatic Remote Deployment option is in effect and you are connected to the z/OS, the files generated are moved to the remote system when the transformation runs. Also, if the Generate JCL Files option is enabled, the JCL files are submitted automatically.

Note also that if Automatic Remote Deployment is checked you must specify information for all the required fields in order to deploy the assets on z/OS. Because we have already defined those data sets in the workspace preferences, the data set names will be filled when this dialog is opened.

If Automatic Remote Deployment is not selected, those values are not required and the assets will be generated in the workspace. That is our choice in this example.

Figure 16-14 shows the values that we have used.

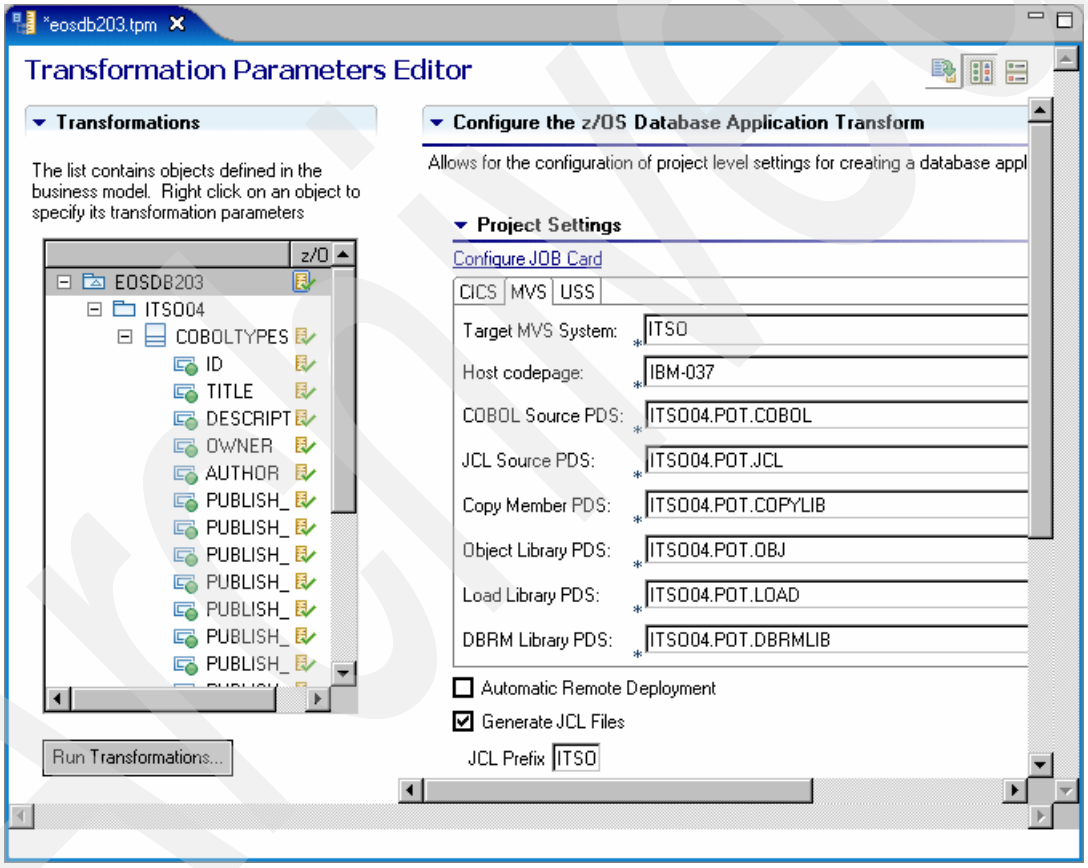


Figure 16-14 Configuring MVS settings

Configuring the UNIX System Services settings

When you create CICS Services, you need to work with UNIX System Services. For example, the WSBIND fine used by CICS must be located in one hierarchical file system (HFS) directory.

If you have Automatic Remote Deployment checked, you must specify information for all the required fields in order to deploy the assets on z/OS.

Figure 16-15 shows the UNIX System Services settings used in our projects; we did not specify any value, because we did not use the Automatic Remote Deployment.

When you finish with the Transformation Parameter Editor, save your changes by pressing Ctrl+S.

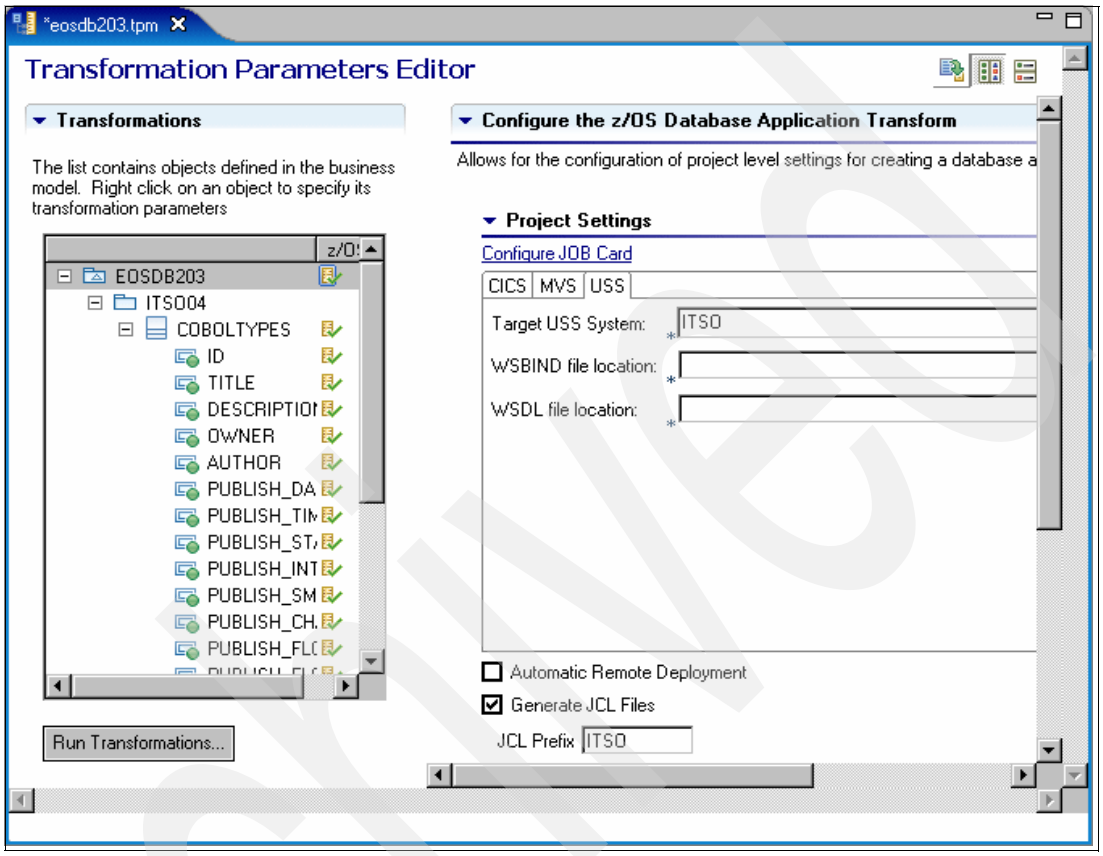


Figure 16-15 Configuring UNIX System Services settings

Running the transformation

After saving the editor, click **Run Transformations...**.

In the Specify configuration properties window, select the target container and the desired elements. Click **Finish** to run the transformations, as shown in Figure 16-16 on page 581.

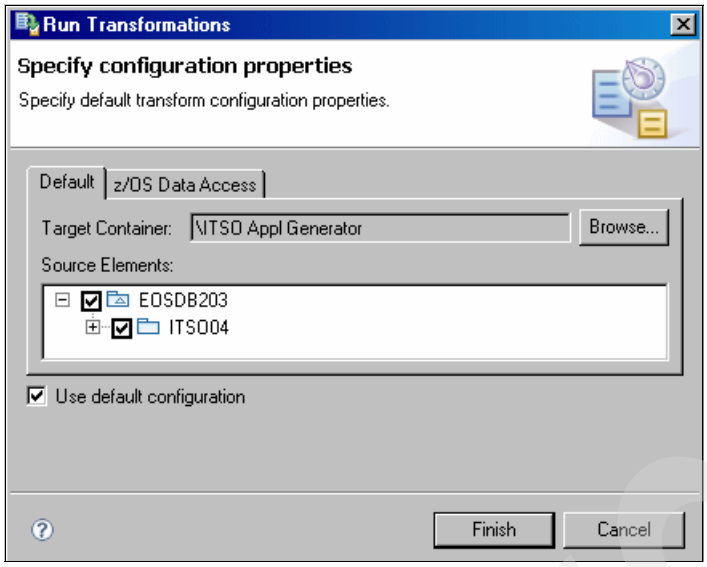


Figure 16-16 Specifying configuration properties

Because we did not specify the UNIX System Services information, you might have a warning message that will inform you that one or more deployment options are missing and that some deployment artifacts will not be generated. You can ignore and dismiss this warning.

The transformation output files are generated to the local project. You can copy them over to the host machine.

Archived

Our example is listed in Figure 16-17.

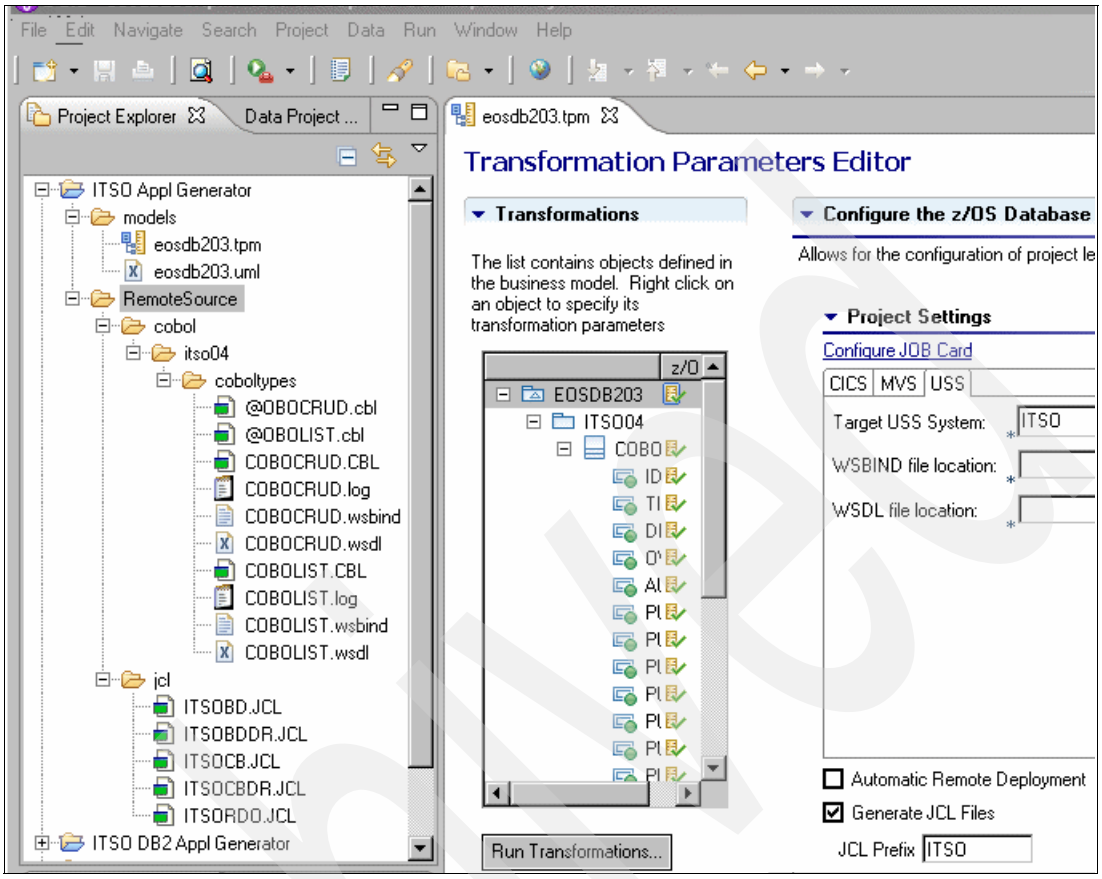


Figure 16-17 Generated assets

Brief descriptions of the generated assets are listed below:

itso04 The schema name. We have used our user ID as the default (itso04).

coboltypes The table name.

@OBOCRUD.cbl CRUD Web Services for CICS TS 3.x Converter Driver

@OBOLIST.cbl LIST Web Services for CICS TS 3.x Converter Driver

COBOCRUD.cbl The program that performs the Create, Read, Update, and Delete operations against the DB2 table.

COBOCRUD.log This is a log file for the CRUD application. It logs the date, WD/z version used, parameters used for the transformation, and so on. This file must be given to IBM in case support is needed. It also has the hexadecimal content of the WSBIND file generated.

COBOCRUD.wsbind This is the WSBIND required by CICS to create the Web Service for the CRUD application. This file will need to be installed in the z/OS system in the pickup directory.

COBOCRUD.wsdl This the Web Service Definition Language (WSDL) file that is needed later for testing the CRUD Web service created. This file does not need to be moved to the z/OS system, since the XML parsing is done by the generated COBOL driver.

COBOLIST.cbl The program that performs the read of multiple rows from the table. In our example, we used the default, which is 50 rows.

COBOLIST.log This is a log file for the LIST application, which contains contents similar to the CRUD application.

COBOLIST.wsbind This is the WSBIND file required by CICS to create the Web service for the LIST application.

COBOLIST. wsdl This is the WSDL that defines the LIST Web service application. It will be used later for testing the implemented assets.

A brief description of the generated JCLs are listed below. Note that the JCLs generated will use the z/OS system connection properties; if your connection properties are not correctly configured, you will need to correct the generated JCLs.

ITSOBD.JCL A JCL used to build the generated CRUD and LIST COBOL Application programs and bind the DB2 plans and packages. This JCL will invoke the proc ITSOCB.

ITSOBDDR.JCL A JCL used to build the CRUD and LIST COBOL generated driver programs. This JCL will invoke the proc ITSOCBDR.

ITSOCB.JCL A procedure used by ITSOBD.JCL to:

– DB2 pre-compile or pre-process COBOL programs with EXEC SQL statements

– CICS pre-compile COBOL programs with CICS statements

– COBOL compile

– Linkedit compiled programs

ITSOCBDR.JCL A procedure used by ITSOBDDR.JCL to perform the COBOL/CICS operations for the generated driver program.

ITSORDO.JCL A JCL to define CICS resources through the Resource Definition Option (RDO). The RDO Resource Definitions:

– Define program resources for table COBOLTYPES

– Define the resources to access DB2

– Define the Web Services resources for table COBOLTYPES

If you select Automatic Remote Deployment, then the following actions are performed:

- All .cbl files are moved to the specified target COBOL data set on the MVS system.
- All .jcl files are moved to the specified target JCL data set on the MVS system.
- The JCL is submitted to the host.
- All .wsbind files are moved to the specified WSBIND directory on the UNIX System Services system.
- All .wsdl files are moved to the specified WSDL directory on the UNIX System Services system.

Bidirectional considerations

When working with transformations, the Database Application Generator enables you to specify bidirectional options at the project, table, and column levels. You can also set options for the host format and Web Services. For example, for Arabic, select the **IBM-420** code page, For Hebrew, select the **IBM-424** code page. See the WD/z Help menu for more details.

Archived

16.5.3 Understanding the generated assets

For the each DB2 table, the Database Application Generator creates one CRUD and one List program. If all deployment information is provided, it also generates the COBOL Driver for each of the programs created, the WSBIND, and 5 JCLs, as shown in Figure 16-18.

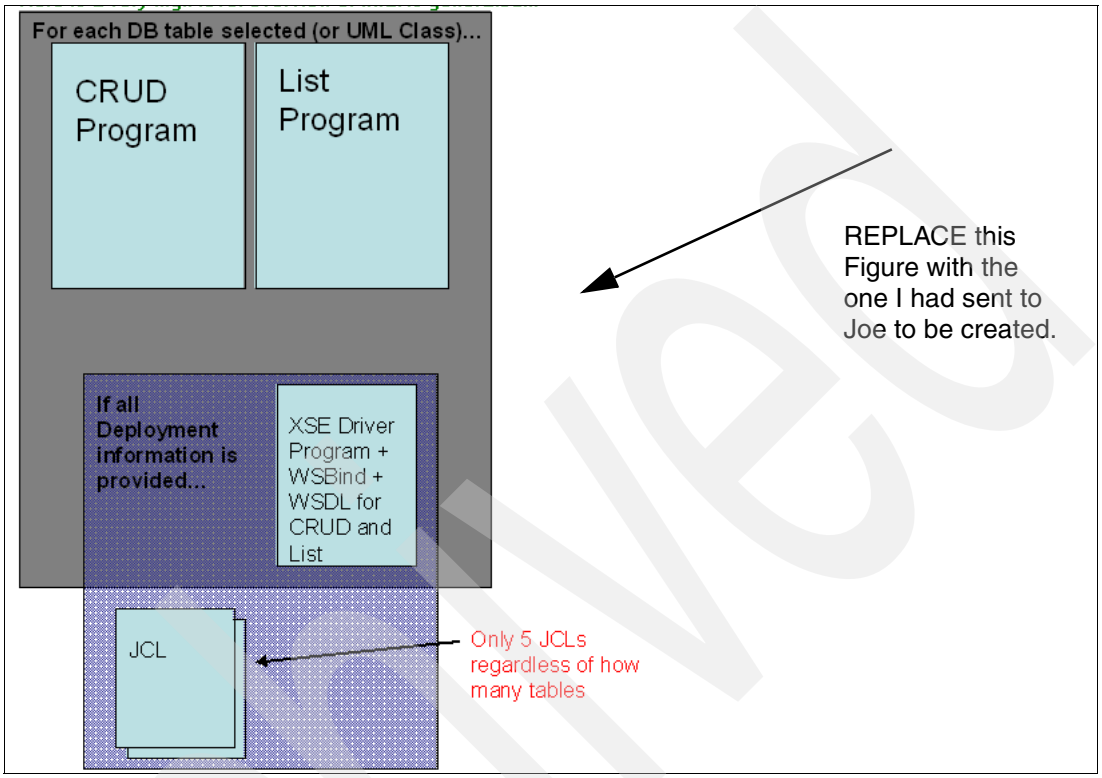


Figure 16-18 Overview of the generated code

Understanding the COBOL driver programs

The wizard generates two files starting with @ that we call COBOL drivers. In our scenario, they are @OBOCRUD.cbl and @OBOLIST.cbl. The driver programs are responsible for the inbound and outbound communication with the service invocation. Also, the driver is responsible for XML parsing. These programs do not touch DB2 and they should never be modified.

Note that these files have more than one physical COBOL program, but each file will be responsible for the creation of one load module. For example, @OBOCRUD.cbl file has the 12 programs shown in the Table 16-1. but these programs create the load module @OBOCRUD. The same architecture applies to the @OBOLIST.cbl driver.

Table 16-1 Example of @OBOCRUD driver programs

COBOL program name	Remarks
@OBOCRUD	Defines interfaces, calls @OBOCRUX, @OBOCRUI, and @OBOCRUO. Uses @OBOCRUF, @OBOCRUX
@OBOCRUF	Exception handler.
@OBOCRUX	Converter meta data aggregate service

Archived

COBOL program name	Remarks
@OBOCRUI	Inbound XML converter. Calls @OBOCRUJ. Executes the XML parse for the input XML. Uses @OBOCRUA.
@OBOCRUJ	Maximum inbound language structure size In bytes.
@OBOCRUA	Inbound XML converter exception handler.
@OBOCRUO	Outbound XML converter. Calls @OBOCRUC, @OBOCRUL, and @OBOCRUK. Uses @OBOCRUE.
@OBOCRUC	Outbound XML 1.0 message builder. Calls XCHRFLTR.
XCHRFLTR	Outbound illegal XML 1.0 character filter.
@OBOCRUK	Maximum outbound XML message size in bytes.
@OBOCRUL	Maximum outbound language structure size in bytes.
@OBOCRUE	Outbound XML converter exception handler.

Note that a JCL procedure named ITSOCBDR.JCL is created to assemble those programs, and a JCL named ITSOBDDR.JCL, which uses this procedure, is also created, and is responsible for building the driver load module @OBOCRUD.

Understanding the COBOL program that performs the CRUD operation

The generated COBOL programs COBOCRUD.CBL and COBOLIST.CBL are the programs that will access the DB2 table and perform the CRUD and List operations. Each of these programs will create a load module named COBOCRUD and COBOLIST.

The program COBOCRUD does the Create, Read, Update, and Delete operations. This program is loaded by the CICS when the Web service is executed. If you do not want to implement the CICS Web Services, you might use those programs as a model to access the DB2 table.

The possible operations will be C (create), R (Read), U (Update), and D (Delete). Figure 16-19 shows part of the working storage of this program with those definitions.

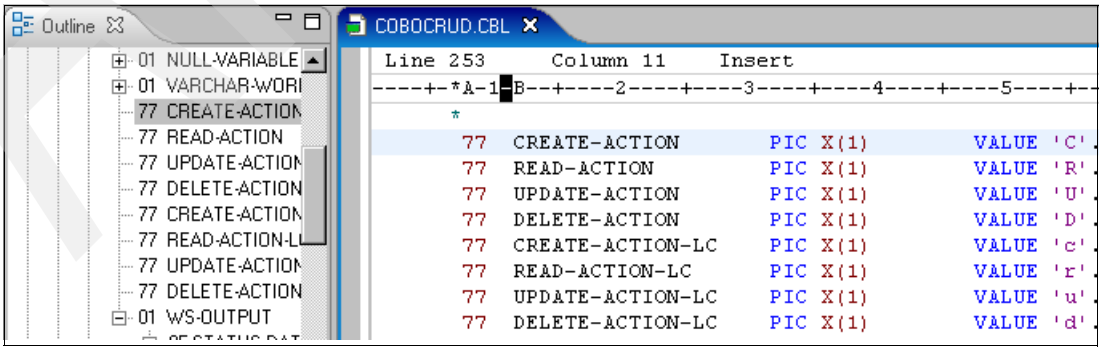


Figure 16-19 Actions defined for program COBOCRUD.cbl

This program is built using the CHANNEL program interface instead of COMMAREA, which allows you to use more than the 32K memory limit. Figure 16-20 shows part of the LINKAGE SECTION in the Input area.

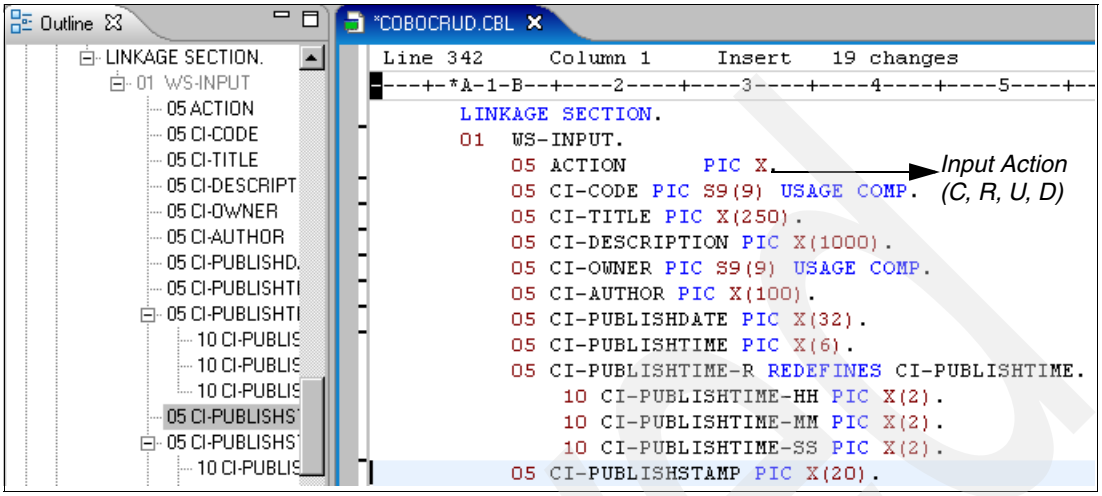


Figure 16-20 Input area being used by COBOCRUD program

Figure 16-21 shows part of the Output area used by COBOCRUD and defined in the Working area.

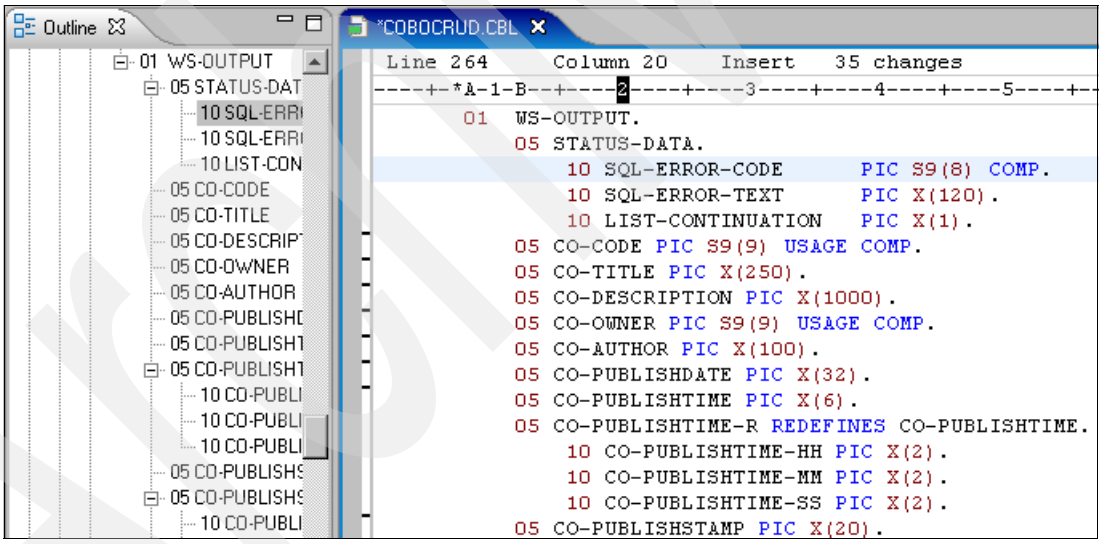


Figure 16-21 Output data used by COBOCRUD.CBL

The procedure division would handle the various CRUD operations. Figure 16-22 on page 587 shows the generated code for the Delete operation.

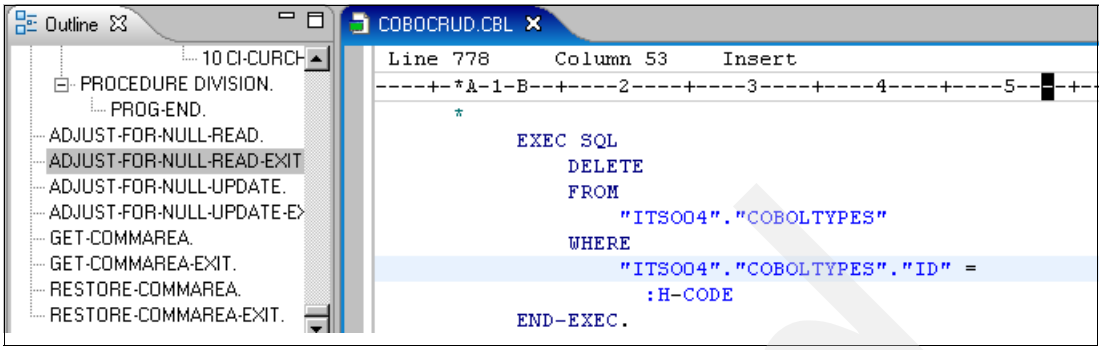


Figure 16-22 Procedure Division statements for the Delete operation

Understanding the COBOL program that performs the list operation

The COBOLIST program does the List operation. This program is loaded by the CICS when the Web service is executed. Again, if you do not want to implement the CICS Web Services, you might use those programs as a model to access the DB2 table.

The possible parameters that this program receives are:

- Where to start in the list, and the first DB2 row to be listed
- How many rows to retrieve. The limit in our example is the default, that is, 50 rows.
- Fields to search, that is, customized by the TPM. We used the default, but we could have done the search by TITLE, for example.

Figure 16-23 shows the LINKAGE SECTION of this program with the input parameters.

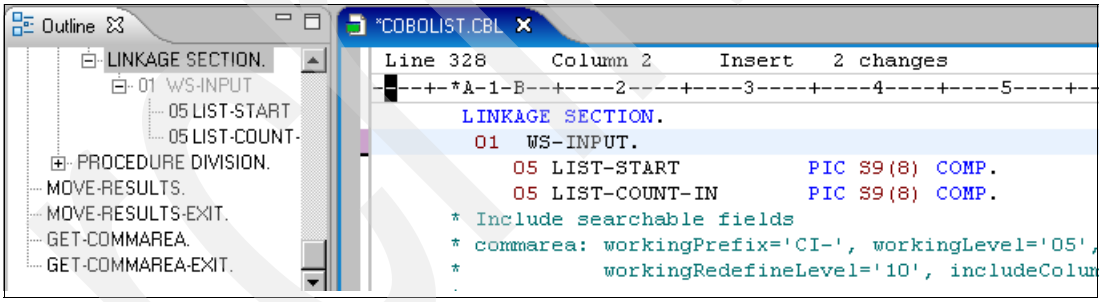


Figure 16-23 Input area used by COBOLIST program

Figure 16-24 shows the listing output to be displayed. Note that LIST-CONTINUATION is the field that tells if there is more rows remaining and LIST-COUNT-OUT shows how many rows are returned.

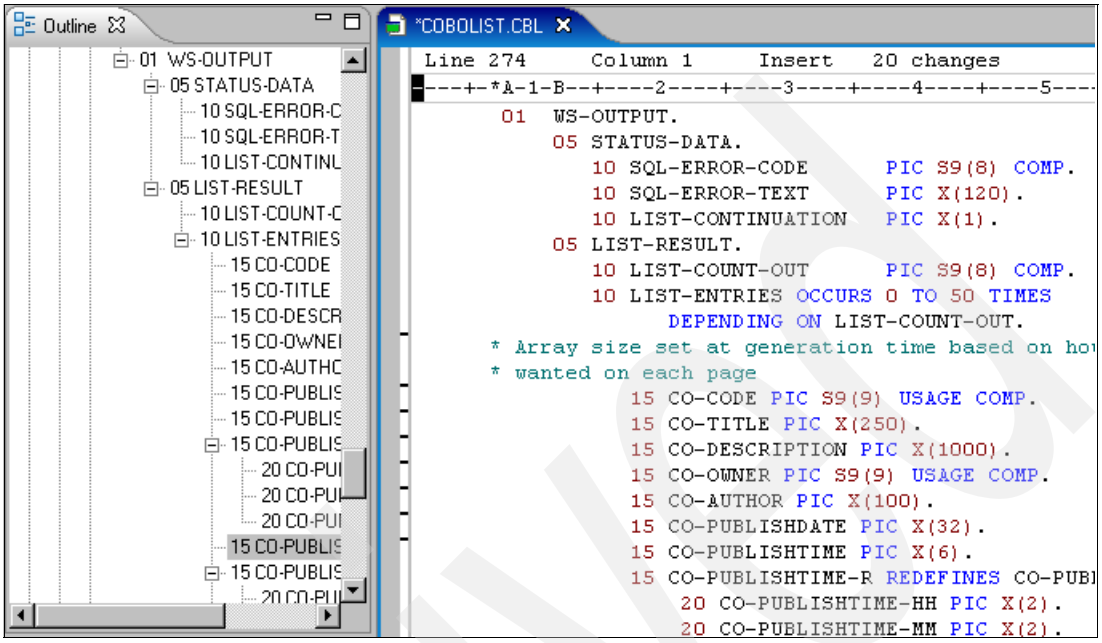


Figure 16-24 Output area used by COBOLIST program

Figure 16-25 on page 589 shows the DB2 SQL statements used to declare the cursor to later fetch the rows. Note that the table qualifier was one of the parameters that we used in the TPM editor; in our example, it was ITSO04.

Archive

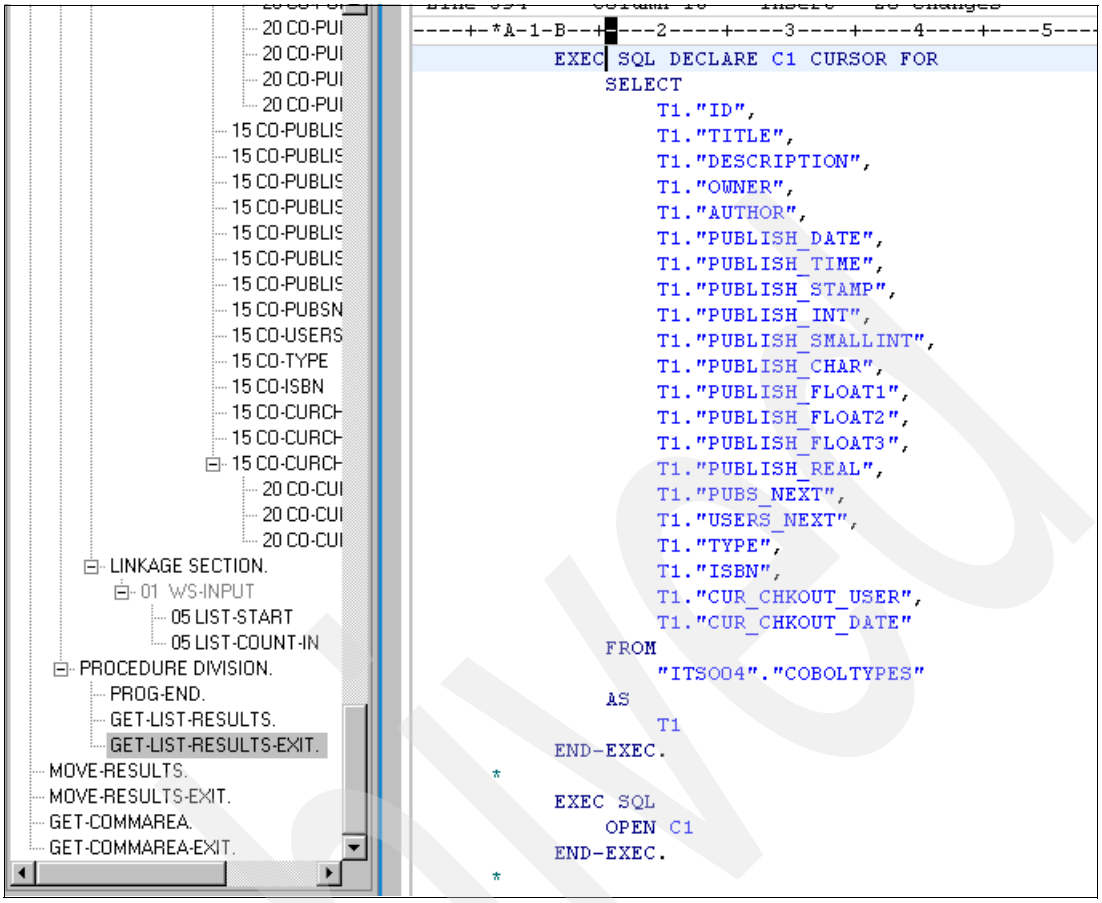


Figure 16-25 SQL DECLARE used by program COBOLIST

16.5.4 Working with the generated JCL

Even if the JCL generated is complete, you might want to verify that it matches your z/OS definitions to avoid surprises and mistakes.

You can customize the JCL templates using the Java Enabler Templates (JET). For more information about this topic, refer to the article “Create more -- better -- code in Eclipse with JET”, found at

http://www-128.ibm.com/developerworks/opensource/library/os-ecl-jet/

In our example, the JCLs used for generation are defined in the elements cobcl.jet and cobcldvr.jet that are inside of the JAR file com.ibm.etools.wdz.uml.transform.templates_1.0.0.v200612041349.jar, which is located in the directory C:\Program Files\IBM\SDP70Shared\plugins.

You may try to change the JCLs manually at your own risk, but future updates can damage your changes. For more details, refer to the JET documentation.

Archived

Adapt the generated JCL procedures for your z/OS system

The two JCL procedures that are generated must be changed, because they are using some data sets names that you probably do not have, and if you do not change them, you will have JCL errors. The parameters that you might need to modify in the procs are:

DB2LOAD	DB2 DSNLOAD data set
CICSLOAD	CICS SDFHLOAD data set
TRANSYSL	COPYLIB for the CICS translator
LANGLIB	COBOL compiler library
LNKLIB	LINKLIB data set

Example 16-3 shows the changes that we made in our example for the proc ITSOCB. You also need to make changes on the proc ITSOCBDR.JCL.

Example 16-3 Changes made in the proc ITSOCB.JCL

```
//ITSOCB  PROC COBDSN=ITSO04.POT.COBOL,
// COBMBR=DUMMY,
// DB2LOAD=DB2.V8R1.SDSNLOAD,
// CICSLOAD=CICSTS.V3R1.CICS.SDFHLOAD,
// DBRMDSN=ITSO04.POT.DBRMLIB,
// TRANSYSL=CICSTS.V3R1.CICS.SDFHCOB,
// LANGLIB=COBOL.V3R4.SIGYCOMP,
// OBJDSN=ITSO04.POT.OBJ,
// COPYLIB=ITSO04.POT.COPYLIB,
// LOADDSN=ITSO04.POT.LOAD,
// LNKLIB=CEE.SCEELKED
...
```

Change the JOB that does the DB2 BIND

You might need to check the JOB named ITSOBD.JCL that does the DB2 BIND, and you also might need to customize the DB2 data set name. Also, you might need to add other packages that you might need to bind with this plan. In Example 16-4, we customized our job by changing the DB2 library and adding the package “WDZPLAN” to be bound to EOTPLAN, as well as eliminating the QUALIFIER keyword. The changes depend on your installation.

Example 16-4 Part of the ITSOBD.JCL that does the DB2 BIND

```
//BIND EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//DBRMLIB  DD  DSN=ITSO04.POT.DBRMLIB,DISP=SHR
//SYSTSIN DD *
 TSOLIB ACTIVATE DA('DB2.V8R1.SDSNLOAD')
 ALLOC FI(DBRMLIB) SHR DA('ITSO04.POT.DBRMLIB')
 DSN SYSTEM(DSNC)
....
BIND PLAN(EOTPLAN) -
      PKLIST(EOTPLAN.*, WDZPLAN.*) -
      ACT(REP) -
      RETAIN -
      VALIDATE(RUN) -
      ISOLATION(CS) -
     -QUALIFIER(EOTPLAN) -
      OWNER(ITSO04) -
      ENCODING(037) -
```

```
      ENABLE(CICS)
/*
//
```

Customize the JCL used by the RDO definitions

The JCL ITSORDO.JCL generated must be customized to match your CICS and DFHCSD data set. Example 16-5 shows the changes that we made in our example to match our z/OS system definition.

Example 16-5 Changes made to the job ITSORDO.JCL

```
//DEFINE   EXEC PGM=DFHCSDUP,REGION=1024K,PARM='CSD(READWRITE)'
//STEPLIB  DD DSN=CICSTS.V3R1.CICS.SDFHLOAD,DISP=SHR
//         DD DSN=CICSTS.V3R1.CICS.SDFHAUTH,DISP=SHR
//DFHCSD   DD DSN=CICSTS.CICSAOR1.DFHCSD,DISP=SHR
//SYSPRINT DD SYSOUT=*
```

16.5.5 Deploying the generated assets to z/OS

During the wizard execution, we created the assets in the WD/z workspace, not in the z/OS system, because we did not select the Automatic Remote Deployment. We now must move all required assets to the z/OS system and submit the generated JCL for execution.

Copying the COBOL programs to the z/OS

In our example, the generated COBOL programs must be copied to the data set ITSO04.POT.COBOL, because we did not use the Automatic Remote Deployment option. For the JOB submissions, the JCL can stay in the workspace or be copied to the z/OS, based on your preferences. You do not need to have the JCLs on the z/OS system to submit them for execution, but you need to have the COBOL programs there.

To copy the COBOL programs, do these steps:

1. Using the Remote Systems view, be sure that you are connected to the z/OS system.
2. Using the Remote Explorer view, select all generated COBOL programs (**@OBOCRUD.cbl**, **@OBOLST.cbl**, **COBOCRUD.cbl**, and **COBOLIST.cbl)**, right-click, and click **Copy**.

Archived

3. Switch to the Remote System view, expand **MVS Files**, right-click **ITSO04.POT.COBOL**, and click **Paste**. All the programs will be copied to the z/OS, as shown in Figure 16-26.

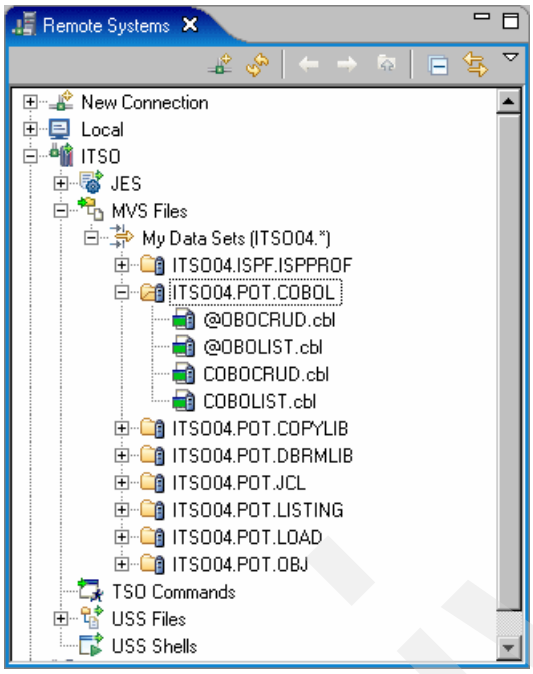


Figure 16-26 COBOL programs copied in the z/OS system

Copying the JCL generated procedures to the z/OS

As we have explained before, it is your decision whether or not to move the generated JCL to the z/OS system, but you must at least copy the JCL procedures that will be executed when the JCL is submitted. The two JCL procs that you must move are ITSOCB.JCL and ITSOCBDR.JCL, which you can do by doing these steps:

1. Using the Remote Explorer view, select **ITSOCB.JCL** and **ITSOCBDR.JCL**, right-click, and click **Copy**.
2. Switch to the Remote System view, expand **MVS Files**, right-click **ITSO04.POT.JCL**, and click **Paste**. All the programs will be copied to the z/OS, as shown in Figure 16-27.

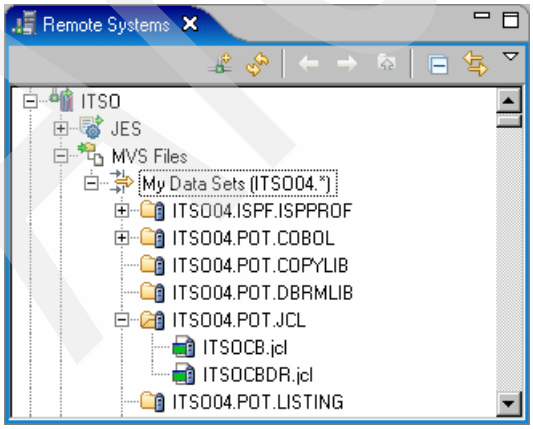


Figure 16-27 JCLs copied to the z/OS system

Archived

Submitting the JCL to create the executable

The easiest way to submit a JCL that is in your local workstation is by editing it and typing **`submit`** in the command line. In our example, we typed **`submit to ITSO`** using the editor's command line. The message `JOBID: JOBnnnnn` informs you that the job has been submitted.

In our example, we must submit the JCLs ITSOBD.JCL (CRUD and LIST Application) and ITSOBDDR.JCL (Driver). You can use the JES subsystem to check the results.

Figure 16-28 shows the result of the JOB ITSOBD.JCL execution that compile, link, and DB2 Bind the COBOL of the CRUD and LIST applications. Note that the return codes cannot be higher than 4.

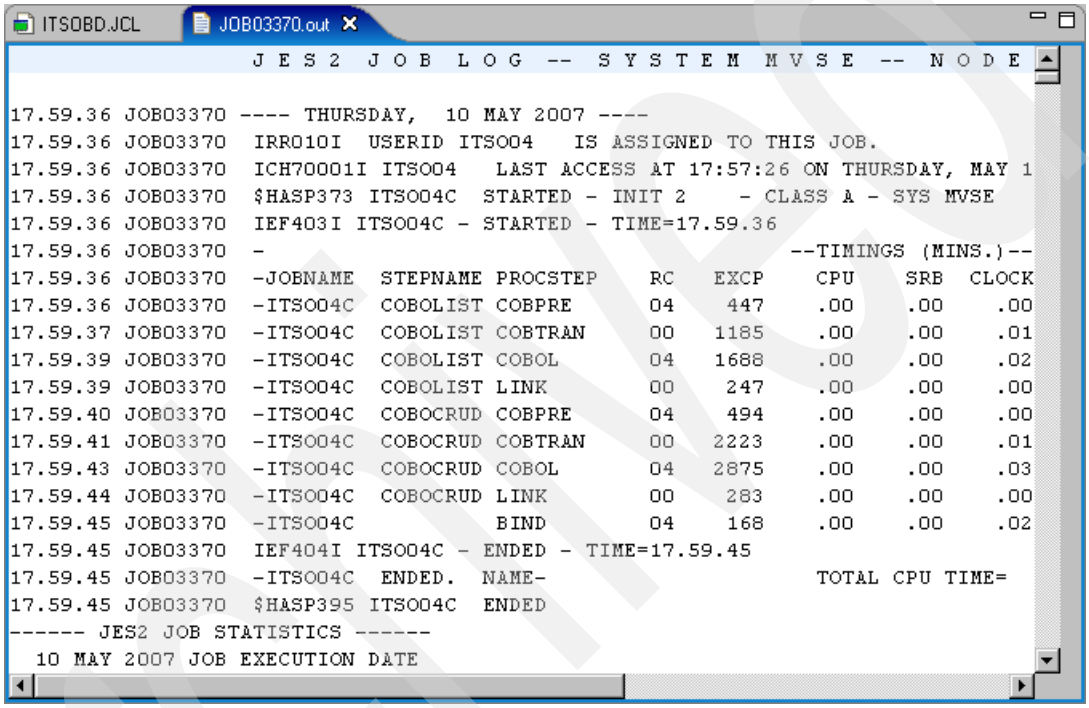


Figure 16-28 Compile, link, and BIND of COBOLIST and COBOCRUD applications

Archive

Figure 16-29 shows the result of the JOB ITSOBDDR.JCL execution that compiles and links the COBOL Driver for the CRUD and LIST applications. Note that the return codes cannot be higher than 4.

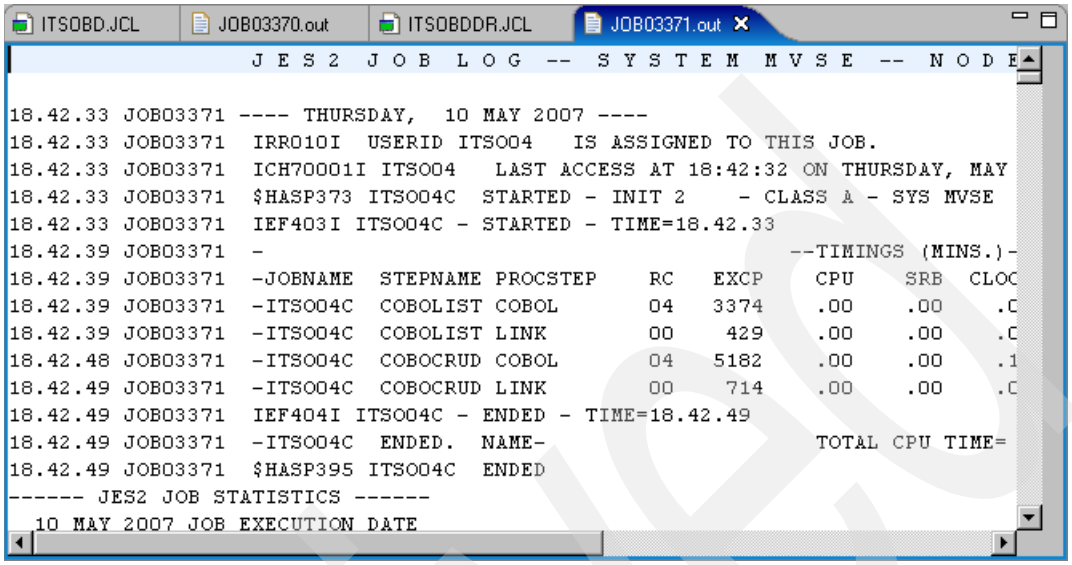


Figure 16-29 Compile and link the COBOL driver for the CRUD and LIST applications

Figure 16-30 shows the result of the JOB ITSORDO.JCL definition of the CICS resources for the CRUD and LIST applications. Note that the return codes cannot be higher than 4.

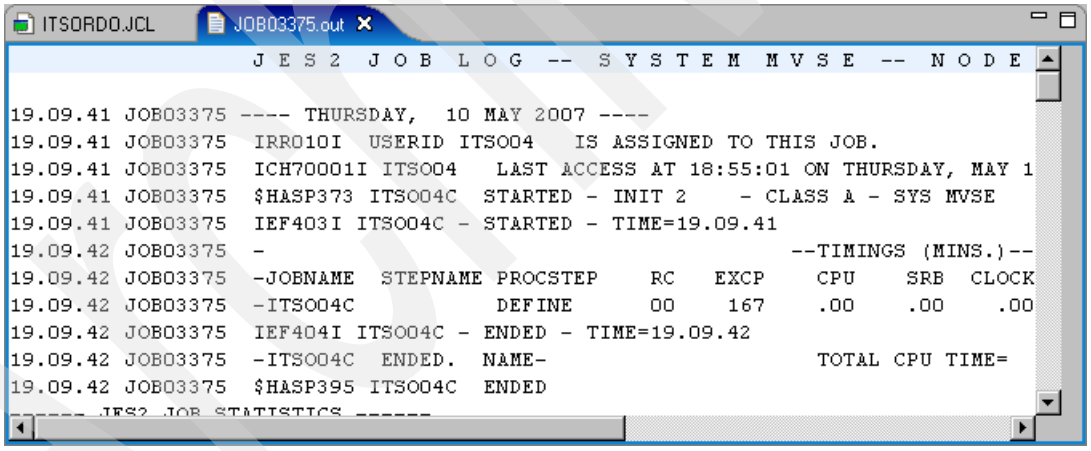


Figure 16-30 CICS RDO execution

16.5.6 Configuring CICS Web Services on z/OS

The Database Application Generator Wizard is very helpful in creating assets, but you still need to create the CICS definitions to implement the CICS Services that execute this generated application.

We briefly show you how we created the CICS Web Services definitions in our example.

Creating the z/OS HFS directories

To expose a CICS program as a Web service over HTTP, TCPIPService and PIPELINE resources are required. The PIPELINE resource definition points to some Hierarchical File System (HFS) directories and also points to a pipeline configuration file.

The CICS Services setup requires configuring files and directories in the Hierarchical File System (HFS).

In our example we have created the following HFS directories:

/u/itso04/wdzv7pot/shelf/

The shelf directory is used by CICS to store the Web service binding files that are associated with WEBSERVICE resources. Each WEBSERVICE resource is associated with a PIPELINE. The shelf directory is managed by CICS together with the PIPELINE resource.

Note that several PIPELINEs can use the same shelf directory because CICS ensures a unique directory structure beneath the shelf directory for each PIPELINE. Also, because CICS writes to this directory, CICS requires appropriate security permissions.

/u/itso04/wdzv7pot/wspickup/provider/

The pickup directory contains the Web service binding files that should be associated with the PIPELINE. When a PIPELINE is installed, or in response to a PERFORM PIPELINE SCAN command, CICS searches this directory for files ending in .wsbind. For each .wsbind file, CICS dynamically creates a WEBSERVICE and URIMAP resource, associates it with the PIPELINE, and installs them ready for use.

You can use a WD/z UNIX System Services shell to create those HFS directories. Figure 16-31 shows one example of creating the provider directory. Remember that in our CICS Configuration dialog we specified the URI `http://zserveros.dfw.ibm.com:3091/u/ITSO04/wdzv7pot/wspickup/provider/`.

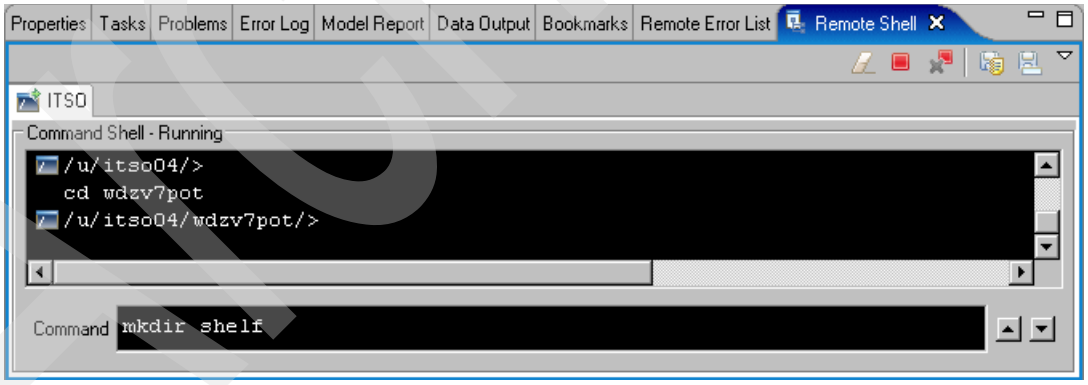


Figure 16-31 Using a WD/z UNIX System Services shell to create HFS directories

Copying the WSBIND file to the z/OS

The WSBIND generated file must be in the directory that the CICS Service will pick up (wsdir) and that was defined in the CICS PIPELINE.

In our example, this will be the HFS directory /u/itso04/wdzv7pot/wspickup/provider/.

The WSBind file is an EBCDIC binary file; it needs to be transmitted as binary to the target z/OS system. You can use the Remote System Explorer facilities of WD/z to copy it to a HFS directory on the target z/OS system. CICS will then use this file during its PIPELINE and Web service generation process.

In order to move the WSBIND file to z/OS, we need to make sure that file extensions of wsbind are handled as binary files. You can check this using the Preferences window (select **Windows** → **Preferences** → **Remote Systems** → **Files**); note that the file type of *.wsbind has, as a default, the Binary File transfer Mode. You do not need to make any changes.

To copy the WSBIND programs, do these steps:

1. Using the Remote Explorer view, select the two WSBIND files (**COBOCRUD.wsbind** and **COBOLIST.wsbind**), right-click, and click **Copy**.
2. Switch to the Remote System view, expand **UNIX System Services Files**, right-click **itso04/wdzv7pot/wspickup/provider/**, and click **Paste**. The two WSBIND files are copied to the z/OS. Figure 16-32 shows these files in the HFS directory.

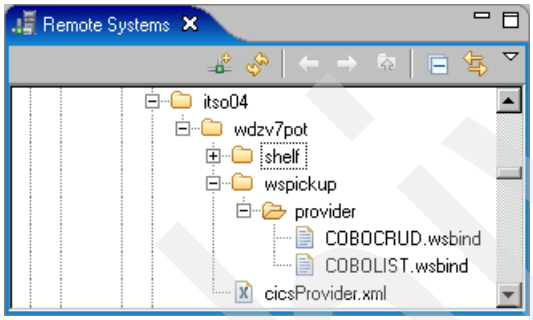


Figure 16-32 WSBIND files copied into HFS directory

Creating a CICS PIPELINE resource definition

A PIPELINE is CICS resource that is used when a CICS application is in the role of a Web service provider or requester. It provides information about the message handler programs that act on a service request and on the response. Typically, a single PIPELINE definition can be used by many applications.

A PIPELINE resource definition contains a reference to the pipeline configuration file, the shelf directory, and the wsdir pickup directory. The configuration file is in XML format and details the message handlers that will act on Web service requests and responses as they pass through the pipeline. We are going to use the CICS supplied PIPELINE configuration file cicsProvider.xml for a SOAP V1.1 handler.

To define the PIPELINE, we used the CEDA transaction CEDA DEFINE PIPELINE(ITSOPIPE) GROUP(ITSOWDZ). See our example in Figure 16-33 on page 597. Note that the HFS directories are case sensitive and must match the directories created before.

Archived

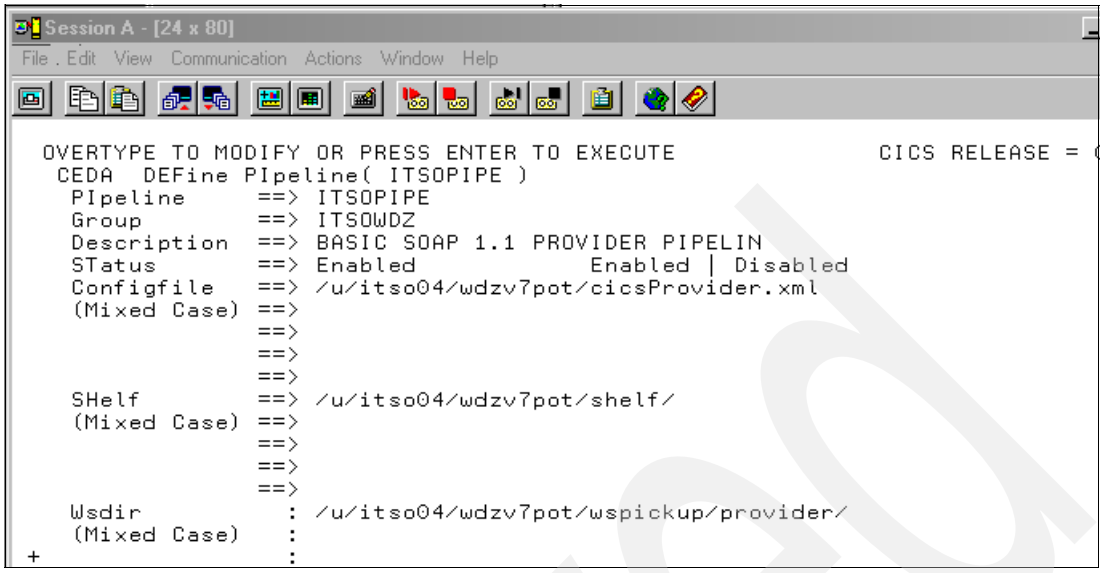


Figure 16-33 CICS CEDA Transaction to define the pipeline

16.5.7 Installing the z/OS CICS services

After the CICS resources are defined, they must be installed.

Using the CEDA CICS transaction, you can see all the resources that we defined in our group **ITSOWDz**, as shown in Figure 16-34 and install them. Note that in our example ITSOPORT and ITSOPIPE were already installed.

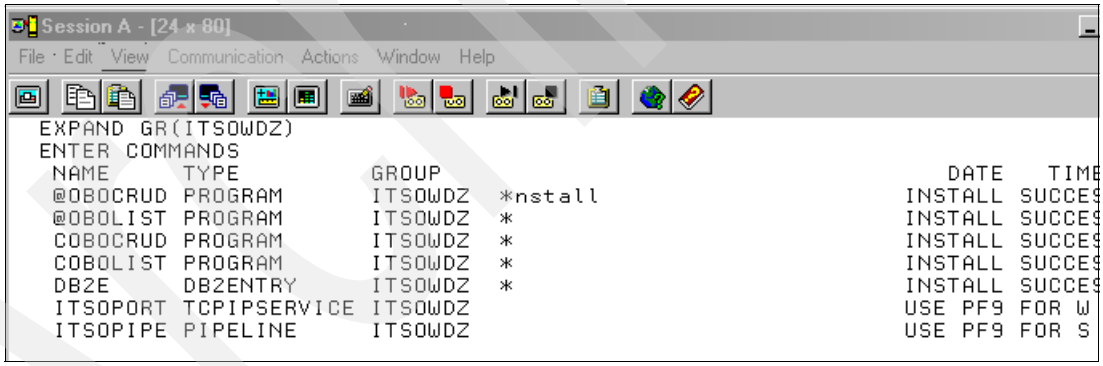


Figure 16-34 Resources defined to CICS in our example

Making your executable available to your CICS

As with any other executable program, before you test your program, you must be sure that your CICS is able to load it. Be sure that the CRUD and LIST load modules are in the loadlib in the CICS library (DFHRPL) concatenation.

In our example, those programs are @OBOCRUD, @OBOLIST, COBOCRUD, and COBOLIST.

Archived

Installing the CICS PIPELINE and TCPIPSERVICE

As with any CICS resources defined, the PIPELINE and the TCPIPSERVICE must be installed in order for them to work; you can do this using the INSTALL option of the CEDA transaction. Figure 16-35 shows one example of this installation, using the command CEDA EXPAND GROUP(ITSOWDZ).

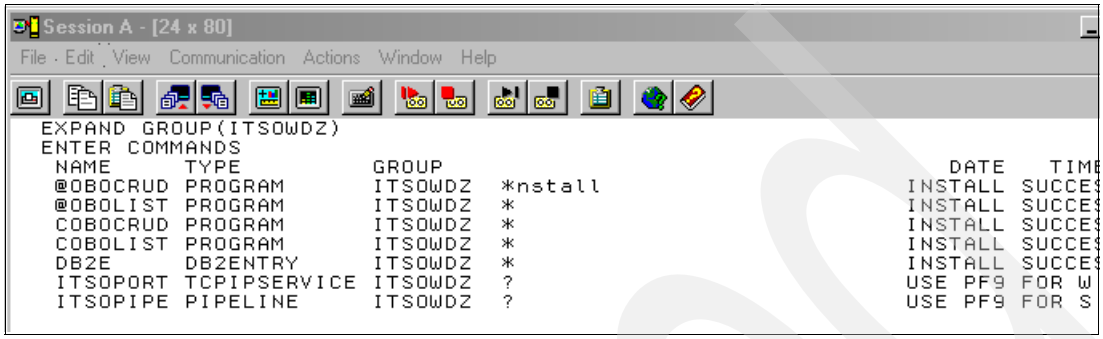


Figure 16-35 Using CEDA INSTALL to install the resources

To be sure that the installation has worked, do an inquiry. Figure 16-36 shows the results of the CEMT inquiry to the installed PIPELINE.

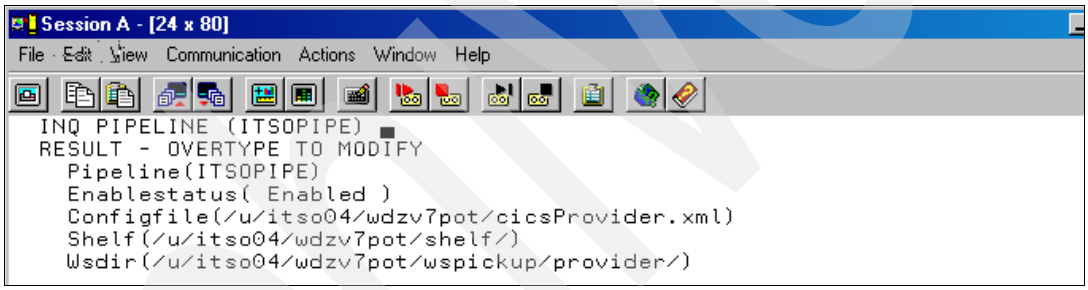


Figure 16-36 CEMT transaction inquiry of the PIPELINE

Creating a CICS TCPIPSERVICE resource definition

The TCPIPSERVICE resource tells CICS to listen on a specified TCP/IP port and accept HTTP requests.

As the client connects to your Web Services over an HTTP transport CICS, you must define a TCPIPSERVICE to receive the inbound HTTP traffic. Usually, the CICS Systems Programmer has to define the PIPELINE with an entry, which we have named TCPIPService, when using this CICS Services mechanism. For example, we used the transaction CEDA DEFINE TCPIPSERVICE (ITSOPORT) GROUP(ITSOWDZ).

Figure 16-37 on page 599 shows our example of the TCP/IP Service creation.

Archive

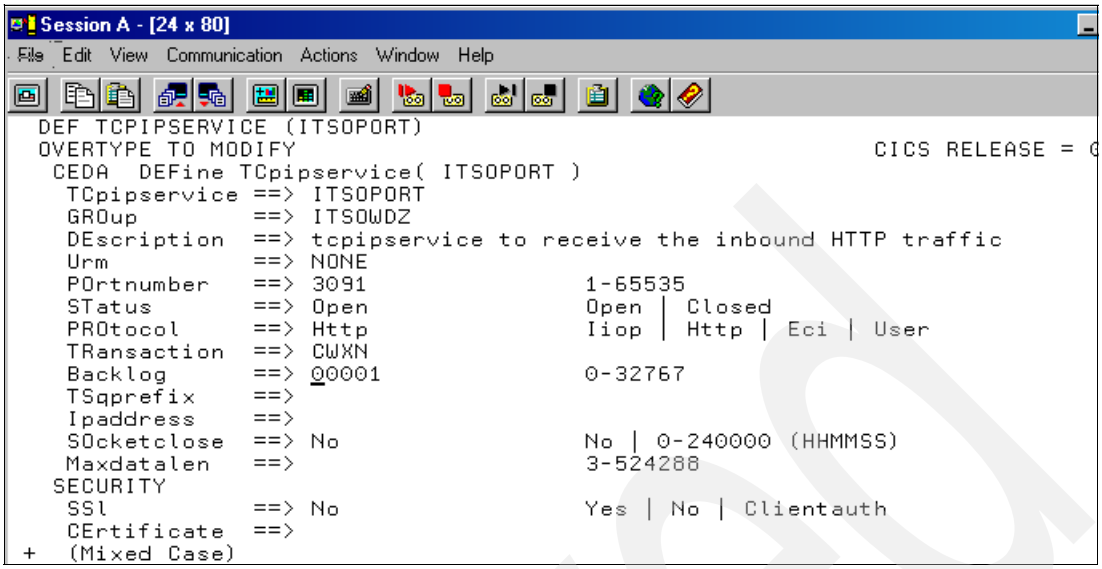


Figure 16-37 CEDA transaction to define the TCPIPSERVICE

Scan the PIPELINE referencing the pickup directory

The PIPELINE resource contains a reference to a pickup directory. In the previous step, you copied the WSBIND files to the pickup provider directory named /u/itso04/wdzv7pot/wspickup/provider/.

When you install each PIPELINE resource, CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory). For each Web service binding file in the directory, which is for each file with the ".wsbind" suffix, CICS installs a WEBSERVICE and a URIMAP, if they do not already exist. Existing resources are replaced if the information in the binding file is newer than the existing resources.

Since we copied a new WSBIND file to the pickup directory, a PIPELINE SCAN is required. The PERFORM PIPELINE SCAN command initiates the scan of the PIPELINE's pickup directory. CICS will read the WSBIND files in that pickup directory and automatically define WEBSERVICE and URIMAP resources associated with those files.

Using a 3270 terminal emulation, execute the CICS transaction CEMT PERFORM PIPELINE(ITSOPIPE) SCAN.

Archived

You should have `RESPONSE: NORMAL` returned, as shown in Figure 16-38.

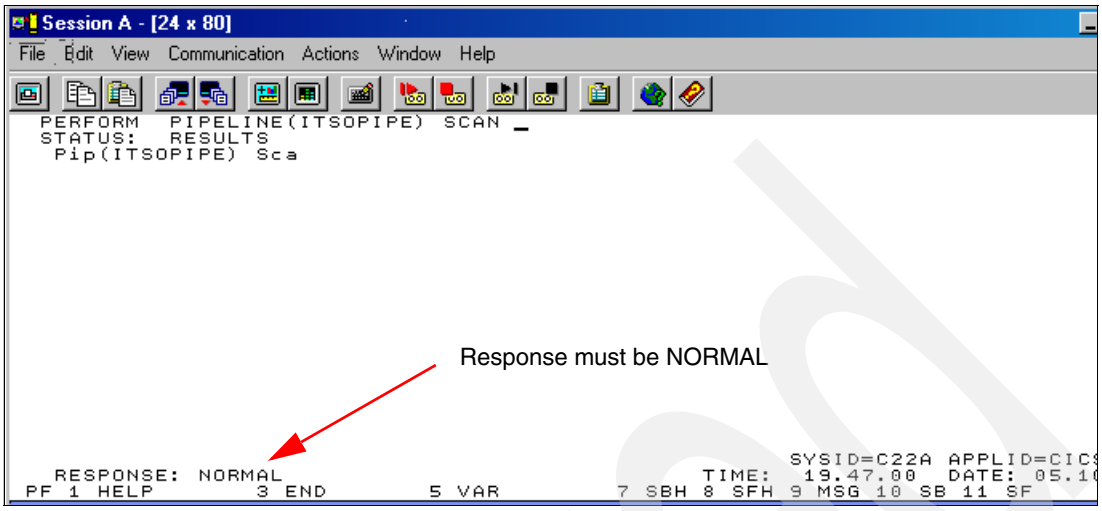


Figure 16-38 CEMT CICS PIPELINE SCAN

Inspecting the CICS WEBSERVICE and URIMAP

To inspect the CICS WEBSERVICE, use the CICS transaction CEMT INQ WEBSERVICE (COBOCRUD,COBOLIST), as shown in Figure 16-39. Note that CICS defined one URIMAP for each WEBSERVICE.

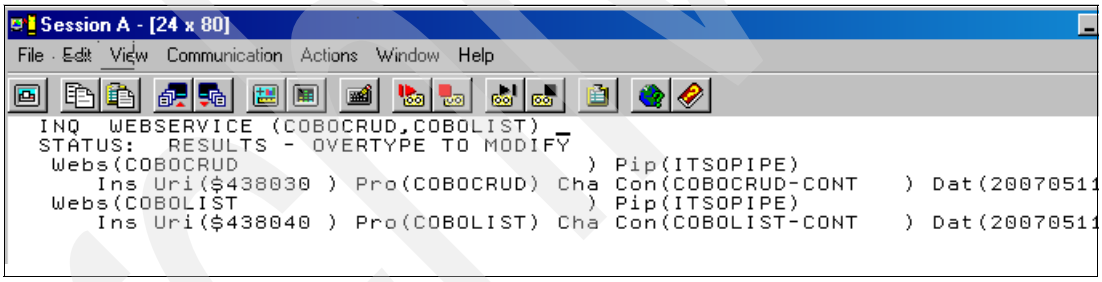


Figure 16-39 Inquiring the CICS WEBSERVICE

Still using the results shown in Figure 16-39, move the cursor to the resource you are interested in using and press Enter. In Figure 16-40 shows our example, where we are inspecting the COBOCRUD WEBSERVICE.

```
INQ  WEBSERVICE (COBOCRUD,COBOLIST)
RESULT - OVERTYPE TO MODIFY
  Webservice(COBOCRUD)
  Pipeline(ITSOPIPE)
  Validationst( Novalidation )
  State(Inservice)
  Urimap($438030)
  Program(COBOCRUD)
  Pgminterface(Channel)
  Container(COBOCRUD-CONT)
  Datestamp(20070511)
  Timestamp(04:38:03)
  Wsdlfile()
  Wsbind(/u/itso04/wdzv7pot/wspickup/provider/COBOCRUD.wsbind)
  Endpoint(http://zserveros.dfw.ibm.com:3091/u/ITSO04/wdzv7pot/wspickup/pr)
  Endpoint(ovider/COBOCRUD)
  Binding(COBOCRUDBinding)
```

Figure 16-40 COBOCRUD WEBSERVICE

To inspect the URIMAP, you can use the name defined and shown in the WEBSERVICE inquiry. For example, for the COBOCRUD, we use the transaction CEMT INQ URIMAP($438030), as shown in Figure 16-41.

```
INQ URIMAP($438030)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uri($438030 ) Pip Ena     Http
    Host(*                          ) Path(/u/ITS004/wdzv7pot/wspickup/prov)
```

Figure 16-41 Inquiring the URIMAP

16.5.8 Testing the Web Services

At this point, we have all ore assets created for our example and we are ready to invoke the CICS Web Services that were created.

Since this table is empty and we need to add some dates, we use the LIST application to be sure that we have no errors when the Web service is executed, and later we test the CRUD application by adding some data.

Using WD/z Web Services Test Explorer to invoke the LIST application

The Web Services Test Explorer allows you to test Web Services running locally or remotely in order to get instant feedback.

You do not need to be connected to the z/OS system or to the DB2 Database to perform those tests, but you need to have access to the z/OS system in which you have deployed the CICS Services.

To launch the Web Services Explorer from a WSDL file, select the WSDL file, right-click, and select **Web Services** → **Test with Web Services Explorer**. The Web Services Explorer opens. The Test with Web Services Explorer action will preload the WSDL into the WSDL window.

To test our LIST application, do these steps:

1. Using the Project Explorer view, expand **ITSO Appl Generator**, right-click **COBOLIST.wsdl**, and select **Web Services** → **Test with Web Services Explorer**.
2. When Web Services Explorer opens, expand **COBOLISTBinding**, select **COBOListOperation**, and type 1 for “list start” and 2 for “list count in” and click **Go.** This will list two rows, starting on the first row.

Archived

3. If everything is working correctly, the execution takes place, but because the DB2 table has no rows, you will not get any results, but you will have no errors either, and the SQL return code will be 0. See our example in the Figure 16-42.

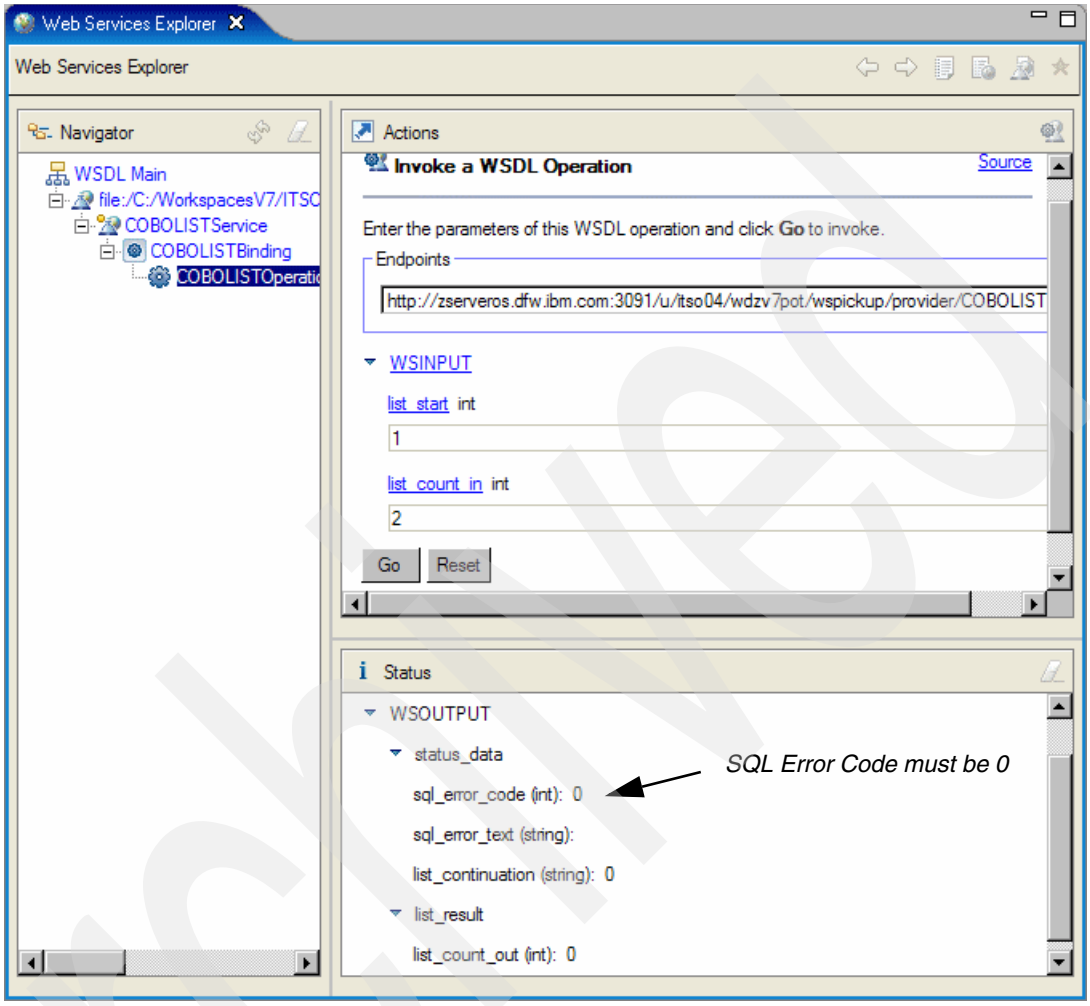


Figure 16-42 Using Web Services Explorer to invoke the LIST application

Using the CICS CEDX to trace transaction CPIH

It is a good idea to verify that the CICS services is working as designed. One simple way to do this verification is to use the CICS CEDX transaction. Each time the transaction associated with CICS WEB Services and the DB2 plan starts, we see the results of the transaction.

Using a CICS terminal, type the command CEDX CPIH, which will return the message `TRANSACTION CPIH: EDF ON`, as shown in Figure 16-43.

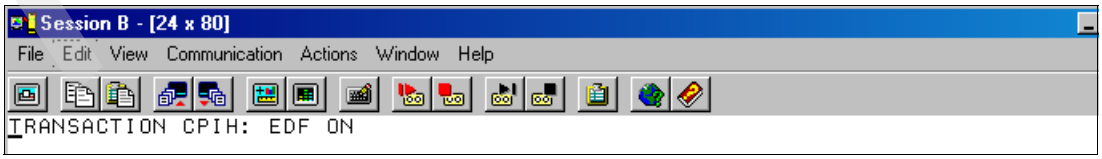


Figure 16-43 Using CICS EDF to verify CICS Web Services

Using the WD/z Web Service Explorer, click **Go** and use the values entered previously. The first EDF window appears, as shown in Figure 16-44. Note that the COBOL Driver program @OBOLIST is being invoked.

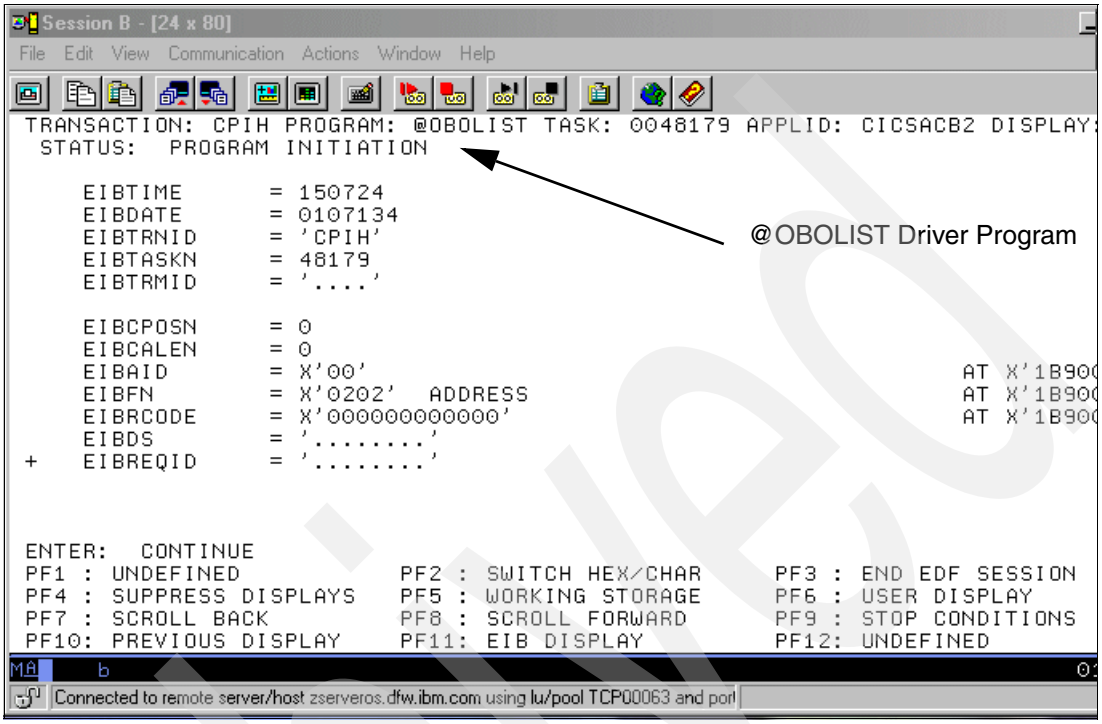


Figure 16-44 Using EDF to debug the CICS Web Service with the @OBOLIST Driver being invoked

Archived

Keep pressing Enter until you see the COBOLIST program being invoked and accessing the DB2 table. You must have 0 as the error return code, as shown in Figure 16-45. Press PF3 when you want to terminate the EDF session.

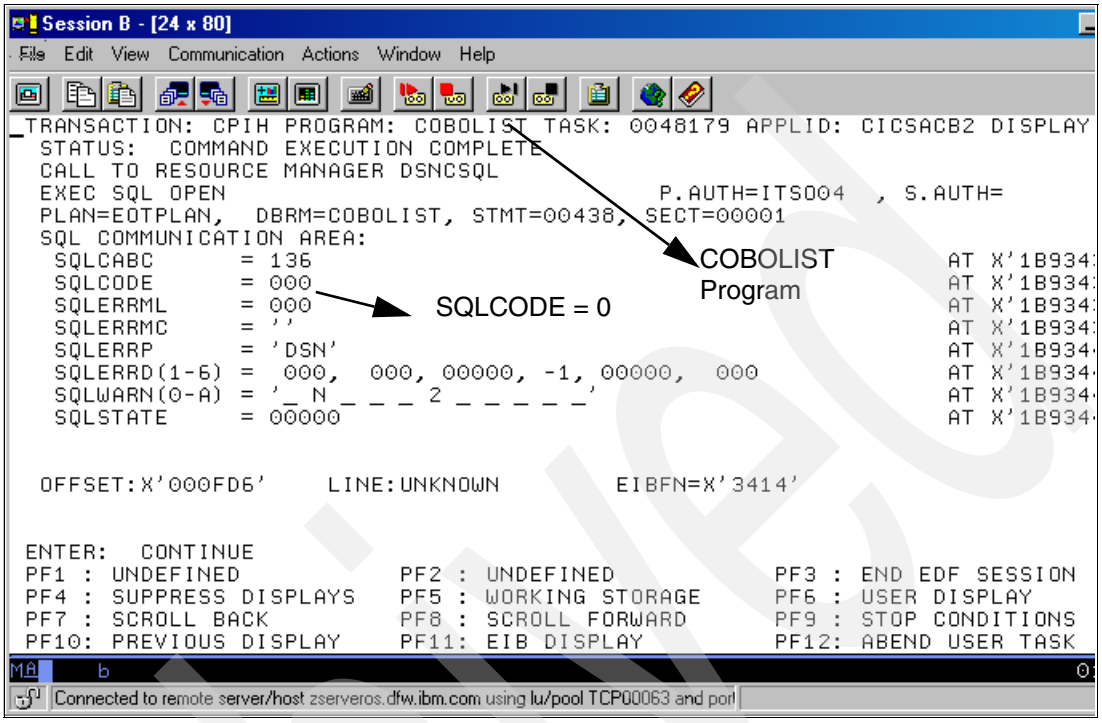


Figure 16-45 Using EDF to debug the CICS Web Service with COBOLIST being invoked

At this point, you have successfully tested the LIST application. You might test it again later with some data in the DB2 table.

Using WD/z to load data into the DB2 table

In our example, the DB2 table is still empty. You could use the CRUD application to create rows, but is easier to load rows into the table using DB2 insert commands. You could do this using DB2 tools, but you also can use the WD/z Data Project Explorer view to execute a DB2 DDL that will insert the rows.To perform this task, do these steps:

1. Be sure that you are connected to the z/OS DB2 database where the table was created. If you are not connected, use the Database Explorer view to connect to it.
2. Expand the Data Design project **ITSO DB2 Appl Generator** and insert the DDL that will be executed into the folder SQL Scripts. Example 16-6 shows the DDL that we used to load five rows into the table.
3. You can execute the DDL by using the same process you use when running SQL statements. Figure 16-46 on page 605 shows the Data project Explorer view with the DDL to be executed.

Example 16-6 DDL to load five rows in the DB2 table COBOLTYPES

```
-- Load DB2 table COBOLTYPES on zOS
INSERT INTO COBOLTYPES VALUES(01,'Exploring WebSphere Studio Enterprise
Developer',
   'Redbooks about WSED V 5.1.2',123, 'R. Barosa', '05/12/2005',
   '01:00:00', '2005-05-12-12.12.23.01',7701, 3301, 'ABCDE', 401, 501, 601, 8801,
   4501, 3401, 'Redbooks', '0738490660','SG24-6483-00','2007-05-15' );
```

```
INSERT INTO COBOLTYPES VALUES(02,'TITLE02','DESCRIPTION02',123, 'AUTHOR02',
'12/02/2007', '02:00:00',
        '2006-11-14-12.12.23.02',
        7702, 3302, 'ABCDE', 402, 502, 602, 8802, 4502, 3402, 'TYPE02', 'ISBN02',
        'CURS-CHEK-USE028','2007-05-02' );
INSERT INTO COBOLTYPES VALUES(03,'TITLE03','DESCRIPTION03',123, 'AUTHOR03',
'12/03/2007', '03:00:00',
        '2006-11-14-12.12.23.03',
        7703, 3303, 'ABCDE', 403, 503, 603, 8803, 4503, 3403, 'TYPE03', 'ISBN03',
        'CURS-CHEK-USER03','2007-05-03' );
INSERT INTO COBOLTYPES VALUES(04,'TITLE04','DESCRIPTION04',123, 'AUTHOR04',
'12/01/2007', '04:00:00',
        '2006-11-14-12.12.23.04',
        7704, 3304, 'ABCDE', 404, 504, 604, 8804, 4504, 3404, 'TYPE04', 'ISBN04',
        'CURS-CHEK-USER04','2007-05-04' );
INSERT INTO COBOLTYPES VALUES(05,'TITLE05','DESCRIPTION05',123, 'AUTHOR05',
'12/01/2007', '05:00:00',
        '2006-11-14-12.12.23.00',
        7705, 3305, 'ABCDE', 405, 505, 605, 8805, 4505, 3405, 'TYPE05', 'ISBN05',
        'CURS-CHEK-USER05','2007-05-05' );
```

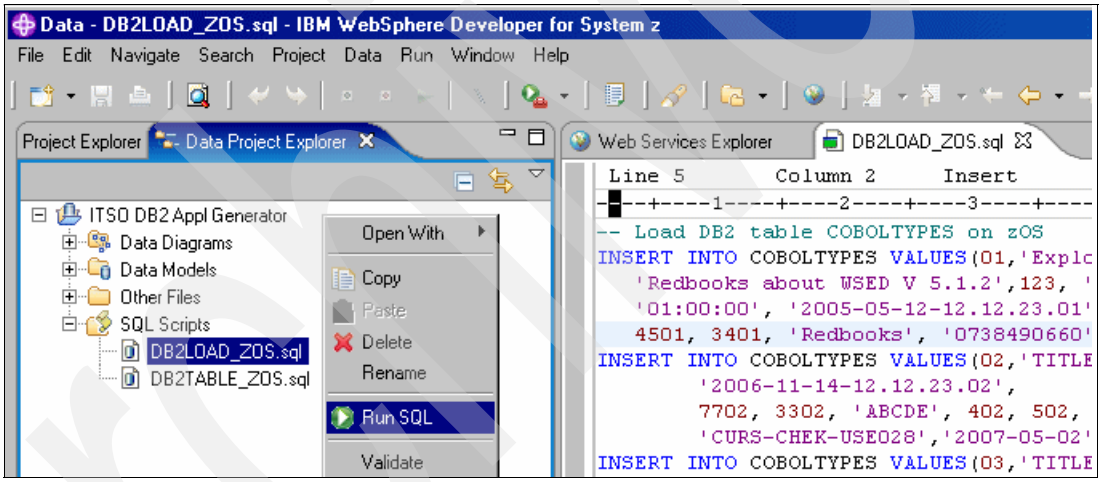


Figure 16-46 Loading data into DB2 table COBOLTYPES

Using WD/z Web Services Test Explorer to invoke the CRUD application

At this point, our DB2 table has five rows. You can perform activities on this table using the CRUD application and typing in a desired action. The CRUD application will accept the following actions:

- C or c: Create a DB2 row.
- R or r: Read a row.
- U or u: Update a row.
- D or d: Delete a row.

Note that the primary key is the column named “code” (ci code) and this value is always required for the operations above. Also, you need to have a value in all fields that require a numeric value. A small red asterisk is shown on the right of this field: ci_code int *

If you do not have values in the int, float, long, or decimal fields in the Status window, you will receive the error message `IWAB0383E Error validating WS-INPUTPart` when you click the **Go** button.

To test our CRUD application, do these steps:

1. Using the Project Explorer view, expand the **ITSO Appl Generator** project, right-click **COBOCRUD.wsdl**, and select **Web Services** → **Test with Web Services Explorer**.
2. When Web Services Explorer opens, expand **COBOCRUDBinding**, select **COBOCRUDOperation**, and type R for action, 1 for ci code and type any numeric value (like 0) in the ci owner, ci_publishint, ci_publishfloat1, ci_publishfloat2, ci_publishfloat3, ci_publishreal, and ci_pubsnext fields and click **Go.** This action will read row 1 of the table COBOLTYPES.
3. If all is working fine, the execution takes place, and you will see the first row, as shown in Figure 16-47.
4. You should also see the SOAP envelope that is being used in this communication. Click the **Source** link in the Status window. Figure 16-48 on page 607 shows the XML data and the SOAP Request and Response Envelope.

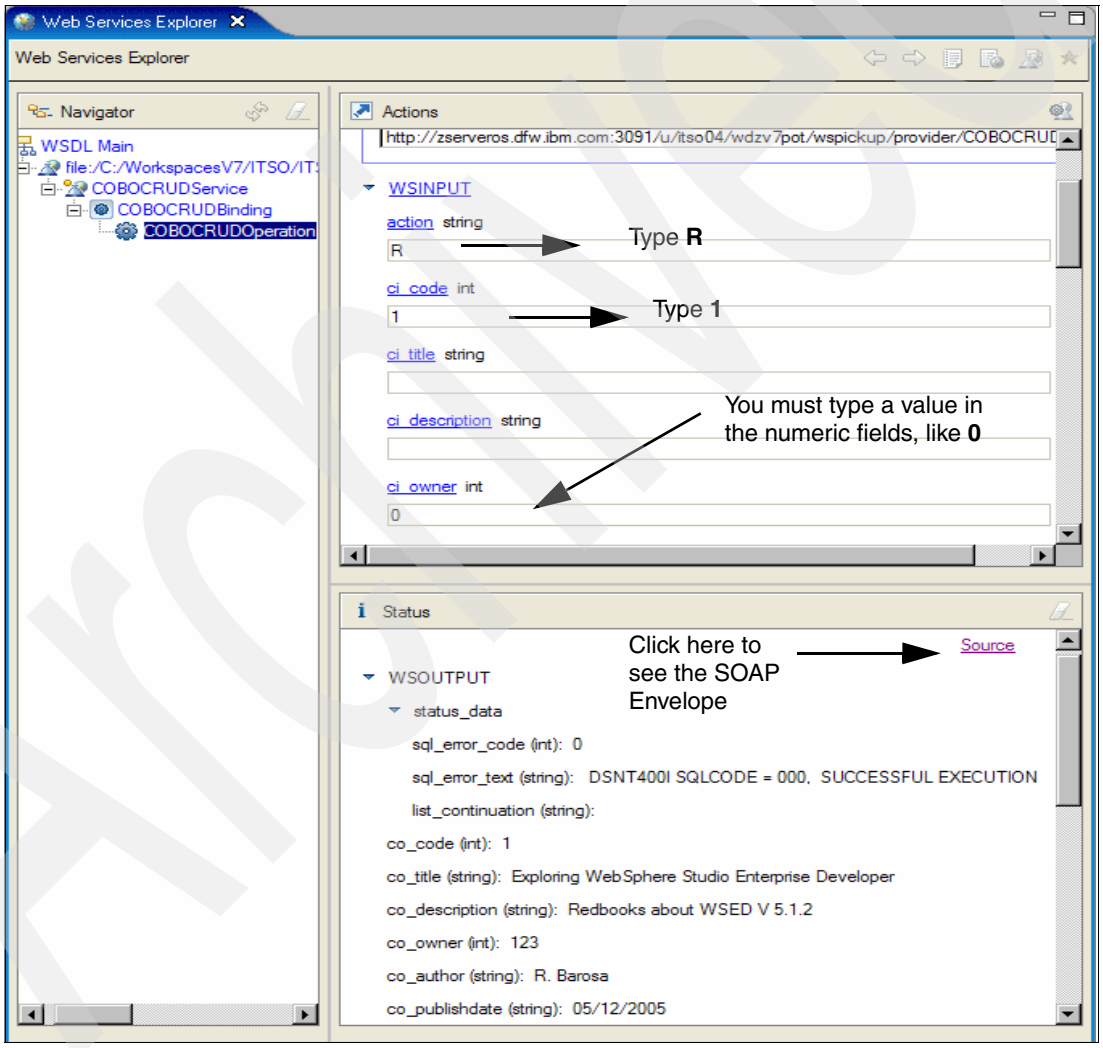


Figure 16-47 Testing the CRUD application with action R (Read)

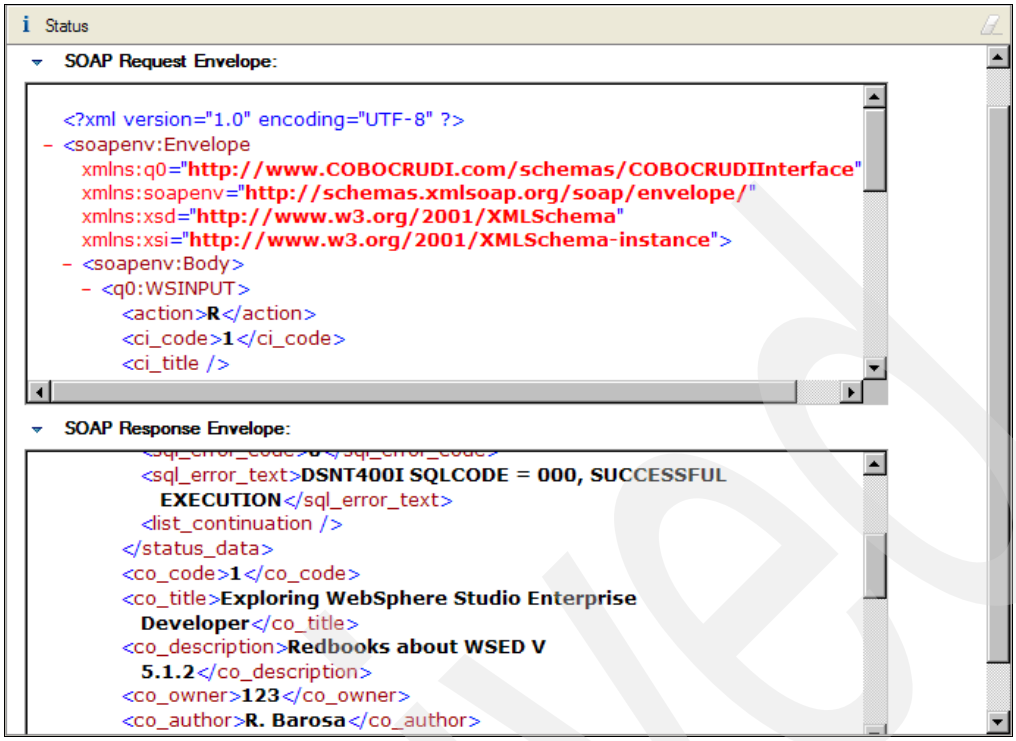


Figure 16-48 SOAP Envelope used in the Read action of CRUD application

You may try other actions, such as Create, Update, and Delete.

Also, if you have the Debug Tool for z/OS installed on your System z server, you will be able to debug the COBOL generated code.

16.6 Creating a Web service client using Java Server Faces

We tested a CICS Web Service using the Web Service Test Explorer, but if you deploy this CICS service to users, you need to create a better client interface. This can be done by using multiple tools that create client interfaces from existing WSDL. In this section, you create a Web client interface using a Java Server Faces (JSF) framework and WD/z wizards that help create Web pages using Java Server Pages (JSPs).

To make this example simple, we create just one page using the List application generated before (COBOLIST).

The steps that you must perform are:

1. Create a Web project with a simple JSP page.
2. Add the Faces JSP controls that will invoke the Web service.
3. Test the JSP Web page.

Archived

16.6.1 Creating a Web project with a simple JSP page

You must create a dynamic Web project to contain the Faces JSP file. You can create the Web project by following these steps:

1. Select **File** → **New** → **Other** → **Web** → **Dynamic Web Project**. Click **Next**.
2. Type ITSO Client in the Name text field. Select your run time from the Target Runtime list; if your preferred server is not listed, click **New** and add the required information to configure it. In our example, we used WebSphere Application Server v6.1. In the Configurations field, select **Faces Project** from the list. In the Ear Project Name, use ITSO ClientEAR as the name. Click **Next**.
3. Be sure that **Base Faces support** and **JSTL** are selected and click **Finish**. These are necessary frameworks for the JSF. You might de-select others like Crystal Reports.

Figure 16-49 shows our example.

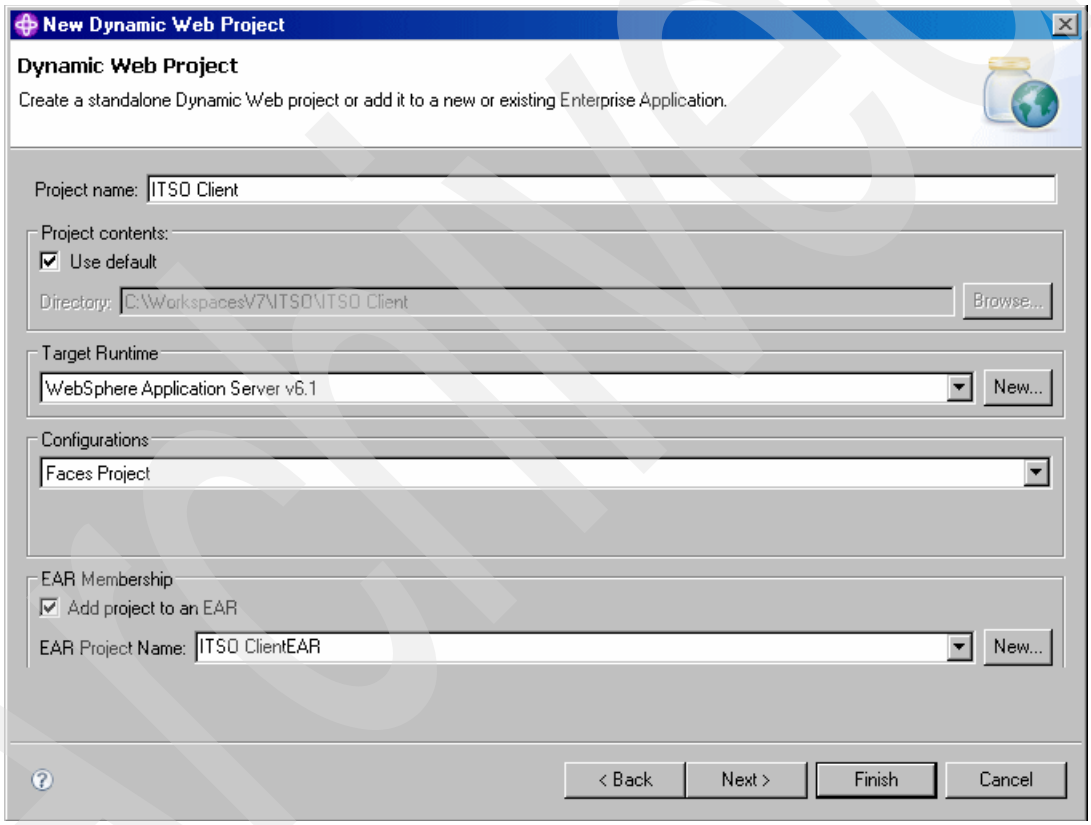


Figure 16-49 Creating a Dynamic Web Project to hold a Faces JSP file

If you are prompted to switch to the J2EE perspective, select **No**.

You can create a Faces JSP file on which the created Web service proxy code will reside by doing these steps:

1. Expand the Web project that you created, and select the **WebContent** folder.
2. From the File menu, select **New** → **Other** → **Web** → **Web Page**.
3. Select a name for the Faces JSP file. Ensure that the JSP template is selected from the template list, and click **Finish**. We use ITSO_PAGE.jsp in our example, and we select the **A_blue** sample template. To be sure that you are creating a JSP page instead of an HTML

file, type the suffix at the name being created, such as xxx.jsp. The Faces JSP file that you have created opens in an editor. Figure 16-50 shows our example.

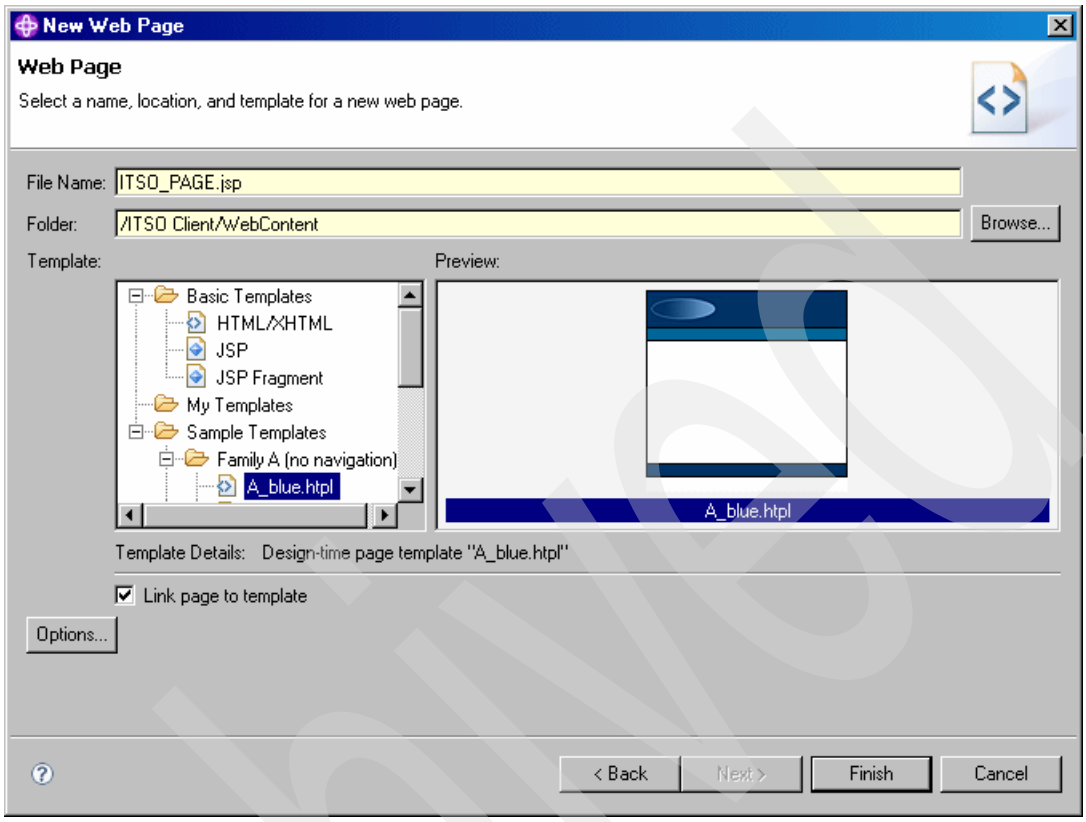


Figure 16-50 Creating a JSP page

16.6.2 Add the Faces JSP controls that will invoke the Web service

You must now add the user interface controls to the JSP window so that you can enter the values and invoke the CICS Web Service later.

Do these steps:

1. Using the Web Perspective, in the Palette, select the **Data** tab. Select **Web Service**, right-click, and click **Insert**.
2. When the Web Service Discovery Home window opens, click **Web Services from your workspace**. The WSDL that you need to use is on your workspace, but in another project.
3. In the Filter field, use the drop-down and select **Workspace WSDL documents**. The services that you have in your workspace are listed. Click **COBOLISTService**.

4. The Web service we have selected is now displayed in the table. Click **Port: COBOLISTPort** to select it and click **Add to Project**. Figure 16-51 shows our example.

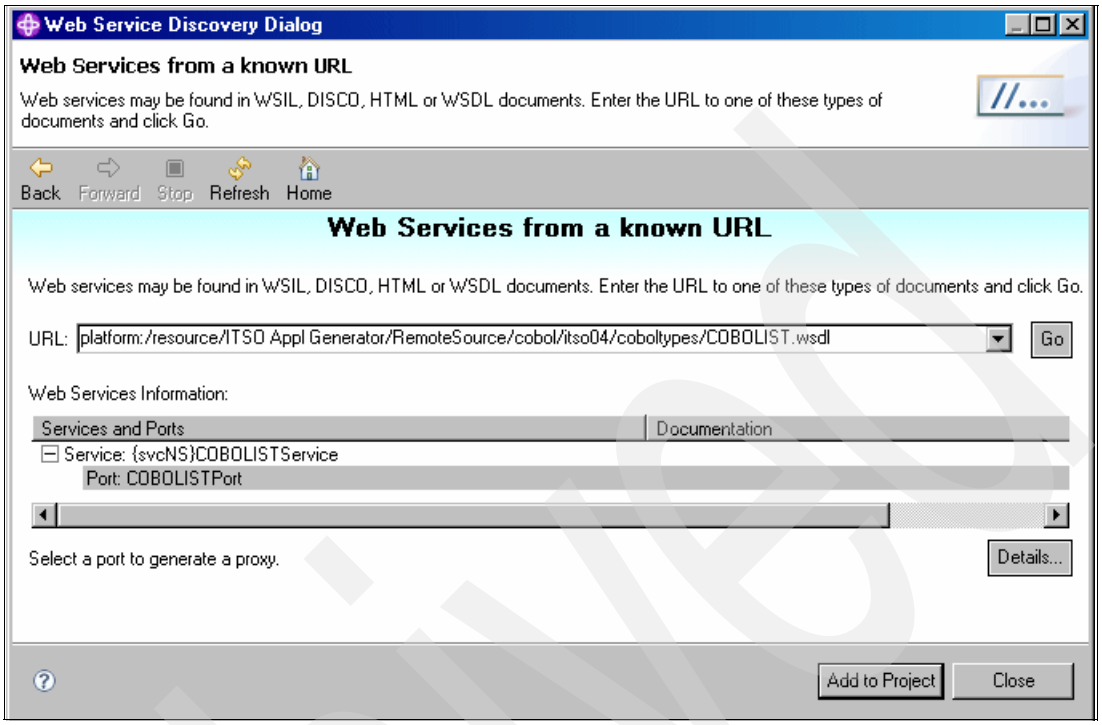


Figure 16-51 Adding the Web Service COBOLIST to our JSP window

5. If the warning message “Cannot create the file ‘web.xml’ relative to the path ‘/ITSO Client/WebContent/WEB-INF’ because automatic file overwriting has not been enabled. Do you want to enable it for this file?” appears, click **Yes**.
6. The Web service COBOLISTPortTypeProxy is now listed in the list of existing Web Services. When you select it, the only method available for this Web service is listed in the methods list.
7. Click **Next** to see the data controls that you might have in the Input Form.
8. Click **Next** again to see the controls in the Results Form. Note that the list_entries will be a Multi-Column Data Table.
9. Click **Finish** to add the controls to your JSP. Figure 16-52 on page 611 shows the JSP with all default controls created.
10. Save the JSP.

You could customize the JSP, but you can make the first tests at this point.

Archived

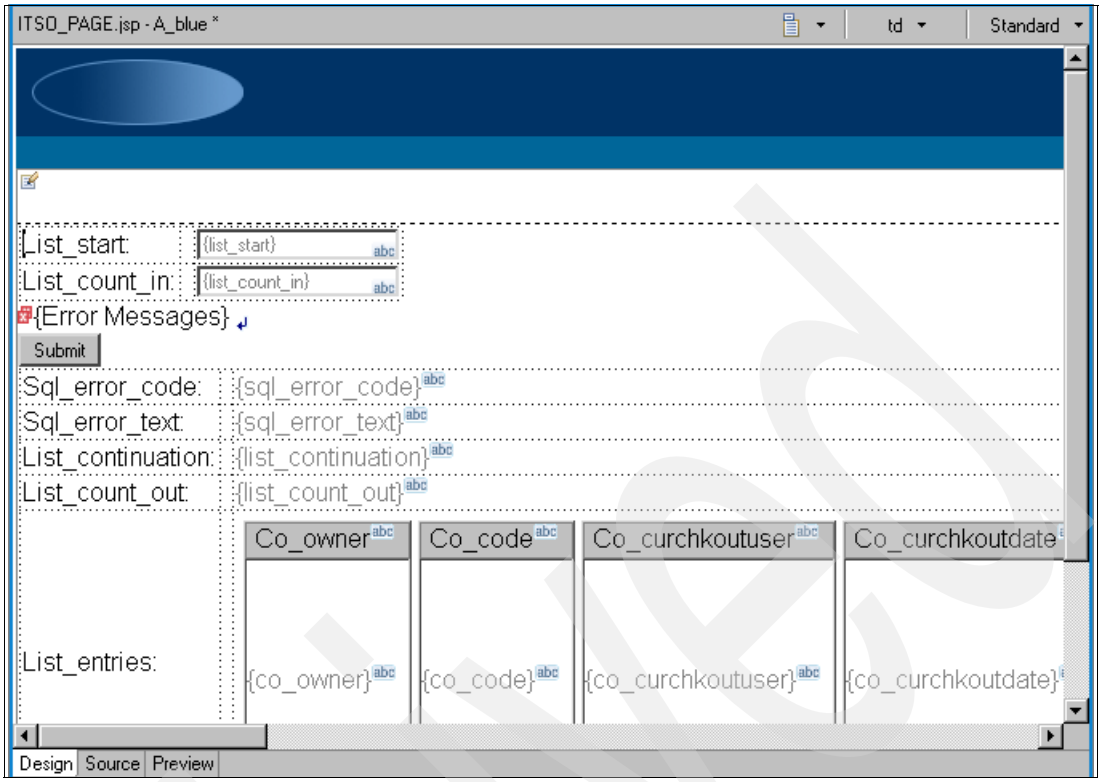


Figure 16-52 JSP window with the Web service and its controls

The proxy COBOLISTPortTypeProxy should now be shown in the Page Data view. If the Page Data view is not displayed, you can open it by selecting **Window** → **Show View** → **Page Data**.

16.6.3 Testing the JSP Web page

WD/z created the necessary Java components that will deploy this Java Client application. The project named ITSO ClientEAR must be deployed to the Application Server for tests.

This can be done automatically during the tests. No special definitions need to be done at the application server, as all the database I/Os will be done at the CICS in the z/OS system. The Java Client application will just invoke it.

To run this Java client application, do these steps:

1. In the Project Navigator, right-click **ITSO_PAGE.jsp** and select **Run As** → **Run on Server**.
2. Select the server that is associated with the dynamic Web project, in our example, it is **WebSphere Application Server v6.1**, and click **Next**.
3. You will see the project ITSO ClientEAR added to your server. Click **Finish**. If you want to run the Faces JSP on another server, you would need to re-target the Web project that you created to the new server.

Archived

4. Once the JSP is running on the server, it should open a page in a Web browser in the workspace. You can test the Web service by entering 1 in List_start and 5 in the List_count_in fields and clicking **Submit.** The first five rows will be listed, as shown in Figure 16-53.

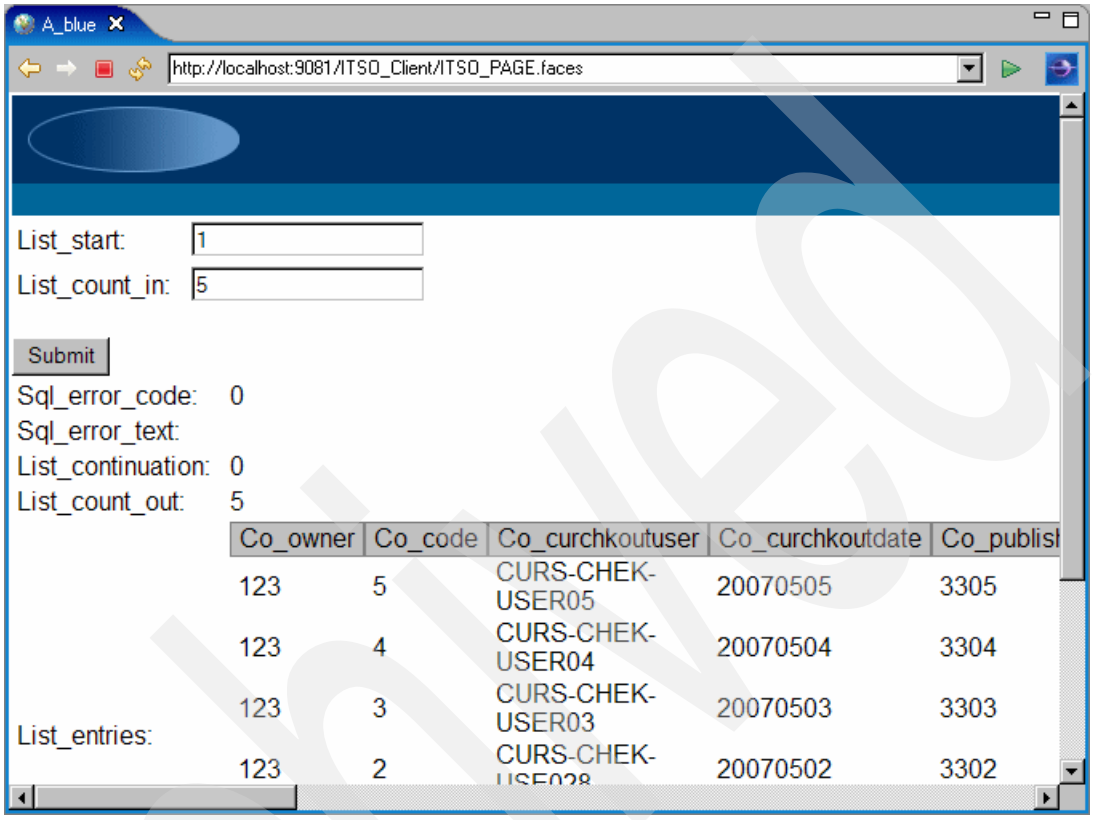


Figure 16-53 Web page results when invoking the COBOLIST CICS Web service

Note that you are executing a Java client program that is running in WebSphere Application Server Version 6.1, which is invoking the CICS Web Service that is deployed in the z/OS. Figure 16-54 on page 613 shows this architecture.

Similar JSP pages could be created for the COBOLCRUD CICS Service.

Archived

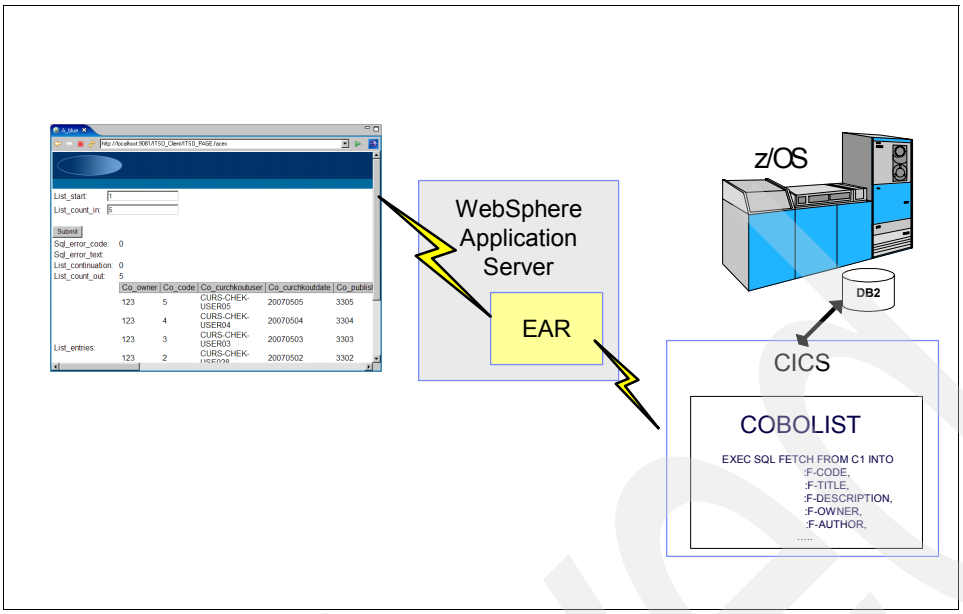


Figure 16-54 Web page invoking CICS Web Services

16.7 Scenario 2: Working with an Unified Modeling Language model

In this scenario, we assume that you have created an Unified Modeling Language (UML) model using Rational Software Architect and want to transform this model into a database and generate COBOL programs using the Data Base Application Generator wizards.

16.7.1 Exporting the UML model using Rational Software Architect

In Unified Modeling Language, models contain model elements, such as actors, use cases, classes, and packages, and one or more diagrams that show a specific perspective of a system. Rational Software Architect is the IBM product that allow users to work with UML models. It has perspectives and views especially designed for this task.

Note that a model is both a concept and a modeling element. As a concept, a model is an abstraction of a system from a particular perspective and level of detail. As a modeling element, a model and its corresponding model elements appear in the Rational Software Architect Project Explorer view. These model elements are graphically represented as diagram elements in a diagram.

A modeling project can contain any number of models. You can open and manage each model in the Project Explorer view, and you can open diagrams from more than one model at the same time.

A UML class diagram provides a graphical representation of the structures and relationships of elements in an application. A class diagram can depict some or all of the components or elements in an application. You can use class diagrams to create your own context to examine, understand, collaborate, and design using a subset of the components or elements in an application.

Archive

The assumptions we have while working with UML are:

- Any class in the UML is treated as a DB2 table.
- Any property of a class is treated as a column.
- Other UML items are ignored, including Operations and Methods in the class.

We assume in this scenario that you have used Rational Software Architect (RSA) to create the UML model and that the DB2 table to be used in this application has not been created yet. The UML model that we used is shown in Figure 16-55.

Note that we want to implement a DB2 table for Employee later, which is why we added more attributes to this class.

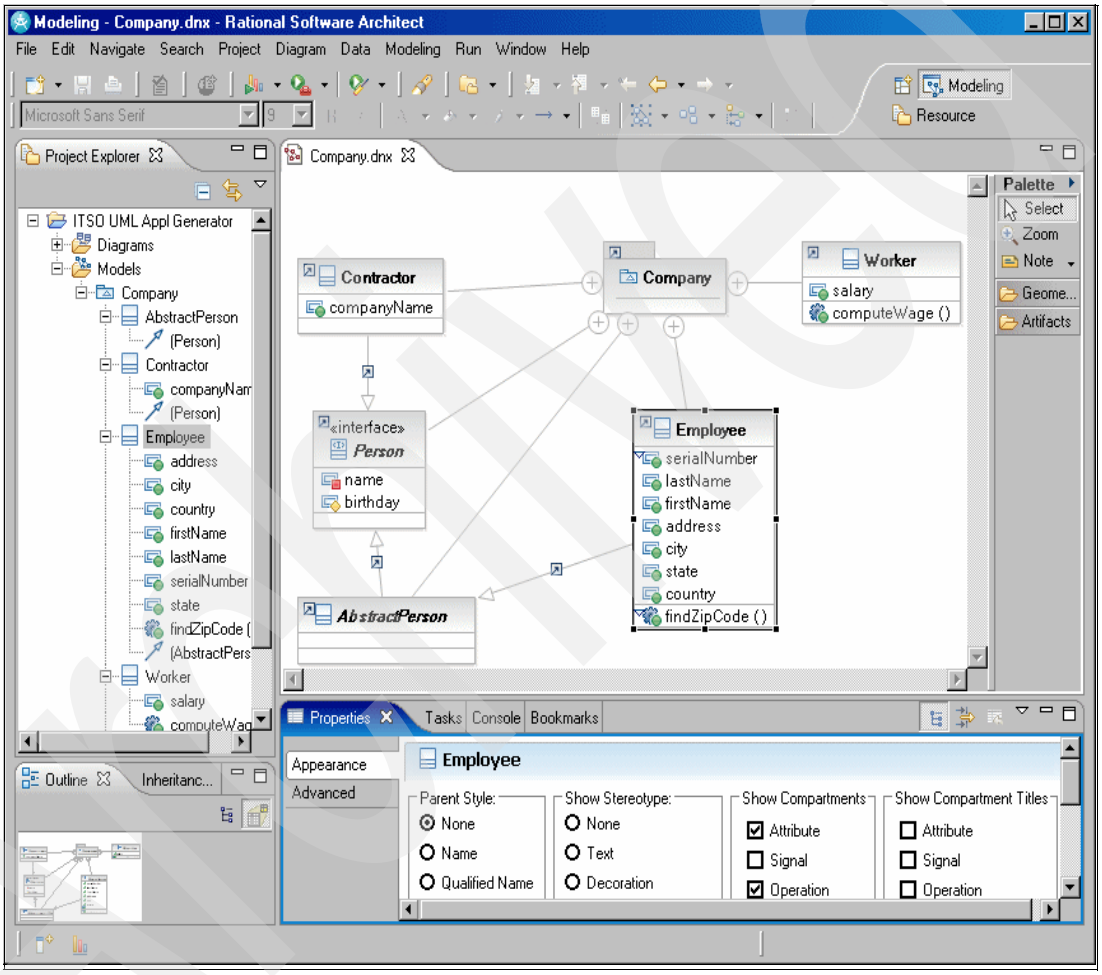


Figure 16-55 Rational Software Architect Modeling Perspective with the class diagram

Using RSA, the UML can be exported by doing these steps:

1. Select **File** → **Export**.
2. In the Export wizard, in the Select an export destination field, click **UML Model**, and click **Next**.
3. In the UML Model window, in the Source field, specify the model that you want to export, and, in the Destination field, specify the target directory for the exported model, and click **Finish**.

16.7.2 Creating a Simple Project and importing the UML model

You could use the same existing database application project created in the first scenario, but to separate the scenarios, we created a new project.

Because in this scenario we assume that there are no DB2 tables created, you cannot create a database application project that will require a table. You must create a simple project and import the UML model into this project.

Creating a Simple Project

To create a WD/z Simple Project, follow these steps:

1. Using any perspective, from the menu bar, select **File** → **New** → **Project...**.
2. In the New Project wizard, select **General** → **Project** and then click **Next**.
3. In the Project name field, type your name as the name of your new project. We used ITSO UML Appl Generator. Leave the box checked to use the default location for your new project. Click **Finish** when you are done. Note that you must have a Project Explorer view to see this project; in our example, we have added this view to our Data perspective.

Importing the UML into your project

The Import wizard can be used to copy the UML into your project. Do these steps:

1. Right-click the project **ITSO UML Appl Generator** and select **Import...**.
2. In the Import wizard, expand **General**, select **File system**, and then click **Next**.
3. Point to the directory containing the UML exported file and click **Finish**. Figure 16-56 shows the imported UML.

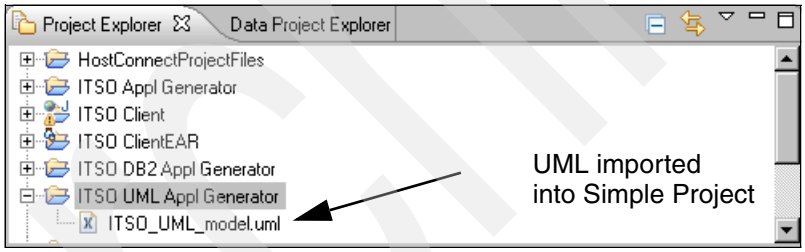


Figure 16-56 Project Explorer view with the UML imported to a Simple Project

16.7.3 Using the Database Application Generator wizard from UML

Note that the model does not have all implementation characteristics. For example, when implementing an employee's serial number, you must define this attribute as numeric. These implementation details must be configured by developers.

When working with transformation, you normally follow these steps:

1. Create a Transformation Parameters Model (TPM) file, which is a file tied to the UML model that describes how to translate the UML elements into source language elements. The TPM file contains such information as how to convert types and names into the source language, and how the output code should be structured.
2. Run the transformations and create code in the output language.
3. Edit and implement the output code.
4. You may iterate through this cycle several times, either to generate different parts of the application or to test and debug the application.

Creating a transformation parameters file

A TPM file contains meta data that describes how to transform a UML file into another type of code.

Follow these steps to create a new TPM file for our new project example:

1. Using the Project Explorer view, right-click the UML file that you have imported (**ITSO_UML_model.uml**) and then click **Edit Transformation Parameters**.
2. In the Create a New TPM File window, make sure that your UML file is listed in the Target Model field and click **Finish**. The new TPM file opens in the transformation parameters editor.

Editing a transformation parameters file

The transformation parameters file allows you to customize the transformation of the UML model to source code.

The left side of this editor is a table that matches the elements in the UML file with the transformations applied to those elements. The rows of the table are the elements in the UML model, which you can expand and contract with the + and - buttons. Figure 16-57 shows part of our example after expanding all the elements.

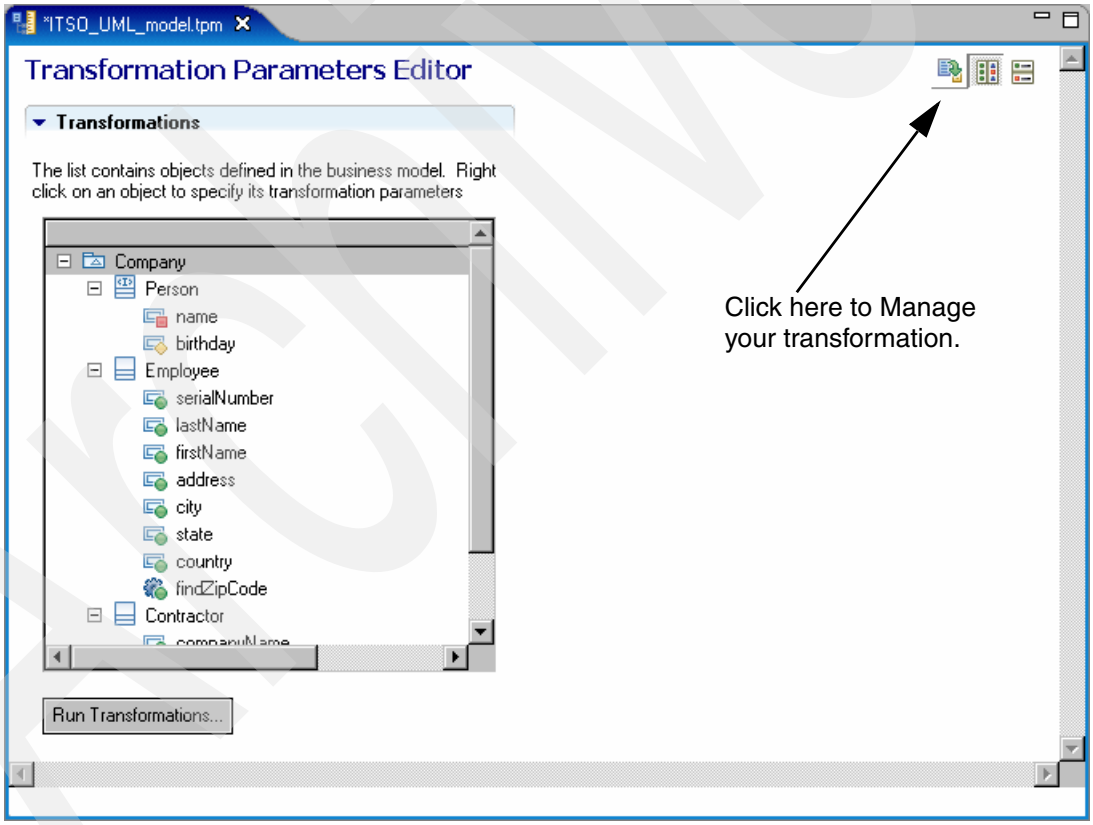


Figure 16-57 TPM Editor for our example

Selecting the available transformations

If you have multiple transformations on your WD/z, you might select which one you want to apply. This step is not required for users that have installed WD/z on top of Rational Application Developer without any other product that can manage transformations.

Archived

In that case, follow these steps to select the transformations that you want to be available in the transformation parameters editor:

1. At the top right corner of the editor, click the Manage applied transformations check box icon . The Manage Your Transformation Parameters window opens.
2. Move the transformations that you want to use into the Selected Transforms list by clicking a transformation in the Available Transforms list. In our example, select **z/OS Data Access** and then click **Add** and **OK**.

Applying a transformation to the UML elements

To apply a transformation to an element in the UML model, right-click the UML element and then click the name of the transformation.

In our example, the only choice is z/OS Data Access, but we could have more transformations, depending on the product being used. For example, if you have Rational Business developer extensions installed, you will have the choice to have EGL transformation also.

Figure 16-58 shows this action for the first element Company. You must execute the same action for each element that you want to transform. Since you are interested in the class Employee, you need to right-click each element of the Employee class and select **z/OS Data Access** for transformation.



Figure 16-58 Selecting the UML element to be transformed

The check mark symbol appears in the row of the UML element and in the column of the selected transformation, indicating that the transformation will be applied to the associated UML element.

Depending on the type of transformation, you may want to apply the transformation to the root element of the UML model or to any of its child elements. When you apply a transformation to a parent element, the editor does not automatically apply that transformation to the child elements.

Archived

Also, note the following considerations when working with the UML model:

- If you have defined an abstract class in UML, such as AbstractPerson in our example, you will not be able to apply the transformation to the class. A warning will be displayed when this element is selected, as shown in Figure 16-59.

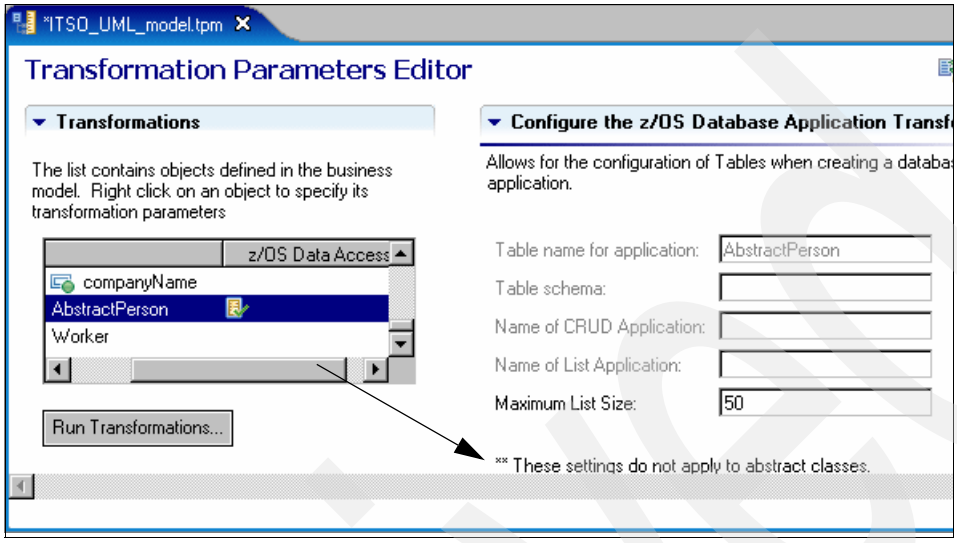


Figure 16-59 Warning message when trying to transform an abstract class

- If you have defined an interface in UML, such as Person in our example, you will not be able to apply the transformation to the interface. However, you can configure properties.
- You cannot configure column properties of abstract classes and interfaces that are set as private, such as, in our example, the UML element name (from Person).
- Some UML elements are ignored in the transformation, including Operations and Methods in the class, such as the findZipCode and computeWage elements in our example.
- If you have defined any relationships between classes, they will be ignored.

Configuring the transformation

To edit and configure the parameters associated with a transformation, click the check mark symbol in the transformation's column in the UML element's row.

Because the transformation parameters apply to a particular UML element and a particular transformation, make sure you have clicked the correct check mark symbol. The parameters available to be set are different for each transformation and for each type of UML element.

When you have clicked a check mark symbol, the parameters for the selected transformation and for the selected UML element appear in the right side of the transformation parameters editor. Not all combinations of UML elements and transformations have parameters.

Figure 16-60 on page 619 shows all the UML elements that we have selected for the transformation. Basically, we have selected all of them, except Worker. Note that even though we have selected almost all of them, some elements will not be transformed (Person, name, AbstractPerson, and computeWage).

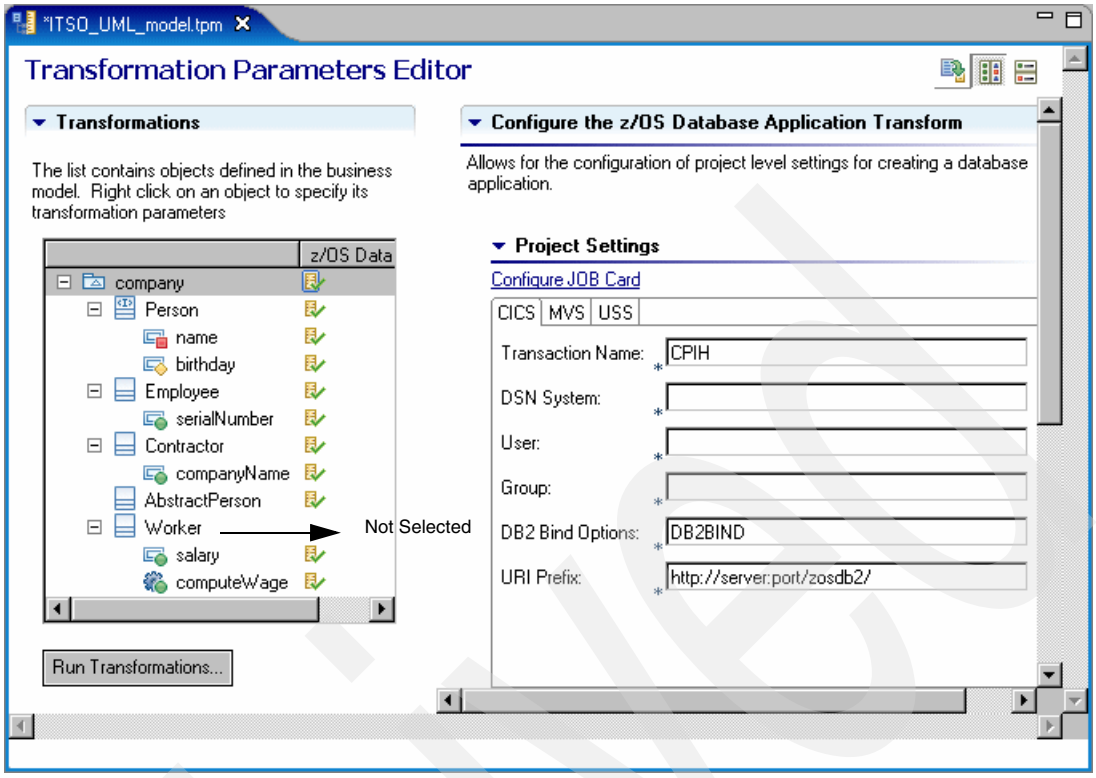


Figure 16-60 UML elements selected for transformation

Table 16-2 shows the configurations for each element of class Employee.

Table 16-2 Configurations

Employee elements	Column SQL Type	Maximum length	OBS
serialNumber	INTEGER		Primary Key Column
lastName	VARCHAR	25	Searchable Column
firstName	VARCHAR	15	Searchable Column
address	VARCHAR	20	Nullable Column
city	VARCHAR	20	Nullable Column
state	VARCHAR	5	Nullable Column
country	VARCHAR	15	Nullable Column
Person elements			
birthday	DATE		Nullable Column

In our example, we defined serialNumber as the primary Key Column, as shown in Figure 16-61 on page 620. Also, similar configurations for the scenario listed in 16.5, “Scenario 1: Working with the existing z/OS DB2 table” on page 567 were done in this example, like DSN DB2 system, CICS transaction group, DB2 bind options, and so on.

You need to configure “birthday” also, even if it is not an attribute of the Employee class. The reason is because “Employee” extends the abstract class “AbstractPerson”, which implements the interface “Person”, and “birthday” is a non-private attribute of the interface Person, so this element will be also part of the table “Employee” and need to be configured also.

You will notice the “name” attribute is a private attribute of class “Person”, so it is not generated.

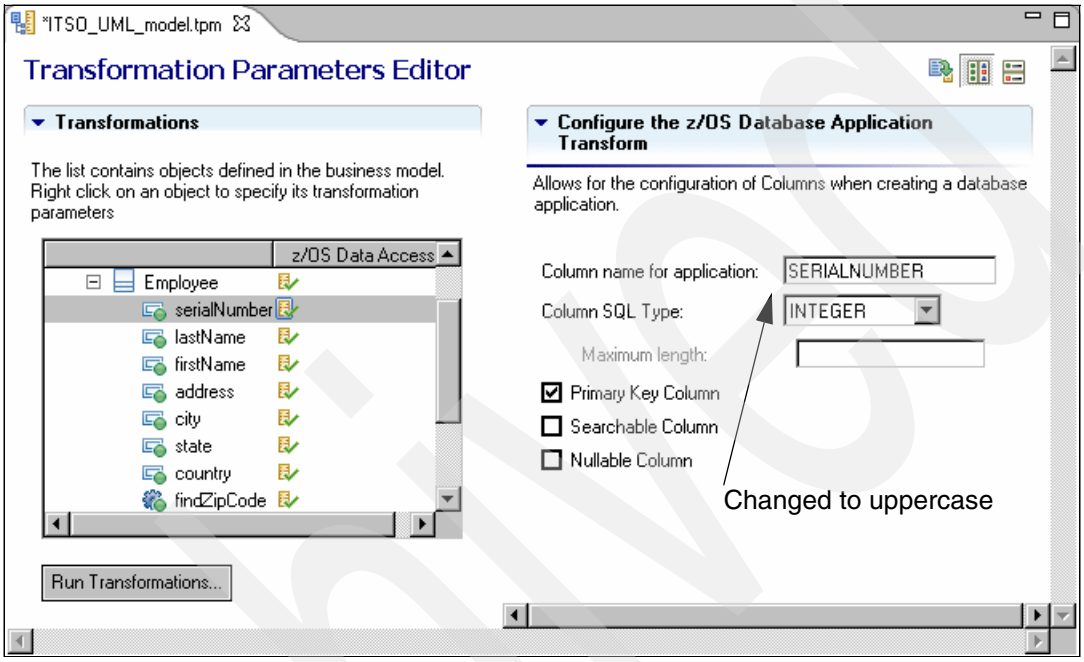


Figure 16-61 Configuring the serialNumber UML element

Note: In the WD/z version that we used in writing this book, all column names in the SQL statements where generated as mixed case. Even though we had no compile errors, we had errors at BIND time, such as:

```
SQLSTATE 42703 (or SQLCODE -206): Column name not in any table referenced inthe
FROM clause or in the table on which the trigger is defined.
```

In order to have the bind working, we need to change all SQL statements to uppercase. So we recommend you change those names to uppercase in the TPM editor to avoid manual changes after the generation is performed, which is what we did in our example, including the column “birthday” from Person.

Basically, you need to configure the following settings after clicking **Company**:

- CICS page: DSN System, User, CICS Group, DB2 Bind Options, and URI Prefix.
- MVS page: The data sets names. If you have already defined them in the workspace properties, they will all be already specified here.
- UNIX System Services page: No configuration is required unless you want Automatic Remote Deployment.
- De-select **Automatic Remote Deployment** and use UML as the JCL Prefix.

When configuring Employee, you must use ITSO04 as the table schema.

After you are finished, save the TPM file.

Running the transformation

Before you run the transformation, you must connect to the z/OS system. You can use the connection created before. In our example, we use the connection named ITSO.

The main steps to be performed are:

1. As you did in 16.5, “Scenario 1: Working with the existing z/OS DB2 table” on page 567, click **Run Transformations**.
2. The Specify configuration properties window appears. Select all of the properties and click **Finish** to execute the transformation. Note that nothing will happen if you are not connected to the z/OS system.
3. A warning message may tell you that some deployment artifacts will not be generated. Click **OK**.

You may expand the **RemoteSource** folder under you project to verify the generated code, as shown in Figure 16-62.

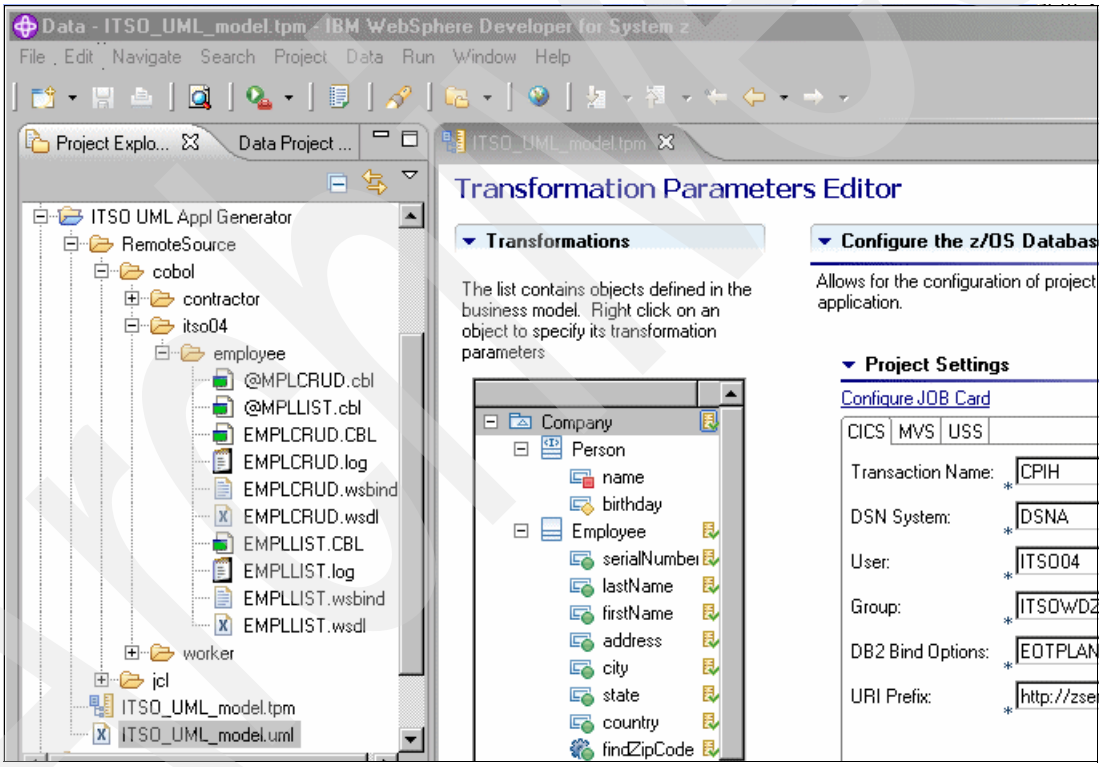


Figure 16-62 Generated assets from transformation

Note that the transformation assumes that each class will be mapped into a DB2 table and create applications for each of these tables. Because we have three classes (Person is an interface and AbstractPerson is abstract), we have three groups of applications, and each COBOL program will have the name of those future DB2 tables as a prefix.

Note: The WD/z version we used when writing this book did not allow us to generate just one table; we had to generate ALL of them, even though we wanted to only implement Employee. We discarded the other COBOL generated code (contractor and worker), but we also need to tailor the generated JCL, because it was prepared to use compile, link, and bind all programs

The next steps are creating the DB2 table in the z/OS system and deploying the generated assets, as we did in 16.5, “Scenario 1: Working with the existing z/OS DB2 table” on page 567.

16.7.4 Creating the DB2 table in the z/OS system

The transformation does not generate any DDLs that you can use to create the DB2 table, but it is easy to create a DDL by using the COBOL generated programs that use that table.

For example, to create the Employee table, you can edit the program EMPLCRUD.cbl, and you will see, in the working storage area, the COBOL definitions for this table, as shown in Figure 16-63. From here, it will be easier to create a DDL to create the table.

You also may use WD/z and the Data perspective to create and load this table, as you did in “Using a DB2 Data Design Project to create and load the DB2 table” on page 568.

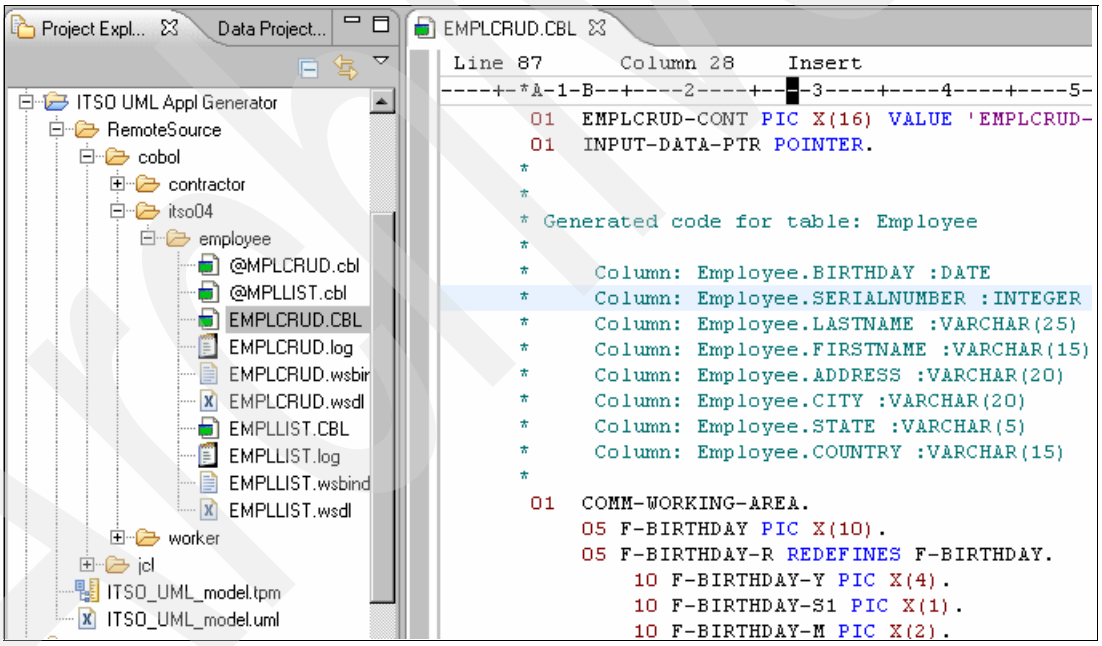


Figure 16-63 Program CRUD generated for table Employee

In our example, we used the DDL shown in the Example 16-7 on page 623 to create and load data into the table.

Example 16-7 DDL used to create and load table Employee

```
-- create and load table EMPLOYEE on zOS
 DROP TABLE EMPLOYEE;
CREATE TABLE EMPLOYEE
(  BIRTHDAY     DATE,
SERIALNUMBER INTEGER NOT NULL,
LASTNAME     VARCHAR(25) NOT NULL,
FIRSTNAME    VARCHAR(15) NOT NULL,
ADDRESS      VARCHAR(20),
CITY         VARCHAR(20),
STATE        VARCHAR(5),
COUNTRY      VARCHAR(15),
 PRIMARY KEY (SERIALNUMBER) );
CREATE UNIQUE INDEX XEMPLOYEE
    ON EMPLOYEE
      ( SERIALNUMBER ASC);

....
INSERT INTO EMPLOYEE VALUES('04/22/1987', 6,'Barosa','Thiago','7 Sao
Benedito','Sao Paulo','SP','BRASIL');
INSERT INTO EMPLOYEE VALUES('07/12/1985', 7,'Barosa','Daniel','123 Calle Oro',
'Buenos Aires','DF','ARGENTINA');
INSERT INTO EMPLOYEE VALUES('01/25/1967', 8,'Page','Megan','2 Victory
Rd','Boston', 'MA','USA') ;
```

16.7.5 Deploying the code generated

To deploy the generated COBOL code, you must perform the same steps that were completed earlier.

At this stage, you might already have some of the CICS configured, and will just add another service to this configuration.

Do these steps:

1. Adapt the generated JCL to your environment. In our example, we had to change the JCL, because we did not want to implement programs for all tables, but just for the Employee table.
2. Move or copy the COBOL generated code to z/OS, if Automatic Remote Deployment is not selected.
3. Move or copy the WSDBIND files to the HFS directory for the programs you want to implement, if Automatic Remote Deployment is not selected.
4. Submit the JCL generated to compile, link, and bind the programs and create the CICS Resources. We also changed the generated JCL because there was some resources for COBOL programs that we do not want to implement.
5. Install CICS resources and use the CEDA transaction to scan the CICS PIPELINE.

Now you can test the code.

Archived

16.7.6 Testing the Web Services

You can test the CICS Web Services using the Web Services Explorer. Because in this example we specified the lastname and firstname DB2 columns as searchable, when testing the EMPLLIST Web service, you have the option to type the string to be searchable, as shown in Figure 16-64.

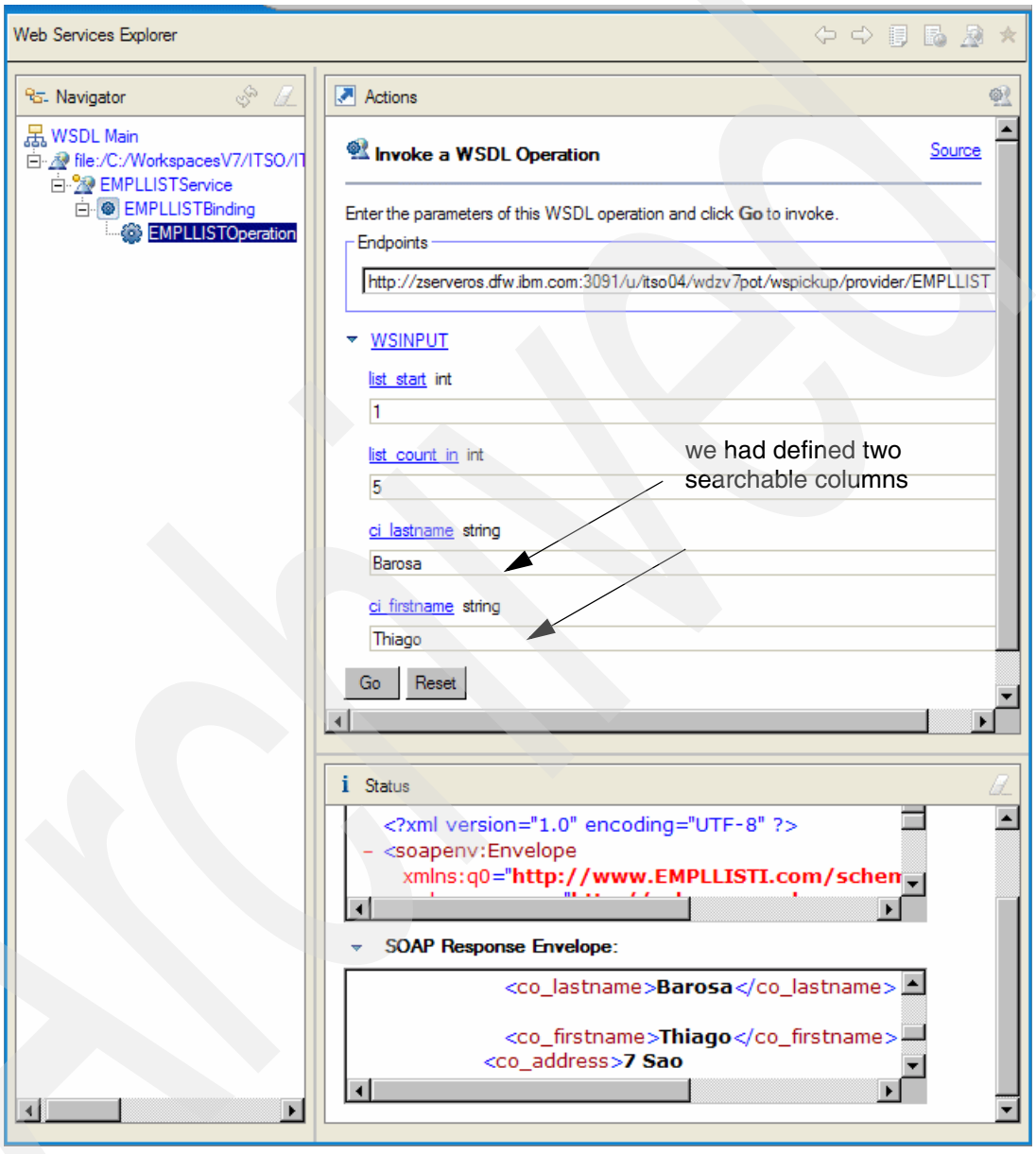


Figure 16-64 Testing the CICS Web service EMPLLIST

Archived

Part 3

Appendixes

Archived

© Copyright IBM Corp. 2009. All rights reserved.

Archived

A

Source script for the build engine

This is the full source of a working prototype Perl script that implements the build engine.

Example: A-1 Source for the build engine script

```
#!$CLEARCASE/bin/ccperl.exe
#
# Standalone z/OS build helper script which can also be called from a ClearQuest
stage hook.
# Based on original work by Brandt Onorato, Stephen Seifert, Dale Newby, all of
IBM
#
# David Lawrence, IBM, June 2007
#
use strict;
use File::Basename;
use Getopt::Std;
use File::Copy;
use POSIX qw(strftime);
use lib "C:\\Program Files\Rational\commmon\lib";
#use CQPerlExt;

my $debug        = 0;   # Set to 1 for verbose debugging output
my $rccbuild     = 'rccbuild.exe';
my $perl_cmd     = "ccperl";
my $myatt; #for retrieving attribute values from CAL
my $myattname; #for retrieving attribute values from CAL
my $attname;
my $i; #loop counter
#
# Element name (member) is passed in from the calling hook
#
my %options;
my ($ccpn) = $ARGV[0];
my ($stage) = $ARGV[1];
```

© Copyright IBM Corp. 2009. All rights reserved.

```
my ($system) = ARGV[2];
my ($subsys) = ARGV[3];

#
my ($vdrive) = "v:"; # NOTE: Change this to reflect your local MVFS mount point.
$ccpn =~ s/\$(\w+)/$1/g;
#
# First retrieve the Top level ClearCase Application object
# Then retrieve the element.
# Process the version name to get just member name and extension for z/OS.
# element passed in from the hook and filter the name accordingly
#
my $CCApp= Win32::OLE->new ("ClearCase.Application") or die "Can't connect to
ClearCase object $!";
my $version = $CCApp->Element($ccpn) or die "Failed to retrieve element";
my $parent = $version->Parent;
my $filepath = $version->Path;
my $membernamebase = uc(basename($filepath));
my @leafname = split (/\./, $membernamebase);
my $parentdir = $parent->Path;
my $name         = uc( $ENV{'USERNAME'} );
my $mem          = uc( @leafname[@leafname-2]);
#
my $logdir       = $parentdir;
$logdir          = "$logdir/$name";
my $rccVerbosity = "-P $logdir/$mem.";
$rccVerbosity    = "$rccVerbosity -V -V -V" if $debug;
my $sysout       = 'RCCEXT=RCCOUT,DISP=(NEW,CATLG,DELETE),SPACE=(32000,(30,30))';
$sysout          = 'RCCEXT=RCCSTD,DISP=(NEW,CATLG,DELETE),SPACE=(32000,(30,30))'
if $debug;

cleanup();

my ( $env, $db2sys, $menv, $os390_hostname, $os390_port, $stage ) =
setEnvironmentVar();
my $jcl = "$mem.jcl";
my $tempDSN = "$env.$name.$mem";
my $OID     = '40 char CC Obj ID for use in load module';
my $ncpycnt = 7;    # number of CICS NewCopy steps
my $lcnt    = 0;    # loop counter
my $scnt    = 10;   # step counter

my $DLL = 'F';
if ( $mem =~ /^\w{4}[S]\w{2,3}$/ ) {    # 5th Char = 'S' (Subroutine)
    $DLL = 'U';
}

my $operation;   # from perspective of mainframe
my $pcFile;
my $indsn;
my $outdsn;
my $rccext;
my $binary;
my $loadlib;
```

```
my %CCattr = getCCattrs();

if ($stage =~ m/IT|ST|MIR1|PROD/) {
print "stage is $action \n";
     die 'Deletion error.'                  if ( s1delete() != 0 );
     die 'Allocation error.'                if ( s1alloc() != 0 );
     $operation = 'RECEIVE';
     $pcFile    = $ccpn;
     if ($ccpn =~ /\.cbl$/) {
        $outdsn = "$env.DSCC$stage.ALL.SOURCE";
        $rccext = 'CBL';
        $binary = 't';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.DBM$/) {
        $CCattr{'DB2'} = 'Y';
        $outdsn = "$env.DSCC$stage.ALL.DBRMLIB";
        $rccext = 'DBM';
        $binary = 'b';
        die 'Transfer error.'                  if ( Transfer() != 0 );
        die 'Error generating Bind Statements' if ( GenBind() != 0 );
        die 'Bind Error'                       if ( BindProg() != 0 );
     }
     elsif ($ccpn =~ /\.DBG$/) {
        $outdsn = "$env.DSCC$stage.ALL.SYSDEBUG";
        $rccext = 'DBG';
        $binary = 'b';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.LST$/) {
        $outdsn = "$env.DSCC$stage.ALL.LIST";
        $rccext = 'LST';
        $binary = 't';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.BT$/) {
        $indsn   = "$tempDSN.BTLOAD";
        $outdsn  = "$env.DSCC$stage.ALL.BATCH.LOAD";
        $rccext  = 'BT';
        $binary  = 'b';
        $loadlib = 'l';
        die 'Transfer error.'                  if ( Transfer() != 0 );
     }
     elsif ($ccpn =~ /\.CS$/) {
        $CCattr{'CS'} = 'Y';
        $indsn   = "$tempDSN.CSLOAD";
        $outdsn  = "$env.DSCC$stage.ALL.CICS.LOAD";
        $rccext  = 'CS';
        $binary  = 'b';
        $loadlib = 'l';
        die 'Transfer error.'                  if ( Transfer() != 0 );
        die 'Error during CICS newcopy'        if ( cicsNewcopy() != 0 );
     }
     elsif ($ccpn =~ /\.SP$/) {
```

Archived

```
        $CCattr{'SP'} = 'Y';
        $indsn   = "$tempDSN.SPLOAD";
        $outdsn  = "$env.DSCC$stage.ALL.DB2.LOAD";
        $rccext  = 'SP';
        $binary  = 'b';
        $loadlib = 'l';
        die 'Transfer error.'                  if ( Transfer() != 0 );
        die 'Error during SP copy'             if ( storedProcCopy() > 4 );
      }
    s1delete() if (! $debug);
}
else {
   if  ( $ccpn =~ /\.cbl$/ ) {
      die 'Deletion error.'                 if ( s1delete() != 0 );
      die 'Allocation error.'               if ( s1alloc() != 0 );
   }
  %CCattr = genCCattrs() if (! %CCattr );

# copy source to mainframe
      $operation = 'RECEIVE';
      $outdsn    = "$env.DSCC$stage.ALL.SOURCE";
      $pcFile    = $filepath;
      $rccext    = 'CBL';
      $binary    = 't';
      die 'Transfer error.'                 if ( Transfer() != 0 );

# copy derived objects to pc
# CO/CI of derived objects always done in post-op (use workarea $logdir)
      $operation = 'XMIT';

      print "CICS Attribute is $CCattr{'CICS'} \n";
      if ( $CCattr{'CICS'} eq 'Y' ) {
             print "CICS XMIT, ATTR is $CCattr{'CS'} \n";
         die 'Translation error.'           if ( CicsTranslate() > 4 );
      }
      if ( $CCattr{'DB2'} eq 'Y' ) {
         die 'Precompile error.'            if ( DB2Precompile() > 4 );
         $outdsn = "$env.DSCC$stage.ALL.DBRMLIB";
         $pcFile = "$logdir\\$mem.DBM";
         $rccext = 'DBM';
         $binary = 'b';
         die 'Transfer error.'              if ( Transfer() != 0 );
      }
      die 'Compile error.'                  if ( CobCompile() > 4 );
      $outdsn = "$env.DSCC$stage.ALL.SYSDEBUG";
      $pcFile = "$logdir\\$mem.DBG";
      $rccext = 'DBG';
      $binary = 'b';
      die 'Transfer error.'                 if ( Transfer() != 0 );
      $outdsn = "$env.DSCC$stage.ALL.LIST";
      $pcFile = "$logdir\\$mem.LST";
      $rccext = 'LST';
      $binary = 't';
      die 'Transfer error.'                 if ( Transfer() != 0 );
      $outdsn = "$env.DSCC$stage.ALL.SIDEDECK";
```

Archived

```
      $pcFile = "$logdir\\$mem.SDK";
      $rccext = 'SDK';
      $binary = 't';
      die 'Transfer error.'                  if ( Transfer() != 0 );
      die 'Prelink Error.'                   if ( Prelink() > 4 );
      die 'Link Error.'                      if ( LinkProg() != 0 );
      if ( $CCattr{'DB2'} eq 'Y' ) {
         die 'Error generating Bind Statements' if ( GenBind() != 0 );
         die 'Bind Error'                       if ( BindProg() != 0 );
      }
      if ( $CCattr{SP} eq 'Y' ) {
         die 'Error during SP copy'          if ( storedProcCopy() != 0 );
      }
      if ( $CCattr{CS} eq 'Y' ) {
         die 'Error during CICS newcopy'     if ( cicsNewcopy() != 0 );
      }
   s1delete() if (! $debug);
}


exit(0);

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< writeFile >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub writeFile {
    my ( $outputfilename, $text ) = @_;

    print "OUTPUTFILE: $outputfilename \n" if $debug;
    open( FOUT, ">> $outputfilename" ) or die 'Unable to open output file';

    my $n = length($text);
    my $first = substr( $text, 0, 1 );
    my $last;
    if ( $n > 2 ) {
        $last = substr( $text, $n - 2, 2 );
        if ( $last eq "\n\n" ) {
            chop($text);
        }
    }
    if ( $first eq "\n" ) {
        $text = substr( $text, 1 );
    }
    print $text if $debug;
    print FOUT $text;

    close(FOUT);
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< getCCattrs >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub getCCattrs {
#
# Use CAL to retrieve Attributes
#
#my $elt = $version->Element;
my $attlist = $version->Attributes;
my $attcount = $attlist->Count;
```

Archived

```
for ($i=1; $i<=$attcount; $i++) {
   $myatt = $attlist->Item($i);
   $myattname = $myatt->Type->Name;
   $CCattr{$myattname} = $myatt->Value;
 }

    return %CCattr;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< genCCattrs >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub genCCattrs {
   my %CCattr;

   $CCattr{'DB2'}  = 'N';
   $CCattr{'CICS'} = 'N';
   $CCattr{'FLOW'} = 'N';
   $CCattr{'MQ'}   = 'N';
   $CCattr{'SP'}   = 'N';
   $CCattr{'CS'}   = 'N';
   $CCattr{'BT'}   = 'N';

   if ( substr( $mem, 5, 1 ) eq 'T' ) {
      $CCattr{'DB2'} = 'Y';
      $CCattr{'SP'}  = 'Y';
   }

my $version = $CCApp->Version($ccpn) or die "Failed to retrieve VOB element";
   open (IN, "$version") || die("Error Reading File: $ccpn $!");
   undef $/;
   my $infile = <IN>;
   close (IN) || die("Error Closing File: $ccpn $!");

   if ( $infile =~ / SQL / ) {
      $CCattr{'DB2'} = 'Y';
   }

   if ( $infile =~ / CICS / ) {
      $CCattr{'CICS'} = 'Y';
      $CCattr{'CS'}   = 'Y';
   }

   if ( $infile =~ /(SCRCODE|WFCODE)/ ) {
      $CCattr{'CICS'} = 'Y';
      $CCattr{'FLOW'} = 'Y';
   }

   if ( $infile =~ /MQOPEN/ ) {
      $CCattr{'MQ'} = 'Y';
   }

   if ( ($CCattr{'SP'} eq 'N') && ($CCattr{'CS'} eq 'N') ) {
      $CCattr{'BT'} = 'Y';
   }
```

Archived

```
#
# Now set the attributes from the hash created above
#
my $elt = $version->Element;
my $eltype = $elt->ElementType;
my $attlist = $elt->Attributes;
my $attcount = $attlist->Count;

for ($i=1; $i <=$attcount; $i++) {
   $myatt= $attlist->Item($i);
   $myattname = $myatt->Type->Name;
   $myatt->SetValue($CCattr{$myattname});
  }

   return %CCattr;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< setEnvironmentVar
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub setEnvironmentVar {
    my @env;    # $env, $db2sys, $menv, $os390_hostname, $os390_port, $stage
    if ($stage =~ /DEV/i ) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'TEST', 'DBT1', 'KCT', '83.255.29.22', '3606', 'DEV' );
    }
    elsif ($stage =~ /IT/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCA', 'DBD1', 'KCA', '83.255.29.22', '10003', 'IT' );
    }
    elsif ($stage =~ /ST/i) {
   printf ("stage was - $stage \n") if $debug;
        @env = ( 'CCS', 'DBD1', 'KCS', '83.255.29.22', '10003', 'ST' );
    }
    elsif ($stage =~ /REL/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCE', 'DBP1', 'KCE', '83.255.29.22', '10003', 'REL' );
    }
    elsif ($stage =~ /MIR1/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCP', 'DBP1', 'KCP', '83.255.29.22', '10003', 'MIR1' );
    )
    elsif ($stage =~ /PROD/i) {
   printf ("stage was - $stage\n") if $debug;
        @env = ( 'CCP', 'DBP1', 'KCP', '83.255.29.22', '10003', 'PROD' );

    }
    else {
        printf ("Unknown  stage - $stage\n") if $debug;
        exit -1;
    }

    printf( "Environment name (%s)\n", @env ) if $debug;
    return @env;
}
```

Archived

```
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< cleanup >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub cleanup {
    mkdir $logdir;
    unlink <$logdir\\$mem.*>;
    unlink 'rccbuild.log',$jcl;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< s1delete >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub s1delete {
    writeFile(
        $jcl, "
//* ---------------------------------------------------------------------
//DELETE   EXEC PGM=IDCAMS
//*        ----
//*        ---- IDCAMS - Delete Temporary Files
//*        ----
//SYSPRINT DD   $sysout
//SYSIN    DD   *
 DELETE $tempDSN.* NONVSAM
 ;
 IF LASTCC LE 8 THEN SET MAXCC EQ 0
/*
//* ---------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b S1DELETE -ft $jcl
-n 0 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< s1alloc >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub s1alloc {
    writeFile(
        $jcl, "
//* ---------------------------------------------------------------------
//S1ALLOC  EXEC PGM=IEFBR14
//*        ----
//*        ---- IEFBR14 - Allocate Temporary Files
//*        ----
//TEMP1    DD  DSN=$tempDSN.DSNHOUT,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(800,(500,500))
//TEMP2    DD  DSN=$tempDSN.LINKPARM,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(400,(20,20),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
//TEMP3    DD  DSN=$tempDSN.PLKSET,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//TEMP4    DD  DSN=$tempDSN.LOADSET,
```

Archived

```
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(800,(1500,1500),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//TEMP5    DD  DSN=$tempDSN.ELMOUT,RCCEXT=CBL,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(6160,(100,100),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//TEMP6    DD  DSN=$tempDSN.LNKLIST,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=3390,SPACE=(3630,500,500),RLSE),
//             DCB=(RECFM=FB,LRECL=121,BLKSIZE=3630)
//TEMP7    DD  DSN=$tempDSN.COBLIST,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=3390,SPACE=(6171,7000,7000),RLSE),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//TEMP8    DD  DSN=$tempDSN.PCLIST,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=3390,SPACE=(6171,(500,500),RLSE),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//TEMP9    DD  DSN=$tempDSN.LOADNAME,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(CYL,(5,5,5)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b S1ALLOC -ft $jcl -n
0 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< DB2Precompile >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub DB2Precompile {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//PRECOMP  EXEC PGM=DSNHPC,REGION=4096K,
//  PARM=('HOST(IBMCOB),APOST')
//*        ----
//*        ---- DSNHPC - Precompile
//*        ----
//STEPLIB  DD  DISP=SHR,DSN=SYS1.DSN710.SDSNEXIT
//         DD  DISP=SHR,DSN=DSN710.SDSNLOAD
//SYSLIB   DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1STAGE.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1ST2.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.ST.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.REL.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.MIR1.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.PROD.BCOPY
"
    );
```

Archived

```
    if ( $CCattr{'STG1'} eq 'DEV' ) or ( $CCattr{'STG1'} eq 'IT' {
        writeFile(
            $jcl, "
//DBRMLIB  DD  DISP=SHR,DSN=ENDV.&C1SYSTEM.&C1SUBSYS.&C1STAGE.BDBRM
"
            );
        }
    if ( $CCattr{'STG1'} eq 'ST' ) or ( $CCattr{'STG1'} eq 'REL' {
        writeFile(
            $jcl, "
//DBRMLIB  DD  DISP=SHR,DSN=ENDV.NOVA.&C1SUBSYS.&C1STAGE.BDBRM
"
            );
        }
        writeFile(
            $jcl, "
//SYSIN    DD  DISP=SHR,DSN=$tempDSN.ELMOUT($mem)
"
        );
    }

    writeFile(
        $jcl, "
//SYSCIN   DD  DISP=SHR,DSN=$tempDSN.DSNHOUT
//SYSPRINT DD  $sysout
//SYSTERM  DD  $sysout
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b PREDB2 -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< CobCompile >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub CobCompile {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//COMPILE  EXEC PGM=IGYCRCTL,REGION=4096K,
//  PARM=('LIB,APOST,SOURCE,LIST,TEST,RMODE(AMY),MAP')
"
    );

    writeFile(
        $jcl, "
//*        ----
//*        ---- Compile - COBOL compile
```

Archived

```
//*        ----
//STEPLIB  DD  DISP=SHR,DSN=IGY.SIGYCOMP
//SYSIN    DD  DISP=SHR,DSN=$tempDSN.DSNHOUT
"
    );
    writeFile(
        $jcl, "
//SYSLIN   DD  DISP=SHR,DSN=$tempDSN.LOADSET($mem)
//SYSLIB   DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1STAGE.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.&C1STAGE.&C1ST2.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.ST.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.REL.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.MIR1.BCOPY
//         DD  DISP=SHR,DSN=ENDV.NASE.&C1SUBSYS.PROD.BCOPY
"
    );

    writeFile(
        $jcl, "
//SYSPRINT DD  DISP=SHR,DSN=$tempdsn.COBLIST,
//SYSTERM  DD  $sysout
//SYSUT1   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT3   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT4   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT5   DD  UNIT=SYSDA,SPACE=(3000,(500,500),,,ROUND)
//SYSUT6   DD  UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT7   DD  UNIT=SYSDA,SPACE=(3000,(500,500),,,ROUND)
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b COMPILE -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}


# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< Prelink >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub Prelink {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//PLKED    EXEC PGM=EDCPRLK,REGION=2048K
//STEPLIB  DD DSN=CEE.SCEERUN,DISP=SHR
"
    writeFile(
        $jcl, "
//*        ----
//*        ---- EDCPRLK - Prelink
//*        ----
//SYSIN    DD  DISP=SHR,DSN=$tempDSN.LOADSET($mem)
```

```
//         DD  DISP=SHR,DSN=CCT.DSCC.ALL.SIDEDECK($mem)
"
    );

    writeFile(
        $jcl, "
//SYSMSGS  DD  DISP=SHR,DSN=CEE.SCEEMSGP(EDCPMSGE)
//SYSLIB   DD  DUMMY
//SYSDEFSD DD  DUMMY
//SYSMOD   DD  DISP=SHR,DSN=$tempDSN.PLKSET
//SYSPRINT DD  DSN=$tempDSN.LNKLIST,DISP=SHR
//SYSOUT   DD  $sysout
//SYSTERM  DD  SYSOUT=*
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b PRELINK -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< LinkProg >>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub LinkProg {
    my $status = 0;

    $OID = `cleartool lshistory -last 1 -fmt \"%1.400n\" $ccpn\@\@`;
    printf( "xtendedName (%s)\n", $OID ) if $debug;

    if ( $CCattr{'SP'} eq 'Y' ) {
       die 'LinkSP Error.'            if ( linkSP() != 0 );
       $indsn   = "$env.DSCC$stage.ALL.DB2.LOAD";
       $outdsn  = "$tempDSN.SPLOAD";
       $pcFile  = "$logdir\\$mem.SP";
       $rccext  = 'SP';
       $binary  = 'b';
       $loadlib = 'l';
       die 'Transfer error.'          if ( Transfer() != 0 );
    }
    if ( $CCattr{'BT'} eq 'Y' ) {
       die 'LinkBT Error.'            if ( linkBT() != 0 );
       $indsn   = "$env.DSCC$stage.ALL.BATCH.LOAD";
       $outdsn  = "$tempDSN.BTLOAD";
       $pcFile  = "$logdir\\$mem.BT";
       $rccext  = 'BT';
       $binary  = 'b';
       $loadlib = 'l';
       die 'Transfer error.'          if ( Transfer() != 0 );
    }
    if ( $CCattr{'CS'} eq 'Y' ) {
       die 'LinkCS Error.'            if ( linkCS() != 0 );
       $indsn   = "$env.DSCC$stage.ALL.CICS.LOAD";
```

Archived

```
        $outdsn  = "$tempDSN.CSLOAD";
        $pcFile  = "$logdir\\$mem.CS";
        $rccext  = 'CS';
        $binary  = 'b';
        $loadlib = 'l';
        die 'Transfer error.'           if ( Transfer() != 0 );
    }
    $loadlib = '';

    return $status;
}

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< buildLoadname >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub buildLoadname {
    my $status = 0;

    $jcl = 'devprint.jcl';

    cofkymod::writeFile( $jcl, $compile_1, "
//* ---------------------------------------------------------------
//LDNAME   EXEC PGM=IEBGENER
//*        ----
//*        ---- IEBGENER - Create Loadname entry
//*        ----
//SYSUT1   DD  *,DCB=BLKSIZE=80
  INCLUDE SYSLIB(SBCTDBC)
  NAME $mem(R)
/*
//SYSUT2   DD  DISP=SHR,DSN=$tempDSN.LOADNAME($mem)
//SYSPRINT DD  $sysout
//SYSIN    DD  DUMMY
//* ---------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b buildLoadname -ft
$jcl -n 4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}


# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< linkBT >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub linkBT {
    writeFile(
        $jcl, "
//* ---------------------------------------------------------------
//LKEDBT   EXEC PGM=IEWL,
//    PARM='MAP,LIST,AMODE(31),RMODE(ANY)'
//*        ----
//*        ---- IEWL - Linkedit for Batch
//*        ----        USE DSNELI FOR TSO BATCH
//*        ----          , DSNRLI FOR SP
```

Archived

```
//*        ----          , DSNCLI FOR CICS DB2
//*        ----
//SYSLIB   DD  DISP=SHR,DSN=CEE.SCEELKED
//         DD  DISP=SHR,DSN=DSN710.SDSNLOAD
//         DD  DISP=SHR,DSN=ISP.SISPLOAD
//         DD  DISP=SHR,DSN=ENDV.NOVA.BASE.PROD.OLOAD
//         DD  DISP=SHR,DSN=ENDV.&C1SYSTEM.&C1SUBSYS.&C1STAGE.BLOAD
"
    );
    writeFile(
        $jcl, "
//SYSLIN   DD  DISP=SHR,DSN=$tempDSN.PLKSET
//         DD  DISP=SHR,DSN=$tempDSN.LOADNAME($mem)
"
    );
    writeFile(
        $jcl, "
//SYSLMOD  DD  DISP=SHR,DSN=ENDV.&C1SYSTEM.&C1SUBSYS.&C1STAGE.BLOAD($mem)
//SYSPRINT DD  DISP=SHR,DSN=$tempDSN.LNKLIST
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,50))
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b LINKBATCH -ft $jcl
-n 4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}

# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< linkCS >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub linkCS {
    writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//LKEDCS   EXEC PGM=IEWL,
//    PARM='RENT,MAP,AMODE(31),RMODE(ANY)'
//*        ----
//*        ---- IEWL - Linkedit for CICS
//*        ----       USE DSNELI FOR TSO BATCH
//*        ----         , DSNRLI FOR SP
//*        ----         , DSNCLI FOR CICS DB2
//*        ----
//SYSLIB   DD  DISP=SHR,DSN=SYS1.LINKLIB
//         DD  DISP=SHR,DSN=$env.DSCC.ALL.CICS.LOAD
//         DD  DISP=SHR,DSN=$env.DSCC.ALL.BATCH.LOAD
//         DD  DISP=SHR,DSN=MCS.$menv.CICS.LOAD
//         DD  DISP=SHR,DSN=MCS.$menv.BATCH.LOAD
//         DD  DISP=SHR,DSN=SYS1.CEE.SCEELKED
"
    );

    if ( $CCattr{'MQ'} eq 'Y' ) {
```

```
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS2.MQSERIES.SCSQLOAD
"
        );
    }

    writeFile(
        $jcl, "
//         DD  DISP=SHR,DSN=SYS2.CICSTS13.SDFHLOAD
//*        DD  DISP=SHR,DSN=SYS2.CICSTS13.SDFHEXCI
//         DD  DISP=SHR,DSN=SYS1.CEE.SCEECICS
//         DD  DISP=SHR,DSN=SYS2.DEBUG.SEQAMOD
"
    );

    if ( $CCattr{'FLOW'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS2.TST321G3.SFMCLOAD
//         DD  DISP=SHR,DSN=SYS2.MQWF321.UTILITY.SFMCLOAD
//         DD  DISP=SHR,DSN=SYSRUN.MQWF321.SFMCLOAD
"
        );
    }

    if ( $CCattr{'DB2'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=DB2.$db2sys.SDSNEXIT
//         DD  DISP=SHR,DSN=DB2.$db2sys.SDSNLOAD
"
        );
    }

    if ( $CCattr{'CICS'} eq 'N' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS.PROD.LOAD
"
        );
    }

    writeFile(
        $jcl, "
//SYSLIN   DD  DISP=SHR,DSN=$tempDSN.PLKSET
"
    );

    if ( $CCattr{'DB2'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INDB2CIC)
"
        );
```

Archived

```
    }

    writeFile(
        $jcl, "
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INCICSCM)
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INCICSDB)
"
    );

    if ( $CCattr{'MQ'} eq 'Y' ) {
        writeFile(
            $jcl, "
//         DD  DISP=SHR,DSN=SYS2.TCKCT.SYSEXEC(INMQCICS)
"
        );
    }

    writeFile(
        $jcl, "
//         DD  *,DCB=BLKSIZE=80
 IDENTIFY $mem('$OID')
 NAME $mem(R)
/*
//SYSLMOD  DD  DISP=SHR,DSN=$env.DSCC$stage.ALL.CICS.LOAD
//SYSPRINT DD  $sysout
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(50,50))
//* ------------------------------------------------------------------
"
    );
    my $cmd    = "$rccbuild -h $os390_hostname\@$os390_port -b LNKCICS -ft $jcl -n
4 -c LE $rccVerbosity";
    my $status = system($cmd);
    printf( "Status was (%d)\n", $status ) if $debug;
    move( $jcl, "$logdir/$jcl.$scnt" );
    $scnt++;
    return $status;
}
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< Transfer >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
sub Transfer {
   die "Unsupported operation" if (! ( $operation =~ /(XMIT|RECEIVE)/ ));
   my $cmd;

   if ($operation eq 'RECEIVE') {
      writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//XFERO$scnt EXEC PGM=IEFBR14
//*        ----
//*        ---- Clearcase upload -i
//*        ----
"
);
      if ($loadlib) {
         writeFile(
           $jcl, "
```

```
//DD1      DD  DISP=SHR,DSN=$indsn,RCCEXT=$rccext
//* ------------------------------------------------------------------
"
);
      }
      else {
         writeFile(
           $jcl, "
//DD1      DD  DISP=SHR,DSN=$outdsn,RCCEXT=$rccext
//* ------------------------------------------------------------------
"
);
      }
      $cmd =
"$rccbuild -h $os390_hostname\@$os390_port -b XFER -ft $jcl -i$binary $pcFile -n 4
-c LE $rccVerbosity";
   }

   if ($loadlib) {
      writeFile(
           $jcl, "
//* ------------------------------------------------------------------
//XFER1$scnt EXEC PGM=IKJEFT01
//*        ----
//*        ---- XMIT / RECEIVE
//*        ----
//INDD     DD  DISP=SHR,DSN=$indsn
//OUTDD    DD  DISP=SHR,DSN=$outdsn
//SYSPRINT DD  $sysout
//SYSTSPRT DD  $sysout
//SYSTSIN  DD  *
"
      );
      if ($operation eq 'XMIT') {
         writeFile(
           $jcl, "
 XMIT A.A DDNAME(INDD) MEMBER($mem) OUTDDNAME(OUTDD) NOLOG
"
         );
      }
      else {
         writeFile(
           $jcl, "
 RECEIVE INDDNAME(INDD) LOGDATASET($tempDSN.XFER.LOG)
"
         );
      }

      writeFile(
           $jcl, "
/*
//* ------------------------------------------------------------------
"
      );
   }
```

Archived

```
   if ($operation eq 'XMIT') {
      writeFile(
        $jcl, "
//* ------------------------------------------------------------------
//XFER9$scnt   EXEC PGM=IEFBR14
//*        ----
//*        ---- Clearcase download -o
//*        ----
"
      );
      if ($loadlib) {
         writeFile(
           $jcl, "
//DD1      DD  DISP=SHR,DSN=$outdsn,RCCEXT=$rccext
"
         );
      }
      else {
         writeFile(
           $jcl, "
//DD1      DD  DISP=SHR,DSN=$outdsn($mem),RCCEXT=$rccext
"
         );
      }
      writeFile(
        $jcl, "
//* ------------------------------------------------------------------
"
      );
      $cmd = "$rccbuild -h $os390_hostname\@$os390_port -b XFER -ft $jcl -o$binary
$pcFile -n 4 -c LE $rccVerbosity";
   }
   my $status = system($cmd);
   printf( "Status was (%d)\n", $status ) if $debug;
   move( $jcl, "$logdir/$jcl.$scnt" );
   $scnt++;
   return $status;
}
```

Archived

B

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG247482

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247482.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
SG247482-Chapter 1.zip	Scenario Code
SG247482-Chapter 6.zip	Scenario Code
SG247482-Chapter 7.zip	Scenario Code
SG247482-Chapter 10.zip	Scenario Code
SG247482-Chapter 14.zip	Scenario Code

© Copyright IBM Corp. 2009. All rights reserved.

SG247482-Chapter 15.zip
Scenario Code
SG247482-Chapter 16.zip
Scenario Code
SG247482_Viewlet_SCENARIO_1.zip
Audio self-paced overview of Scenario 1
SG247482_Viewlet_SCENARIO_2.zip
Audio self-paced overview of Scenario 2

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 200 MB minimum
Operating System: Windows XP
Processor: Pentium® 4 or higher
Memory: 2 GB

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into the folder.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 648. Note that some of the documents referenced here may be available in softcopy only.

- *Collaborative Application Lifecycle Management with IBM Rational Products*, SG24-7622
- *ACI Worldwide's BASE24-eps V6.2: A Supplement to SG24-7268*, REDP-4338
- *IBM IT Governance Approach: Business Performance through IT Execution*, SG24-7517
- *Rational Business Driven Development for Compliance*, SG24-7244
- *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM*, SG24-6399
- *Strategic Reuse with Asset-Based Development*, SG24-7529
- *Using WebSphere Studio Asset Analyzer*, SG24-6065

Other publications

These publications are also relevant as further information sources:

- *Debug Tool for z/OS Debug Tool Utilities and Advanced Functions for z/OS User’s Guide*, SC19-1196
- *Enterprise COBOL for z/OS Language Reference*, SC23-8528
- *Enterprise COBOL for z/OS Programming Guide*, SC23-8529
- *File Manager for z/OS User's Guide and Reference for CICS*, SC19-1242
- *IBM Application Performance Analyzer for z/OS User's Guide*, SC23-8512
- *IBM Fault Analyzer for z/OS User's Guide and Reference*, SC19-1253
- *IBM File Manager for z/OS User's Guide and Reference*, SC19-1239
- *Workload Simulator for z/OS and OS/390 User's Guide*, SC31-8948

Online resources

These Web sites are also relevant as further information sources:

- General information about COBOL compilers

  `http://www.ibm.com/software/awdtools/cobol/`
- Release history of various COBOL compilers

  `http://www.ibm.com/servers/eserver/zseries/zos/le/history/history.html`

© Copyright IBM Corp. 2009. All rights reserved.

- General information about PLI compilers

  http://www.ibm.com/software/awdtools/pli/
- Using WebSphere Studio Asset Analyzer to find mainframe Web Services

  http://www.ibm.com/developerworks/websphere/library/techarticles/0808_miller/0808_miller.html
- WebSphere Studio Asset Analyzer

  http://www.ibm.com/software/awdtools/wsaa
- Unified Change Management

  http://www.ibm.com/developerworks/rational/library/701.html
- Unified Change Management from Rational Software: An Activity-Based Process for Managing Change

  http://www.ibm.com/developerworks/rational/library/2057.html
- Modernizing enterprise application development with integrated change, build, and release management.

  http://www-07.ibm.com/my/events/baitsolution/download/RAW14001-USEN-00.pdf
- developerWorks Interviews: Build-and-release best practices

  http://www.ibm.com/developerworks/podcast/dwi/cm-int111307txt.html
- Rational ClearCase

  http://www-01.ibm.com/software/awdtools/clearcase/
- Rational ClearQuest

  http://www-01.ibm.com/software/awdtools/clearquest/
- Rational Build Forge

  http://www-01.ibm.com/software/awdtools/buildforge/
- Rational Developer for System z

  http://www-01.ibm.com/software/awdtools/rdz/

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Numerics

0C7 50
  with Fault Analyzer 52
64-bit DB2 Universal Database 38

A

action record 418
adapt the generated JCL procedures for your z/OS system 590
add the Faces JSP controls that will invoke the Web service 609
advanced functions 270
allocating PDS on z/OS 572
alternative views 484
ANSI 85 standard 37
application dependency 108
application life cycle 93
Application Performance Analyzer for z/OS 33
applying a transformation to the UML elements 617
assigning element level attributes 73
assigning project level attributes 75

B

batch 132
batch processor 167
batch program 276
batch reanalysis 40
bidirectional considerations 583
BMS maps 531
build 104
build a J2C application to invoke the COBOL/CICS Serve 506
Build Forge 97
Build Forge with Remote Build 96
  Build Forge 97
  Build Forge agent 97
  Build Forge local agent 99
build management 71
build strategy 78
building host applications with a Build Forge local agent 99

C

CARMA 391
CARMA and SCLM Developer Toolkit 468
CARMA APIs to manage runtime messages 401
CARMA basics 392
CARMA out of box solutions 394
CARMA RAM 411
CARMA RAM functions 419
CARMA session 434
changing z/OS Database Application Transformation Editor Layout 576
checking syntax 6
CICS application 133, 295
CICS CEDX to trace transaction CPIH 602
CICS ECI Adapter 5.1 491
CICS ECI Adapter 6.0.2 491
CICS ECI connector using JSP test client 514
CICS infrastructure 146
CICS PIPELINE 598
CICS PIPELINE resource definition 596
CICS TCPIPSERVICE resource definition 598
CICS Transaction Server for z/OS 153
CICS Web Services on z/OS 145, 594
CICS WEBSERVICE 600
ClearCase 73
ClearCase managed artifacts 100
ClearCase TSO Client 87
ClearCase with remote build on UNIX System Services 86
ClearCase with remote build on z/OS 82
  build on UNIX System Services 86
  hints 86
  host-Build.pl script 84
  managed builds - build engine 82
  managed builds - simple scenario 82
  security issues 86
ClearQuest to manage the application life cycle 93
COBOL 35
COBOL driver programs 584
COBOL that performs the CRUD operation 585
COBOL that performs the list operation 587
COBOL/CICS programs 531
COBOL/CICS with VSAM/SFS 538
code on z/OS and no CICS or DB2 15
code on z/OS that uses CICS but not DB2 23
coding phase 122
Column Maps 45
compile converter file 156
compile/link and create the executables 500
compiled XML conversion 165
configure CTG to communicate with TXSeries 497
configure TXSeries to support CTG 497
configuring CARMA RAM functionality 418
configuring CICS Web Services on z/OS 145, 594
configuring the CICS settings 577
configuring the DB2 Configuration columns 577
configuring the JCL job parameters 577
configuring the MVS settings 579
configuring the transformation 618
configuring the UNIX System Services settings 579
configuring the z/OS DB2 Tables Application 576
connect to existing COBOL/CICS programs 490
connecting to a System z 474
connecting to a z/OS remote system 572
connecting to repository 102

© Copyright IBM Corp. 2009. All rights reserved.

connecting to the defined system 480
connection Issues 92
considerations for using the TSO Client 88
controlled elements 103
Convert Process 46
converter file 156
copying the COBOL programs to the z/OS 591
copying the JCL generated procedures to the z/OS 592
copying the WSBIND file to the z/OS 595
CPU usage 36
create a CRUD and list applications 572
create a data design project and load the DDL file 569
create a Java bean that invokes the COBOL/CICS Server 506
create a Web service that invokes the J2C Java bean 516
create and load the DB2 table 568
create custom menu items 234
Create Object Definitions 45
create PIPELINE resource definition 150
Create Process 46
create Service Flow project 205
create TCPIPSERVICE resource definition 153
creating a CICS PIPELINE resource definition 596
creating a CICS TCPIPSERVICE resource definition 598
creating a DB2 table to be used by the Database Application Generator 567
creating a repository 76
creating a Simple Project 615
creating a Simple Project and Importing the UML Model 615
creating a transformation parameters file 616
creating a Web project with a simple JSP page 608
creating a Web service client using Faces 607
creating the DB2 table in the z/OS system 622
creating the z/OS HFS directories 595
customize the JCL used by the RDO definitions 591

D

Database Application Generator 563, 566
Database Application Generator preferences 574
Database Application Generator project 132
Database Application Generator wizard from UML 615
Database Application Transformation Editor Layout 576
DB2 35, 39
DB2 BIND 590
DB2 Catalog 43
DB2 Configuration columns 577
DB2 Data Design Project 568
DB2 has the required sample table 494
DB2 stored procedures builder for PL/I and COBOL 338
DB2 table to be used by the Database Application Generator 567
DB2 UNLOAD utility 41
DDL file 569
Debug Tool for z/OS and Debug Tool Utilities and Advanced Functions 33
Debug Tool Utilities 37
debugging a batch program 276
debugging a CICS application 295
debugging a IMS application 318
debugging COBOL/CICS with VSAM/SFS 538
define build and promote strategy 78
define WSDL file 195
defining a system connection 474
defining a WD/z z/OS remote system 572
deploy application to IMS SOAP Gateway 198
deploy generated assets to CICS Transaction Server for z/OS 153
deploy Web service binding file 154
deploy Web service converter source file 155, 193
Deploy Web service provide 160
deploy WSDL file and the correlator file to IMS SOAP Gateway 194
deploying the code generated on Scenario 2 623
deploying the generated assets to z/OS 591
deploying the Web service to IMS SOAP Gateway 194
determine file system hierarchy 77
developer productivity 122
developing and debugging a batch program 276
developing and debugging a CICS application 295
developing and debugging a IMS application 318
developing single resources or in projects 487
DSNTIAUL 41

E

Eclipse platform 47
editing a transformation parameters file 616
editing using LPEX 254
emulate a mainframe on PC 121
enablement for an existing IMS application 179
end to end debug 123
Enterprise Application Lifecycle and Rational Developer for System z
  build 104
  ClearCase managed artifacts 100
  connecting to your repository 102
  controlled elements 103
  migration 104
  promotions 104
  very brief overview 100
Enterprise Application Lifecycle Management 99
Enterprise COBOL 38
enterprise repository 71
  build management 71
  identification of assets 72
  organization of assets 72
  using ClearCase meta data in z/OS builds 73
Enterprise Service Tool perspective 126
Enterprise Service Tool single-service project 130
Enterprise Service Tools 125
Enterprise Service Tools projects 129
Enterprise Software Configuration Management implementation 69
existing z/OS DB2 table 566
exporting the UML model using Rational Software Architect 613
extending CARMA 394

F

Faces JSP controls that will invoke the Web service 609
Fault Analyzer
  Artefact 52
Fault Analyzer for z/OS 33
Fault Analyzer plug-in for WD/z technical preview 362
Fault History details 53
File Manager for z/OS 33
File Manager plug-in for WD/z 380
file system hierarchy 77
function implementations 422
functional SCM model 416

G

generated assets 584
generated assets to CICS Transaction Server for z/OS 153
generated assets to your z/OS system 154
generated assets to z/OS 591
generated JCL 589
generated JCL procedures for your z/OS system 590
generating SFS file 524
generation of the Web enablement components 182
generation of Web service enablement components 135

H

HFS data sets 41
HFS directories 146
Host-Build.pl script 84

I

IBM Fault Analyzer for z/OS 38
  Analysis options 40
  Support for IBM subsystems 39
IBM File Manager for z/OS 40
  components 41
  highlights 40
IBM Parallel Sysplex Coupling Facility 35
IBM Tivoli Enterprise Portal 35
IBM Tivoli OMEGAMON 35
IDE based Java/J2EE application development 403
identification of assets 72
impact analysis 107, 114
implementation of CARMA RAM 411
implementation of checkIn 429
implementation of getAllMemberInfo 425
implementation of getInstance 422
implementing CARMA RAM functions 419
implementing RAM 417
importers 492
importing assets into IBM Rational ClearCase 79
importing the UML into your project 615
IMS 35, 39
IMS application 318
IMS SOAP Gateway 194, 198
IMS Soap Gateway 180
IMS SOAP Gateway project 132
initial view 470
Insert Process 45
Insert Processing 45
inspecting the CICS WEBSERVICE and URIMAP 600
install CICS Transaction Gateway (CTG 496
install the generated programs and map into TXSeries 503
install TXSeries Version 6.1 and create a region 497
install WebSphere Developer for System z 495
installing the CICS PIPELINE and TCPIPSERVICE 598
installing the z/OS CICS services 597
interaction properties for the application 196
Interactive reanalysis 40
interactive remote debug 123
Interactive System Productivity Facility (ISPF) 37
interpretive versus programmatic generated adapters 165
interpretive XML conversion 166
ISPF 120
ISPF editing versus LPEX editing 251
ISPF profile within LPEX Editor 252

J

J2C 492
J2C application to invoke the COBOL/CICS Server 506
J2C Connection Factory Pool in WebSphere Application Server 520
J2C wizard 492
Java 2 Platform, Standard Edition (J2SE) 36
Java bean that invokes the COBOL/CICS Server 506
Java/J2EE 403
JCL 50
JCL generation 122
JCL job parameters 577
JCL used by the RDO definitions 591
JES 50
JES monitor 123
Job Monitor 464
JOB that does the DB2 BIND 590
JSP Web page 611

L

Load Process 46
load the sample COBOL into WD/z 500
local COBOL 523
local code and no CICS or DB2 7
local code using CICS and DB2 10
local syntax check using local compiler and remote COBOL/CICS program, and copybook with EXEC CICS 29
local syntax check using the local compiler 17
local syntax check with a remote COBOL/CICS program 27
LPEX editing 251

M

making your executable available to your CICS 597
managed and non-managed connection properties 507
managed builds - build engine 82
managed builds - simple scenario 82

managing workspace projects using SCLM DT 405
managing your z/OS assets with ClearCase 80
  access to the ClearCase repository from z/OS 91
  preparing TSO Client 88
  remote build 80
  TSO Client 87
  using ClearQuest to manage application life cycle 93
  using TSO Client 88
mapping 482
Menu Manager 231
Menu Manager preference page 232
meta data 73
migrations 104
modeling considerations 412
modern application development tools 119
modifying COBOL/CICS programs and BMS maps 531
move generated assets to your z/OS system 154
move to local and use the local compiler 18
MVS files properties 245
MVS settings 579

N

non-managed connection properties 507

O

OC7
  With Debug Tool 55
Optim MOVE 33, 43
  Processing Flow 44
  Test Data Management 43
Optim MOVE z/OS 43
optimized code support for COBOL 34
options for the batch processor 168
organization of assets 72
OS/VS COBOL 37

P

parameter record 419
PD Tools
  with RD/z 33
PDS 41
PDS RAM 394
perspectives 470
PIPELINE referencing the pickup directory 599
PIPELINE resource definition 150
port number 56
preparing the workstation 494
preparing to use the TSO Client 88
productivity 122
programmatic access to the ClearCase repository from z/OS 91
  connection issues 92
  recovery from errors 93
  synchronization issues 93
promote strategy 78
promotions 104
Properties Window 50

Q

QSAM 41

R

RAM record 418
Rational Developer for System z 33, 38, 47, 99
Rational Developer for System z V7.1 to work with COBOL programs 3
real-time analysis 40
recovery from errors 93
Redbooks Web site 648
  Contact us xiv
remote build 80
  hints 86
remote build on UNIX System Services 86
remote COBOL/CICS program and copybook with EXEC CICS 29
remote debug 123
remote syntax check using the z/OS compiler 15
remote syntax check using the z/OS compiler with a remote COBOL/CICS program 23
Remote System Explorer (RSE) 464
repository 71
resource adapters 491
restart IMS SOAP Gateway and check the deployment 199
return value 419
returning a z/OS project to online state 21
run batch processor 174
running SQL statements 570
running the transformation 580, 621
runtime code generation and deployment 219

S

S0C7 abend 54
SAM2 57
scan the PIPELINE referencing the pickup directory 599
Scenario 1 - Local code and no CICS or DB2 7
Scenario 1 - Working with the existing z/OS DB2 table 567
Scenario 2 - Local code using CICS and DB2 10
Scenario 2 - Working with an UML Model 613
Scenario 3 - Code on z/OS and no CICS or DB2 15
Scenario 3.1 - Remote syntax check using the z/OS compile 15
Scenario 3.2 - Local syntax check using the local compiler 17
Scenario 3.3 - Working off-line 18
Scenario 4 - Code on z/OS that uses CICS but not DB2 23
Scenario 4.1 - Remote syntax check using the z/OS compiler with a remote COBOL/CICS program 23
Scenario 4.2 - Local syntax check with a remote COBOL/CICS program 27
Scenario 4.3 - Local syntax check using local compiler 29
scenarios 7
SCLM Developer Toolkit V2 402
SCLM project hierarchy 415
SCLM RAM 394

SCLM view 407
SCM data model 412
security issues 86
selecting the available transformations 616
Service Flow 204
Service Flow Modeler 125
Service Flow project 132
Service Flow tooling 203
service-oriented architecture (SOA) 34
SFS file 524
Simple Project and Importing the UML Model 615
Single Web Services enablement styles 130
SNA 46
SOAP 126
SOAP for CICS project 132
Software Configuration Management support in WebSphere Developer for System z V7 389
source script for the build engine 627
specify the options for the batch processor 168
stored procedures builder for PL/I and COBOL 338
submitting the JCL to create the executable 593
synchronization issues 93
syntax checking 122
System z server components 463

T

Table Maps 45
TCP/IP 46
TCPIPSERVICE 598
TCPIPSERVICE resource definition 153
temporary storage queues 42
terminal interactions 210
terminology 126
test deployed service 227
test IMS Web service deployed using WebSphere Developer for System z 200
test provided Client/Server COBOL programs 505
test the CICS ECI connector using JSP test client 514
test the Web service 518
test using the Web Services Explorer 516
test your Web service using WebSphere Developer for System z 163
testing if CTG is communicating with TXSeries 499
testing the JSP Web page 611
testing the Web Services 601
testing the Web Services created 624
traditional and modern application development tools 119
transferring the assets 79
transform existing CICS Application in a Web service 134
transform your IMS Application in a Web service 181
Transformation Parameters Editor 575
transient data queues 42
TSO 132
TSO Client 87
  hints 92
TSO Command Server 467
TSO/ISPF 120
TXSeries 523

U

UML model 566
UML model using Rational Software Architect 613
understanding work style choices 5
UNIX System Services 35, 132
UNIX System Services settings 579
URIMAP 600
using a DB2 Data Design Project to create and load the DB2 table 568
using ClearCase meta data in z/OS builds
  assigning element level attributes 73
  assigning project level attributes 75
  creating your repository 76
  define build and promote strategy 78
  determine file system hierarchy 77
  importing assets into IBM Rational ClearCase 79
  transferring the assets 79
using the CICS CEDX to trace transaction CPIH 602
using the Database Application Generator wizard from UML 615
using the Database Application Generator wizard to create a CRUD and list applications 572
using WD/z to load some data into the DB2 table 604
using WD/z Web Services Test Explorer to invoke the CRUD application 605
using WD/z Web Services Test Explorer to invoke the LIST application 601

V

version difference coexistence 470
VSAM (including IAM - Innovation Access Method™ files) 41
VSAM (SFS) 523
VSAM/SFS 538
VTAM 55

W

WD/z Data Perspective 567
WD/z SCM features 390
WD/z to generate a Web service implementation 516
WD/z to load some data into the DB2 table 604
WD/z Web Services Test Explorer to invoke the CRUD application 605
WD/z Web Services Test Explorer to invoke the LIST application 601
Web enablement components 182
Web project with a simple JSP page 608
Web service 126
Web service binding file 154
Web service client using Faces 607
Web service converter source file 155, 193
Web service enablement components 135
Web service enablement for an existing CICS application 133
Web service enablement for an existing IMS application 179
Web service provider 160
Web service that invokes the J2C Java Bean 516
Web service using WebSphere Developer for System z

163
Web Services 601
Web Services for CICS project 132
Web Services terminology 126
WebSphere Application Server 39
WebSphere Developer for System z 121
WebSphere Developer for System z to work with local COBOL, TXSeries and VSAM (SFS) 523
WebSphere MQ 35
WebSphere Studio Asset Analyzer 107
work style choices 5
working off-line and move to local and use the local compiler 18
working with an UML Model 613
working with Developer for System z and COBOL 5
working with the existing z/OS DB2 table 566–567
working with the Transformation Parameters Editor 575
working with the UML model 566
Workload Simulator 33, 46
workspace 470
WSAA for impact analysis
  identify application dependency 108
  impact analysis 114
WSBIND file to the z/OS 595
WSDL 126
WSDL file and the correlator file to IMS SOAP Gateway 194

X

XML 126
XML conversion 165
XML data from files 42
XML Services for the Enterprise 125
XML tags 42

Z

z/OS Build Forge agent 97
z/OS CICS services 597
z/OS Database Application Generator 563
z/OS Database Application Transformation Editor Layout 576
z/OS HFS directories 595
z/OS interactive remote debug 123
z/OS listings with JES monitor 123
z/OS subsystems 38

Archived





Redbooks

Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM CONFIDENTIAL

Archived

Archived

Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z



Redbooks®

IBM System z, IBM WebSphere, and IBM Rational tools covered

IBM z/OS and IBM System z application building

Scenarios and examples

IBM Rational Developer for System z speeds the development of traditional mainframe, Web, and composite applications and accelerates the development of your Web applications, traditional COBOL and PL/I applications, Web services, and XML-based interfaces.

IBM WebSphere Developer for System z was renamed to IBM Rational Developer for System z.

Rational Developer for System z helps improve the productivity of developers creating and maintaining mainframe applications and Web services before, during, and after the transition to a SOA-based environment, as well as facilitating shorter development cycles that help reduce costs by extracting value from existing applications through reuse and transformation, making them accessible as Web services.

IBM Rational Developer for System z helps to simplify the overall service-flow development process with enterprise service tools and enables developers to rapidly create well-built composite processing that integrates transactional and WebSphere-based environments.

This book has two parts. Part 1 covers Rational Developer for System z and Part 2 covers WebSphere Developer for System z.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7482-00 ISBN 0738432059